ROSノード軽量実行環境mROSのユーザ定義メッセージ型対応に向けた機能拡張

祐源 英俊[1,a)]　高瀬 英希[1,2,b)]　高木 一義[1]　高木 直史[1]

概要：mROSは，ロボットソフトウェアの開発支援フレームワークおよび通信ミドルウェアとして注目されているROSを，組込み機器上で動作可能とする軽量実行環境である．本研究では，mROSにおけるメッセージの型に関する制約を解消することを目的とした，mROSに向けたメッセージのヘッダファイルを自動生成する手法およびmROS通信ライブラリの新たな動作フローを提案する．提案する手法および動作フローにより，ROSで規定されたメッセージの型ならびにユーザが独自定義する型を，mROS上で扱えるようになる．これにより，mROSの汎用性が向上し，ROSおよび組込み機器が協調動作するシステムの開発がより容易にできるようになる．

キーワード：ROS，組込み機器，リアルタイムOS，通信

A functionality expansion of the lightweight runtime environment mROS for the user defined message types

Yugen Hidetoshi[1,a)]　Takase Hideki[1,2,b)]　Takagi Kazuyoshi[1]　Takagi Naofumi[1]

Abstract: mROS is a lightweight execution environment that enables node programs of ROS to be executed on embeded devices. In this research, we aim to remove the constraint on message types that mROS can handle. We propose an approach that generates header files of message types automatically for mROS. We also propose an operation flow of mROS communication library. Proposed approach and flow make it possible for the mROS environment to handle various message types, including primitive types and types defined by users. Therefore, versatility of mROS will be improved, and the development of cooperative system of general-purpose devices using ROS and embedded devices using mROS will become to be easier.

Keywords: ROS, embedded devices, real-time operating systems, communication

1. はじめに

近年，ロボットシステムの開発を支援する様々なソフトウェアフレームワークが注目を集めている．このようなフレームワークの活用により，ロボットシステムの開発を効率的に行うことができる．中でも特に注目を集めているソフトウェアフレームワークとして，ROS（Robot Operating System）[1]をあげることができる．

ROSを活用したシステム開発では，システム内の一部機能を担うプログラムであるノードを組み合わせることによって，所望のシステムを実現する．ROSは，ノード間でメッセージと呼ばれる情報を送受信することができる通信機能も提供している．この通信は，その種類ごとにトピックと呼ばれる通信チャネルを介して行われる．これを利用してノード間の通信を容易に実装することができる．また，ノードの実行ファイルや環境設定ファイルなどは，パッケージとしてまとめ，配布および再利用を行うことができる．パッケージの導入および再利用によってシステム開発にかかる時間を短縮することができる．

ROSにおけるノード間の通信機能はLinuxのミドルウェアとして提供される．したがって，計算能力および消費電力が最小限であり，Linux環境を備えることができない機能仕様の組込みデバイスでは，ROSの実行は不可能である．また，Linux環境を前提としているため，リアルタイム性を確保することが難しい．ここでリアルタイム性と

[1] 京都大学大学院情報学研究科
Graduate School of Informatics, Kyoto University
[2] JST さきがけ
JST PRESTO
[a)] emb@lab3.kuis.kyoto-u.ac.jp
[b)] takase@i.kyoto-u.ac.jp

は，ある処理が一定の時間内に完了できることを指す．

そこで，我々は ROS ノードの軽量実行環境である mROS[2] の研究開発に取り組んでいる．mROS は，計算能力の限られた組込みデバイス上で，ROS のノードとして振る舞うプログラムを動作させることを目指して開発された実行環境である．リアルタイム OS を実行可能な組込みデバイス上で動作し，分散環境におけるホスト PC 上の ROS ノードと双方向の通信を行うことが可能である．mROS のアプリケーションのプログラミングモデルは，ROS ノードプログラムとの互換性を意識して設計されている．このため，既存の ROS パッケージ資産を mROS へと移行させることが可能である．

現時点で mROS が対応しているメッセージの型は文字列型に限定されている．そのため，mROS アプリケーションと通信を行う ROS ノードは，通常の ROS ノードとの通信とは異なるプログラム実装が要求されている．また，この制約に起因して，mROS アプリケーションと ROS ノードプログラムのコード記述様式の相違点が存在する．したがって，既存の ROS ノードプログラムを mROS アプリケーションへ移植するためには，この相違点に応じた改変が必要であると考えられる．これにより，mROS の利用用途が限定され，ROS を実行する汎用デバイスと，mROS を実行する組み込みデバイスとが協調動作を行うシステムの開発が煩雑なものとなってしまっている．

そこで本研究では，この制約を解消し，mROS の汎用性を向上させることを目指す．すなわち，mROS において，文字列型以外のメッセージ型が利用できるようにする手法を提案する．加えて，任意のメッセージ型への対応に伴った mROS 通信ライブラリの動作フローを設計する．

提案手法は，メッセージの型情報を定義するヘッダファイルを mROS アプリケーションにおいて利用できる形で生成する．メッセージごとにヘッダファイルを定め，その内部にメッセージの構造など型に関わる情報およびその型に固有な処理を含める．ヘッダファイル生成の際には，ROS システム内に存在するヘッダファイルを利用する．加えて，出力される mROS アプリケーション向けのヘッダファイルは，組込み機器上での実行を想定した軽量な実装とする．

提案手法に伴う mROS 通信ライブラリの動作フローは，通信処理の際に，メッセージ型に共通な処理から型固有の処理を呼び出すというものである．型固有の処理の内容は，型に応じた適切な内容を共通のインタフェースで呼び出せるように，前述の手法により生成される．これにより，メッセージ型に共通なコードを改変することなく，あらゆるメッセージ型の処理が行える．

本研究により，mROS の汎用性が向上する．これにより，mROS を活用した多様なシステムの開発が可能となる．また，mROS を活用するシステム開発がより容易になる．特に，ROS ノードプログラムの開発者は，少ない学習コストで mROS アプリケーションの開発ができるようになる．

本稿の構成は次の通りである．2 章では，ROS および mROS について説明し，関連研究を紹介する．3 章では，本研究の実現目標について検討する．4 章では，メッセージ型のヘッダファイル生成手法を提案する．5 章では，mROS 通信ライブラリの動作フローを提案する．6 章では，提案手法の有効性を議論する．7 章では，本研究のまとめと今後の展望を述べる．

2. 準備

2.1 ROS

ROS（Robot Operating System）[1] は，ロボットシステムの開発を支援するライブラリおよびツール群である．主な機能の一つとして，ノード間の通信を提供するミドルウェアとしての機能を挙げることができる．

ROS を活用して実現されるシステムは，システム上の機能単位であるノードと，システム全体を管理するマスタから構成される．マスタはシステム内のノード情報を管理し，システム内のノードにその情報を提供することによってノード間通信を支援する．

ROS におけるデータの送受信は，出版購読型通信モデル（Publisher-Subscriber Messaging Model）に基づいて設計されている．このモデルでは，まず送受信する情報を目的および型ごとにトピックと呼ばれる通信チャネルを用意する．そして各トピックを介して通信情報であるメッセージの送信を行うパブリッシャ（Publisher）と，受信を行うサブスクライバ（Subscriber）の二者間で通信を行う．したがって，ROS においてもノード間の通信は必ずトピックを介して行われる．

ROS を活用して構築した，トピックを介した通信を行うシステムの例を図 1 に示す．図 1 において，楕円がノード，矩形がトピックを表している．また，トピックを表す矩形の内部にある鉤括弧で囲まれた部分がトピック名，その下の文字列がそのトピックの型を表している．ただし，図 1 右上の「モータ値型」および左下の「画像型」は，開発者が目的に応じて定義したユーザ定義型を表している．これらの型名の下にある矩形内に列挙される型が，そのユーザ定義型の持つ内部変数を表す．

メッセージの型は，数値および文字列など基本データ型（プリミティブ型）を内部変数として組み合わせることによって定義される．メッセージ内の変数の型に，プリミティブ型以外の型を利用することも可能である．例えば図 1 におけるユーザ定義型であるモータ値型を内部の変数に持つ，新たな変数型を定義することもできる．以降，このような構造を持つメッセージ型を入れ子構造の型と呼ぶ．また，任意の 1 つの型の変数を要素に持つ配列を利用する

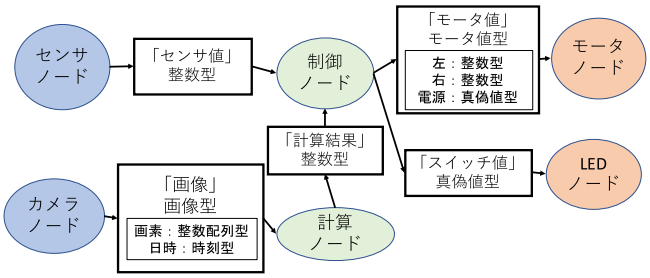

図 1　ROS を活用したシステムの例
Fig. 1　Example of system utilizing ROS

ことも可能である．

ユーザ定義の型を作成する際は，ROS 内で定められたフォーマットに従い型の定義ファイル（`.msg` 形式）を作成する．定義ファイルは，内部変数の型および名前を列挙する形式で記述される．そしてその定義ファイルをもとに，メッセージ型利用のためのプログラムを，ROS のビルドシステムを用いて生成する．C++を用いてシステムを開発する場合，メッセージ型の定義ファイルから生成されるプログラムは，メッセージ型をクラスとして定義するヘッダファイルである．

2.2　mROS

mROS[2][3] は，我々が 2018 年から研究開発を行っている ROS ノードの軽量実行環境である．Linux を実行できない組込みデバイス上において，ROS ノードとして振る舞うプログラムを実行させることを目的としている．対象とする組込みデバイスは，TCP/IP プロトコルスタックの動作が可能かつ消費電力の小さいミッドレンジクラスのデバイスである．また，ROS ノードと mROS アプリケーションの間の互換性を高くするため，mROS アプリケーションのプログラミングモデルは ROS ノードのプログラミングモデルに近いものを採用している．

mROS の通信機能は，mROS アプリケーションを実行するユーザタスクを含むタスク群によって提供される．メッセージ送信時は，以下のようなフローで動作する．

(1) ユーザタスクによって送信されるメッセージの内容が共有メモリに書き込まれ，パブリッシュタスクのデータキューにメッセージ情報がキューイングされる．
(2) データキューへのキューイングをもってパブリッシュタスクが起動する．
(3) キューの情報を元に共有メモリに書き込まれたメッセージの内容をビット列にシリアライズし，送信する．

また，メッセージ受信時は以下のようなフローで動作する．

(1) パブリッシャタスクとの接続が確立された状態で，サブスクライバタスクは周期的に実行され，メッセージを待つ．
(2) 購読するトピックに対しパブリッシャからのメッセージ送信があった場合，ビット列を受け取り，その中からメッセージに該当する部分を抽出し，共有メモリに書き込む．
(3) 共有メモリに書き込まれた内容をデシリアライズし，メッセージのオブジェクトを生成する．
(4) サブスクライバノードと紐づけて登録されたコールバック関数への関数ポインタを取得する．引数に前手順で作成したオブジェクトを与え，ポインタの指すコールバック関数を実行する．

2.3　関連研究

本節では関連研究として，組込みデバイスを活用した ROS システムの構築例および ROS を活用した分散システムを構築している研究を紹介する．

文献 [4] は，自動制御で動作しエリアの探索および目的地への移動を行う車椅子についての研究である．ROS のシステムに Arduino を活用することで低コストかつ柔軟なシステムを提案している．文献 [5] では，モバイルロボットとクラウドを ROS を通じて協調させ，VSLAM のボトルネック緩和を目指したシステムを提案している．ROS を活用し，SLAM に必要な計算をクラウド上のノードに負荷分散させている．いずれも，情報の収集，送受信，および，受信情報に基づく動作のみを行う，いわゆるエッジ機器がシステム構成中に存在している．両文献中においてはノート PC を採用しているが，ここを mROS を実行する組込み機器に置き換えられると考えられる．これにより，エッジ機器の小型化および省電力化ができる．

続いて，ROS と組込み機器の通信および協調を行う研究を紹介する．文献 [6] は，組込み機器上において ROS ノードを実行する rosserial についての公式ドキュメントである．rosserial を活用すれば，他 ROS ノードとの通信行うプログラムを組込み機器上で実行できる．ただし，利用できる通信方式はシリアル通信のみである．このため，ネットワークを用いる場合に比べ，伝送速度や伝送範囲に劣る．文献 [7] は，ROS と，リアルタイムで動作するソフトウェアフレームワークである LUNA との通信を実現するブリッジについて提案および実装を行っている．LUNA と直接の通信を行う LUNA ブリッジノードが ROS トピックに対しメッセージの送信および受信を行うことにより，間接的に ROS ノードと LUNA アプリケーションの通信を実現している．しかし，二者間の通信にはブリッジノードを挟まねばならない．また，ROS および LUNA という 2 つのフレームワークを用いているため，学習コストが高く，コードの再利用性が低い．文献 [8] では，IEC 61131-3 に準拠し，ソフトウェア PLC の主要な製品である CODESYS と ROS の通信インタフェースを提案している．両者間のインタフェースとして，ROS メッセージの送受信および CODESYS の共有メモリへのアクセスを行う ROS ノードを実装している．これも，中間ノードを含むシステム構造

が必要である．また，ユーザ定義メッセージ型や配列については，対応可能性が示唆されているのみである．

3. 実現方針

本章では，本研究の目的を実現するための方針について検討する．

ROS ノードとの通信を実現するために mROS 通信ライブラリが行うべき処理は，メッセージ型に固有の処理および全てのメッセージ型に共通な処理の 2 種類に分けて考えることができる．ここでメッセージ型固有の処理とは，メッセージの構造の違いによって必要となる処理を指す．

そこで，提案する mROS 通信ライブラリ内では，メッセージ型固有の処理は全て，メッセージ型に共通の処理から呼び出されることにより実行されるようにする．これは，共通の処理から呼び出される処理をメッセージ型に応じて変更することで，通信で扱うメッセージ型を容易に変更可能とすることを目的としている．

この処理仕様をより具体的に検討する．まず，メッセージ型固有の処理をメッセージ型ごとに定めるようにする．このとき，メッセージ型一つに対し一つのファイルを作成し，型に必要な処理をそのファイル内に含める．これは，生成の際にメッセージ型に共通な処理，および他のメッセージ型に対応する処理に影響を与えないようにするためである．さらに，これらの処理は任意の型に対し，全ての型に共通のインタフェースで呼び出せるようにする．これは，扱うメッセージ型を変更する際に，メッセージ型に共通な処理を変更する必要が発生しないようにするためである．

さらに，上述の処理仕様に基づくように mROS 通信ライブラリの具体的な動作フローを設計する．ROS ノードとの通信のために mROS 通信ライブラリが行うべき動作は，出版および購読トピックの登録，メッセージの送信，メッセージの受信の 3 つに大別される．このそれぞれについて，実現のための方針を検討する．

出版および購読ノードの登録時は，ノード名，トピック名，および，トピックで通信するメッセージの型名など必要な情報をマスタへ送信する．このため，この必要な情報を返すメッセージ型固有の処理を定め，この処理を呼び出すことによりこれらの値を取得できるようにする．

メッセージ送信時には，メッセージはビット列に変換され，メッセージのサイズを付加されたうえで共有メモリに格納される．この動作の実現のために，メッセージ型に応じたシリアライズ処理，および，メッセージのサイズ計算処理をメッセージ型固有の処理とし，それらが適切に呼び出されるようにする．

メッセージ受信時は逆に，ビット列を受信し，メッセージの型に応じたデシリアライズを行う．この動作の実現のために，メッセージのデシリアライズ処理をメッセージ型固有の処理とし，それが呼び出されるようにする．

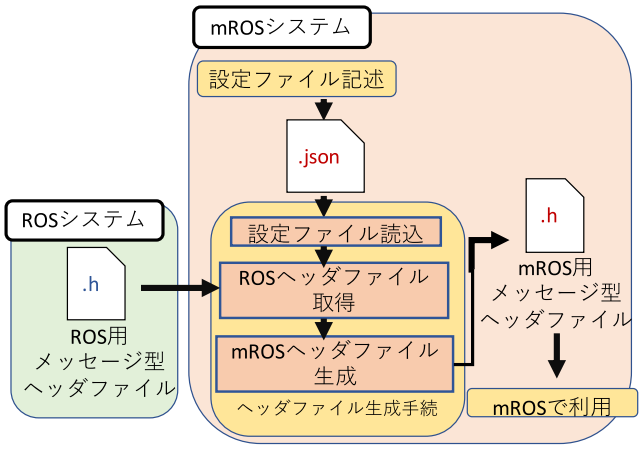

図 2 ヘッダファイル生成手法
Fig. 2 Operation flow of header file generation

4. mROS 用メッセージヘッダファイルの生成手法

前章で述べた処理仕様を満たすよう，メッセージ型固有の処理をヘッダファイルとして生成する手法を提案する．図 2 に，提案する手法のフローを示す．図の左半分の緑色部分が ROS のシステムに存在する部分であり，右半分の橙色部分が本研究で提案する mROS システム内でのフローを表す．mROS システム内において，開発者が行う手順を黄色，自動で行う手続き動作を赤色で表している．

開発者は設定ファイルに，ROS ワークスペースのディレクトリ，および，mROS アプリケーション内で利用するメッセージ型などの情報を記述する．続いて，設定ファイルに記述された ROS のワークスペースのディレクトリを基準に，利用するメッセージ型に対応する ROS システム内のメッセージ型ヘッダファイルを参照する．そしてこれを元に，組込みデバイス上での動作を想定した，mROS 専用のメッセージ型のヘッダファイルを生成する．

まず，ROS システム内のヘッダファイルから，型名，型の定義，および通信の整合性検証のためのハッシュ合計値を取得する．これらの値は，生成するヘッダファイル内にそのままコピーして利用する．さらに，メッセージ型の定義から型の内部変数の情報を取得し，それをもとにメッセージサイズの計算を行う処理，シリアライズ処理，および，デシリアライズ処理を生成する．これらの処理は，前章で述べた通りメッセージの送受信時に必要である．

シリアライズおよびデシリアライズのための関数は，メッセージ型内に定義される各変数に対し，変数の型に応じた処理を順番に行うように生成する．メッセージ型の内部変数は，プリミティブ型，他のメッセージ型，`Header` 型およびその配列のいずれかである．ゆえに，任意のメッセージ型は，プリミティブ型，`Header` 型，および，それらの配列の組み合わせで表現することができる．これは，メッ

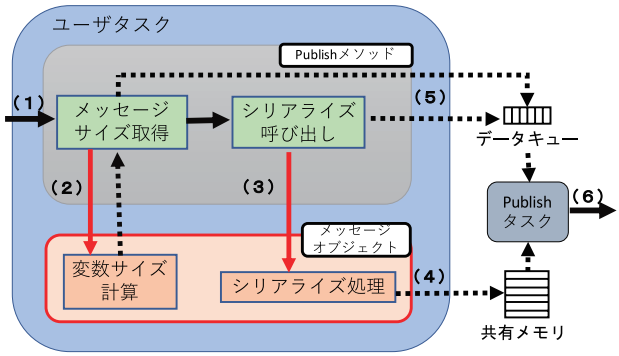

図 3　メッセージ送信時の動作フロー
Fig. 3　Operation flow of message publishing

セージ型が入れ子構造であっても，その内部変数であるメッセージ型を再帰的にたどればいずれは上の 3 種類の変数の組み合わせで表現できるからである．したがって，プリミティブ型，`Header` 型，およびそれらの配列に対する処理を生成できるならば，任意のメッセージ型に対し，そのシリアライズ処理およびデシリアライズ処理を行う関数を生成することができる．各処理の具体的な内容については，次章以降において述べる．

5.　mROS 通信ライブラリの動作フロー

提案する mROS 通信ライブラリの具体的な動作フローについてそれぞれ説明する．

5.1　出版および購読ノード登録時

出版および購読ノードを登録する際の動作フローを以下に示す．

(1) 扱うメッセージ型の情報を，関数呼び出し時の引数などから受け取る．
(2) 必要な情報を，その情報を返すメッセージ型固有の処理を呼び出すことにより取得する．
(3) 取得した情報を XMLRPC 通信でマスタに送信する

ノードの登録動作は，ノードおよびトピックに関する情報をマスタに送信する．この情報は，前章の手法により ROS システム内のファイルから抽出された値を利用し，この値を返すため専用の処理を呼び出すことにより取得する．これは，値の取得方法をメッセージ型の種類によらない共通のものとするためである．

5.2　メッセージ送信時

提案するメッセージ送信時の動作フローを図 3 に示す．図中において表される，動作の説明を以下に示す．

(1) 送信するメッセージをオブジェクトとして受け取る．
(2) 引数として受け取ったオブジェクトに対し，メッセージサイズ計算の関数を呼び出す．
(3) 同オブジェクトに対し，共有メモリの所定箇所のポインタを引数として，シリアライズの関数を呼び出す．

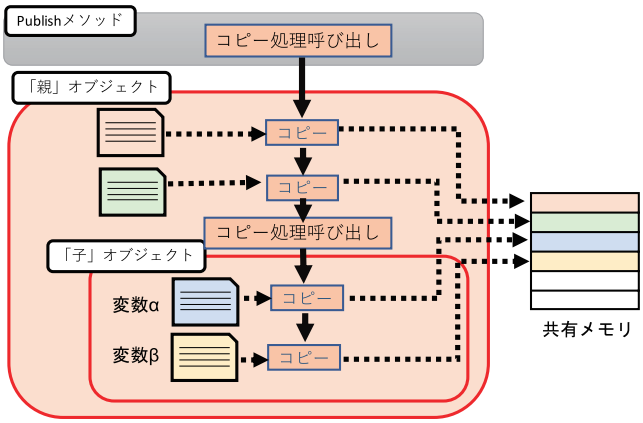

図 4　入れ子メッセージの処理動作フロー
Fig. 4　Operation flow of processing of nested messages

(4) 呼び出されたシリアライズの関数は，引数として渡されたポインタの指すメモリ領域に，メンバ変数の値をコピーする．その後，コピーした値のサイズに応じ，ポインタを動かす．
(5) 前手順でメッセージの値をコピーした領域の先頭ポインタ，および，手順 2 で返されたメッセージ全体のサイズを Publish タスクのデータキューに格納する．
(6) データキューにデータが格納されると，Publish タスクが直ちに実行される．Publish タスクはデータキューから受け取る情報を元に送信データを構成し，送信する．

図中の灰色領域はメッセージ型に共通の処理（以下，共通処理と呼ぶ），図中の橙色領域はメッセージ型に固有な処理（以下，個別処理と呼ぶ）を示す．またこの橙色領域に相当するプログラムは，前章に述べた手法により生成されたものを利用する．図中の赤矢印は，処理の呼び出しを表している．この赤矢印が示す通り，共通処理が個別処理を呼び出す動作フローとなっている．メッセージ内容の共有メモリへの書き込みも個別処理が行う．個別処理を呼び出した共通処理へは，メッセージのサイズのみが返される．これは，扱われるメッセージ型に関わらず整数型の値である．このようにして，共通処理と個別処理との結合度を下げた．これにより，共通処理には変更を加えず，個別処理を所望のメッセージ型に関するものに差し替えるだけで，その型に関する処理が行える．

またこのフローにより，入れ子構造を持つメッセージ型の処理も実現することができる．ここでメッセージ型オブジェクトを親，および，その型内部の 1 変数型として利用されているメッセージ型オブジェクトを子とする．この「親」をシリアライズする際の動作フローを例として説明する．この動作を表す図を，図 4 に示す．

図が示す通り，`publish` メソッドが親オブジェクトに対して行うシリアライズ処理呼び出しと同様の形式で，親オブジェクトが子オブジェクトのシリアライズ処理を再帰的

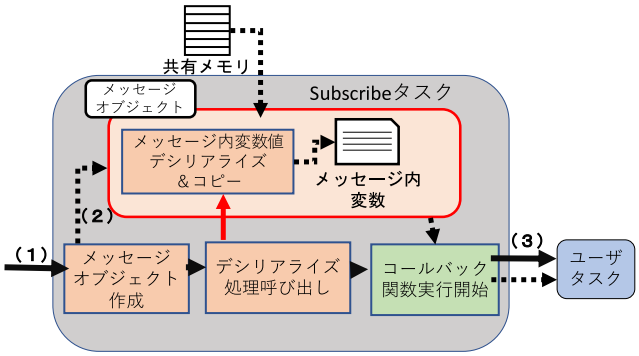

図 5　メッセージ受信時の動作フロー
Fig. 5　Operation flow of message receiving

に呼び出す．また，親オブジェクトと子オブジェクトで，共有メモリ領域を指すポインタを共有する．これにより，入れ子構造を持つ型のシリアライズ処理を実現する．

5.3　メッセージ受信処理

提案するコールバック処理に関連する一連の動作を表す図を図 5 に示す．また，行われる動作の詳細を以下に示す．

(1) 受信したメッセージの型に対応するオブジェクトを生成する．
(2) 作成したオブジェクトに対し，受信したデータが格納される共有メモリへのポインタを引数として，デシリアライズ関数を呼び出す．デシリアライズ関数内では，ポインタが示す値をメッセージオブジェクトに順に格納する．
(3) メッセージオブジェクトへのポインタを引数とし，関数ポインタの指す関数を実行する．

図中の「コールバック関数実行開始」と書かれた矩形に相当する処理は，デシリアライズにより変数値が適切に設定されたメッセージオブジェクトを，所定のコールバック関数に渡す動作である．この動作に向けて破線矢印が伸びているが，これはこの動作の前々手順で作成されたメッセージオブジェクトを利用するということを意味しており，デシリアライズ関数呼び出しの結果値が返されてその返り値を利用するということを表しているわけではない．送信時同様，デシリアライズ処理とそのほかの処理との結合度が低いため，メッセージ型に対し柔軟な処理が行える．また，`publish` メソッドのシリアライズ処理同様，再帰的な手続き呼び出しにより，入れ子構造を持つメッセージ型のデシリアライズも可能である．

6.　評価

本章において，提案の有効性を検証し，またその有用性について議論する．

6.1　評価環境

提案するヘッダファイル生成手法を，ツールを用いて実現した．また，提案する動作フローを mROS 通信ライブラリ内に実装した．

評価環境において，mROS アプリケーションは GR-PEACH 上で実行した．GR-PEACH は TOPPERS/ASP カーネル [9] および mbed ライブラリが利用可能なデバイスである．また，有線 LAN ケーブルによりネットワーク接続が可能である．

GR-PEACH が接続するローカルエリアネットワーク内に，Ubuntu 16.04 LTS を稼働するノート PC を無線 LAN によって接続した．ノート PC において ROS のノードおよびマスタを実行する．mROS アプリケーションは，これらのプロセスと通信を行い，メッセージの送受信を行なった．ROS ディストリビューションは Kinetic を使用した．

```
ros::init(argc,argv,"mros_node");
ros::NodeHandle n;
ros::Publisher pub = n.advertise<
mros_test::PersonalData>("mros_str",1);
ros::Rate loop_rate(5);
mros_test::PersonalData msg;
msg.first_name = "Phil";
msg.last_name = "Woods";
msg.age = 83;
msg.score = 100000;
while(1){
  wait_ms(1000);
  pub.publish(msg);
  msg.score ++;
}
```

図 6　mROS アプリケーションから `publish` を行うコード
Fig. 6　Code to publish messages from mROS

6.2　動作検証

新たに利用可能となったユーザ定義メッセージ型を利用し，mROS アプリケーションと ROS ノードの間で相互に通信を行う．これにより，ユーザ定義メッセージ型を含むあらゆるメッセージ型による通信が，提案に基づく mROS の機能拡張により可能となったことを検証する．検証は，プリミティブ型の各型とユーザ定義型について行った．その結果のうち，ユーザ定義型についての結果を以下に掲載する．検証に利用したメッセージ型は，以下の内部変数を含む `PersonalData` 型を使用した．

- 文字列 `first_name`，`last_name`
- 符号なし 16 ビット整数 `age`
- 32 ビット整数 `score`

はじめに，mROS アプリケーションからの送信について検証する．検証に利用した mROS アプリケーションのコードを図 6 に示す．

実行した結果の ROS ノードの出力を図 8 に掲載する．図 6 の 7-10 行目で入力された内容が表示されていることが見て取れる．また，同コード 3 行目において連結された

```
void Callback(mros_test::PersonalData::Ptr msg){
 syslog(LOG_NOTICE, "I heard a msg from ros host");
 string name = msg->first_name + " " + msg->last_name;
 syslog(LOG_NOTICE, "name:%s",name.c_str());
 syslog(LOG_NOTICE, "age:%u",msg->age);
 syslog(LOG_NOTICE, "score:%u",msg->score);
}
void usr_task2(){
 #ifndef _USR_TASK_2_
 #define _USR_TASK_2_
 ros::init(argc,argv,"mros_node2");
 ros::NodeHandle n;
 ros::Subscriber sub = n.subscribe("test_msg",1,Callback);
 ros::spin();
 #endif
}
```

図 7　mROS から subscribe を行うコード
Fig. 7　Code to subscribe from mROS

```
257]: Phil Woods
969]: age:83
829]: score: 100074
490]: I heard from mROS
018]: Phil Woods
044]: age:83
569]: score: 100075
457]: I heard from mROS
665]: Phil Woods
080]: age:83
686]: score: 100076
342]: I heard from mROS
```

図 8　ROS ノードでの受信結果
Fig. 8　Result of message recieving on ROS node

```
SUB_TASK: 1a 0 0 0
SUB_TASK: msg size [26]B
SUB_TASK:data length [30]
I hear msgs from ros host
name:Charlie Parker
age:32
score:100175
SUB_TASK: 1a 0 0 0
SUB_TASK: msg size [26]B
SUB_TASK:data length [30]
I hear msgs from ros host
name:Charlie Parker
age:32
score:100176
```

図 9　mROS での受信結果
Fig. 9　Result of message recieving on mROS

文字列が正しく出力されているため，first_name および last_name が mROS アプリケーションにおいて正しく文字列として解釈されていることが分かる．さらに，score も受信メッセージごとにインクリメントされていることから，同コード 14 行目において score の値をインクリメントする操作内容が反映されていることがわかる．これは mROS において score が，int として正しく扱われていることを示す．

続いて，mROS アプリケーションにおいての受信を検証する．検証に使用した，メッセージの受信を行う mROS ノードのコードの一部を図 7 に示す．

ROS ノードには，first_name を「Charlie」に，last_name を「Parker」に変更し，かつ図 6 に示すプログラムと同じ動作をする実装を行なった．実行した結果の mROS アプリケーションの出力を図 9 に掲載する．

図 9 から，ROS ノードにおいて設定された値と同一の値が表示されていることがわかる．first_name および last_name に関しては，7 の 3 行目において両文字列を連結される操作が，出力に反映されている．score に関しては，メッセージ毎に値がインクリメントされており，ROS ノード上の実装が反映されていることがわかる．これらのことより，mROS アプリケーションにおいて，メッセージを正常に受信，解釈，および利用できていることが分かる．

6.3　有用性の議論

本研究の有効性ついて考察し述べる．本研究に基づいた機能拡張を mROS に対して施した場合の，性能等の向上について議論する．

まず，通信の際に文字列型への変換を伴う必要がなくなったことにより，通信時のオーバヘッドが削減されたと言える．これは，通信時の無駄な型の変換および逆変換が不要となったことにより，アプリケーションを組込みデバイス上においてより効率よく動作させることが可能となったからである．

続いて，扱う情報に対し適切な型の選択ができていれば，通信に必要な符号数を削減できるといえる．数値の送受信を行う際に文字列として送信する場合，極端なケースを除き，送信する値に対する符号の数が多くなる．さらに ROS の仕様上，可変長メッセージである文字列を送信する際は，そのメッセージ長を伴うことが必要である．この問題点を，本研究の成果によって解消し，伝送する情報に対し効率の良い送信方法の選択が可能となった．

このことについて，32 ビット浮動小数点数型の変数を 3 つもつメッセージ型を使用する場合を例として説明する．このメッセージのボディ部分のサイズは，大きさ 4Byte の浮動小数点数を 3 つ含むため，12Byte となる．続いて，文字列として送信する場合を考える．32 ビット浮動小数点と同等の精度のために，6 桁の数を表示すると想定する．この場合，数値一つあたりに，小数点も含め 7 文字が必要である．文字列型のメッセージの場合，1 文字に 1 バイト必要なため，3 数の合計サイズは 21Byte になる．文字列型は可変長変数のため，32 ビット整数であらわされる変数長を付加しなければならない．このため，ボディ部分のサイズは 25Byte となる．さらに，区切り文字を挿入する場合などにはさらにサイズが大きくなる．このように，数値型を送信する場合などにおいては特に，送信メッセージのサイズを削減することが可能である．

mROS アプリケーションにおいて利用するメッセージ型を設定ファイルに記述することにより，ビルド時にビルドツールからその型が特定可能となった．したがって，利用されるメッセージ型に合わせた，mROS アプリケーションが必要とする共有メモリサイズの最適化が可能となる．この有効性を検証するため，この最適化を行う機能を実装した上で mROS のビルドを実行する実験を行う．ビルドによって生成された mROS ライブラリのプログラムサイズを確認し，その有用性を検証する．

評価において，浮動小数点数型の変数を 3 つ持つ型を用いた．先述した，この型を使用するメッセージのボディ部

分のサイズは 12Byte である．また，ビルドにより生成される `libmros.a` を評価対象とし，Linux の size コマンドを用いてデータサイズを取得した．`libmros.a` は，mROS 通信ライブラリのファイルにあたる．

size コマンドにより取得した mROS 環境のプログラムサイズを表 1 に示す．なお，表 1 および以下の記述において，「新実装」は本文献中の提案に基づいた mROS 実装を示し，「旧実装」は文献 [2] および [3] で示される提案に基づいた mROS の実装を指している．

表 1 mROS 通信ライブラリのオブジェクトサイズの比較
Table 1 Comparation of the program size of mROS communication library

	text[Byte]	data[Byte]	bss[Byte]	dec[Byte]
旧実装	69,464	28	2,097,319	2,166,811
新実装	69,460	28	1,048,739	1,118,281

表に示す結果より，bss セクションのサイズが大幅に削減されたことがわかる．旧実装においては，送信データのタスク間通信のため 512KByte の領域を 2 つ，合計約 1MByte の領域を確保していた．これは，カメラからの QVGA フォーマット画像データなど，サイズの大きいメッセージを扱うことを考慮したためである．本研究の成果を活用することで，この領域をアプリケーション内において用いるメッセージ型のサイズに合わせて縮小することが可能となった．可変長メッセージの上限値を適切に設定することにより，さらなる最適化ができると考えられる．

本研究の機能拡張により，`publish` および `subscribe` メソッドの呼び出し時にトピックの型を指定することが可能となった．また通信の際に，通信メッセージを文字列に変換する記述を追加する必要も無くなった．このことにより，mROS アプリケーションプログラミングモデルを，ROS ノードプログラムのそれにより近づけることが可能となった．通信動作を行うための mROS のコードから，重要な部分を抜粋したものを図 6 および図 7 に示す．これらのコードは，ROS ノードプログラムとほぼ同じ関数呼び出しインタフェースを用いて記述されている．このことにより，既存 ROS ノードのプログラムから mROS アプリケーションへの移植がより容易になったといえる．

7. まとめ

本研究では，ROS ノードの軽量実行環境である mROS の汎用性向上を目的とし，通信メッセージ型に関する制約を解消するために，メッセージ型ヘッダファイル生成手法および mROS 通信ライブラリ動作フローを提案した．提案した手法は，メッセージ型に対応するヘッダファイルを，ROS システム内から取得した情報によって生成するものである．また，動作フローは任意のメッセージ型の処理に対応するため，ヘッダファイル内に定義されたメッセージ型固有の処理を呼び出す構造とした．

提案内容を mROS に実装し，提案の正当性を検証した．検証により，`Time` 型および `Duration` 型を除くプリミティブ型，その配列，および，ユーザ定義のメッセージ型が，mROS において利用可能となったことを確認した．利用できないものとして挙げられた型は，本文献中の提案手法により実現が可能であると考えられるが，時間の都合上実装に至らなかったものである．

加えて，この機能拡張の有用性についての議論も行なった．

本研究で行った提案により，mROS を活用したシステム開発がより容易となった．また，mROS の汎用性も向上し，多様なシステム開発において活用することができるようになった．

今後の課題としては，利用可能に至らなかったメッセージ型の対応，同一デバイス内でのノード間通信についての検証，他フレームワークを利用した場合との比較などの詳細な性能検証などがあげられる．

謝辞 本研究の一部は，JST さきがけ JPMJPR18M8 および JSPS 科研費 18K18024 の支援を受けたものである．

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, *ICRA workshop on open source software*, No. 3.2, pp. 5 (2009).
[2] Hideki, T., Tomoya, M., Kazuyoshi, T., and Naofumi, T.: Work-in-Progress: Design Concept of a Lightweight Runtime Environment for Robot Software Components Onto Embedded Devices, *2018 International Conference on Embedded Software (EMSOFT)*, pp. 1–3 (2018).
[3] 森 智也, 高瀬 英希, 高木 一義, 高木 直史：mROS：組込みデバイス向け ROS ノード軽量実行環境 第 *220* 回システム・アーキテクチャ研究発表会（デザインガイア *2017*）, pp. 1–3 (2018).
[4] Zhengang, L., Yong, X., and Lei, Z.: ROS-Based Indoor Autonomous Exploration and Navigation Wheelchair, *2017 10th International Symposium on Computational Intelligence and Design (ISCID)*, Vol. 2, pp. 132–135 (2017).
[5] Patrick, B., Mohan, M., Paul, R., John, J. P., Mo, J., and Lutcher, B.: Cloud-Based Realtime Robotic Visual SLAM, *2015 Annual IEEE Systems Conference (SysCon) Proceedings*, pp. 773–777 (2015).
[6] Ferguson, M.: rosserial, http://wiki.ros.org/rosserial.
[7] Bezemer, M. M. and Broenink, J. F.: Connecting ROS to a real-time control framework for embedded computing, *Emerging Technologies & Factory Automation (ETFA)*, Vol. 2015 IEEE 20th Conference on, IEEE, pp. 1–6 (2015).
[8] Tiago, P., Rafael, A., and Germano, V.: Bridging Automation and Robotics: an Interprocess Communication between IEC 61131–3 and ROS, *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)*, pp. 1085–1091 (2018).
[9] TOPPERS プロジェクト：TOPPERS/ASP kernel, https://www.toppers.jp/asp-kernel.html.