Vol.2017-SLDM-181 No.42
Vol.2017-EMB-46 No.42
2017/11/8

mROS：組込みデバイス向けROSノード軽量実行環境

森 智也[1]　高瀬 英希[1]　高木 一義[1]　高木 直史[1]

概要：近年のサービスロボット開発では，開発支援フレームワークである ROS（Robot Operating System）の活用が注目されている．ROS はソフトウェア・コンポーネント間の通信層を提供するミドルウェアであり，Linux 上での動作を想定している．そのため，高機能かつ消費電力の大きいデバイスを採用する必要がある．．そこで本研究では，消費電力の小さなロボットシステムの実現を目的とし，リアルタイム OS および TC/IP スタックを使用して，組込みデバイス上で ROS ノードを実行可能な軽量な環境を提案する．提案する実行環境を使用することでコンポーネントの省電力化およびリアルタイム性の確保が容易になると考えられる．

キーワード：ROS，リアルタイム OS，組込みマイコン，ロボット

1. はじめに

近年，様々な場面でモバイルロボットの需要が高まっている．特に，災害救助ロボットやガイドロボット，掃除ロボットなど，社会生活においてサービスを提供するためのロボットの開発が活発になっている．これらのロボットはこれまで開発されてきた産業用ロボットとは異なり，外部電源からのエネルギー供給ではなく，内部電源のエネルギーによって動作するという特徴がある．また，高度かつ多機能なサービスを要求されることが多く，一つのロボットシステムで様々な制約のもと多くの機能を実現しなくてはならない．そのため，ロボットが提供するサービス品質の向上には，省電力かつ多機能の実現が必要になる．

ROS（Robot Operating System）[1] はロボットシステムの設計生産性の向上に資する開発支援フレームワークとして注目されている．ROS ではソフトウェア部品をノードとして表現し，複数のノードを組合わせることでロボットシステムを実現する．実行ファイルと環境設定ファイルなどをまとめた ROS パッケージが数多く公開されており，それらを活用することで効率的なコンポーネント指向開発が可能になる．また，ROS はノード間での通信層を提供するミドルウェアとしての役割も担っている．しかし，ROS は Linux 上での動作を想定した設計になっており，高機能かつ消費電力の大きなデバイスを採用する必要がある．さらに，ROS で構築されたロボットシステムはリアルタイム性を確保することが難しい．

[1] 京都大学
Kyoto Uniersity

本研究ではロボットシステムの省電力化を目的とする．消費電力の小さな組込みデバイスを ROS ロボットシステムで活用することで省電力化を実現できると考えられる．しかし，資源や性能の制約により Linux が搭載することができないため，ROS を動作させることができない．

そこで，本研究では組込みデバイス向け ROS ノード軽量実行環境である mROS を提案する．mROS は省電力な組込みデバイスにおける対 ROS システム用通信ライブラリを提供する．ROS システムにおいて使用される XML-RPC および TCPROS の通信プロトコルをサポートすることで，ROS システムと通信を行えるようにする．現在 TOPPERS/ASP カーネル [2]，ARM mbed ライブラリを使用して GR-PEACH を対象に mROS の開発を行っている．

提案する mROS によって，従来であれば ROS システムに組込むことができなかった消費電力の小さな組込みデバイスを活用することが可能になる．また，mROS では ROS の API を使用してプログラムを記述できるため，既存の ROS パッケージ資産を再利用することも可能になると考えられる．リアルタイム OS と TCP/IP プロトコルスタックが搭載された組込みデバイスを対象とし，それらの特性を使用したアプリケーションを設計することもできる．そのため，リアルタイムシステムの設計も容易になると考えられる．

2. ROS

2.1 概要

ROS は，ロボットソフトウェアの開発フレームワークであり，開発されたソフトウェアの再利用性を高めることを

© 2017 Information Processing Society of Japan

目標として開発された．

ROS では，機能を実現するプログラムをノードという一つの単位として表現し，複数のノードを用いてロボットシステムを実現する．ROS はノード間における通信層を提供するミドルウェアであり，ソフトウェア開発や移植を簡単化している．ノード間の通信は，トピック名でデータの種類を分別し，出版されるメッセージを購読する出版購読通信で行われる．サーバクライアント方式の通信も可能である．

図 1 にトピックによるノード間での出版購読通信のモデルを表す．扱うトピック名が同一であれば，ROS ロボットシステムにおいてハードウェアとそのノードを他のハードウェアと対応するノードに容易に変更できる．プログラム実行時には，ノードを管理する ROS マスタとなる roscore を立ち上げることにより，それぞれのノードの名前空間や変数空間を管理する．

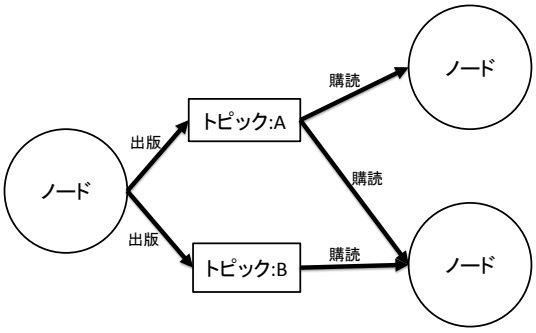

図 1 ROS におけるノード間の出版購読通信モデル
Fig. 1 Publish/subscribe communitacion model in ROS

2.2 ROS システムにおける通信プロトコル

ROS システムでは，ノード間およびノードと ROS マスタ間との通信では以下のプロトコルが使用されている．

- XML-RPC 通信
  XML-RPC は，通信データを XML 形式でエンコードし，HTTP で転送することで手続き呼び出しを行う RPC プロトコルである [10]．ROS システムでは XML-RPC を用いてノードと ROS マスタ間およびノード間での手続き呼び出しを行う．
- TCPROS データプロトコル
  TCPROS は ROS のデータ転送のためのトランスポートレイヤであり，pub/sub 通信およびサーバクライアント通信のデータ転送で使用される．TCPROS データプロトコルでは，データ内容を識別するためにノードの設定に対応して特有のフィールドが規定されている．それぞれのフィールドに対応するデータに対して，長さを示す 4B ビット列を付与したものを一つのメッセージとして転送する．接続が確立した際に，一度だけノード情報をもつコネクションヘッダを送受信しその後データ転送を行う．

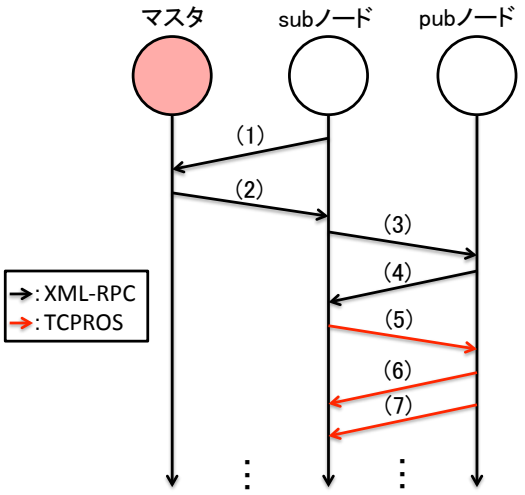

図 2 出版購読通信における通信フロー
Fig. 2 Publish/subscribe communication flow

図 2 にデータを送信するパブリッシャーノードと受信するサブスクライバーノードが出版購読通信が開始されるまでの通信フローを示す．

(1) まず，ROS マスタに対して XML-RPC における register メソッドにより自身のノード名，扱うトピック名および URI の登録を行う．
(2) マスタからは登録されているパブリッシャーノードの URI と XML-RPC の受付ポートが通知される．
(3) マスタから与えられた URI とポート番号に接続し，パブリッシャーノードに対して XML-RPC でトピックのリクエストを行う．
(4) パブリッシャからのレスポンスでは TCPROS を受け付けるポート番号が送られる．
(5) 得られたポート番号に接続し，TCPROS 通信を行う．TCPROS 通信では，接続した直後に TCPROS コネクションヘッダの送信を行う．
(6) パブリッシャーノードが TCPROS コネクションヘッダを送信する．
(7) コネクションヘッダの送受信が完了し接続が確立された後，トピックに対応したデータ転送が開始される．

mROS において，以上の通信フローをサポートすることで ROS ノードを組込みデバイス上で実行可能にする．

2.3 関連研究

[3], [4] によれば，ROS は汎用向け Linux 上でしか動作しないため，リアルタイム性の確保は難しく，分散リアルタイムシステムの構築には向いていないと述べられている．そこで，ROS にリアルタイム性を持たせるための手法が提案されている．

ROS の次世代バージョンとして開発されている ROS2[*1]

[*1] ROS2 に対応させて本章で紹介した現行バージョンの ROS を

© 2017 Information Processing Society of Japan

では，リアルタイム OS を含む様々な環境で実行可能であるように設計されている．ROS2 では master ノードを介さない DDS 通信の採用が想定されている．しかし，ROS2 では master ノードが存在せず，ノード間の通信システムが ROS とは異なるため，既存パッケージ含めて ROS1 との互換性は持たない．

ROS システムにおいて組込みデバイスを利用する方法として，LinuxOS の代わりにリアルタイム OS を使用する手法が提案されている [5], [6]．また，Linux の動作するホストデバイスと組込みデバイス上で動作するプログラムとの通信をサポートする ROS パッケージもある [7], [8]．これらの方法では，組込みデバイス上で実行されるプログラムが動作する環境は広くサポートされておらず，ROS API で記述された ROS ノードをそのまま実行できるものではない．そのため，オープンソースとして公開されている ROS パッケージを組込みデバイス上で活用することが難しい．

3. mROS

本研究では，ROS ノードの軽量な実行環境を提案する．提案する実行環境は mROS と名付けており，組込みマイコンを搭載したデバイス上で ROS ノードが動作できるように設計を進めている．

3.1 開発目標と設計要件

本研究では，ROS および Linux の動作するホストデバイス，および，エッジ端末として ROS ノードの動作する組込みデバイスで構成される分散システムを想定する．ROS のノードを管理するマスタはホストデバイス上で動作していることとする．組込みデバイス上で実行されるプログラムは，ROS ノードとして振る舞えるようにする．すなわち，ROS の通信層を介して ROS システムとの通信をサポートする組込みデバイス向け通信ライブラリを提供する必要がある．

汎用 PC における ROS では，Linux が提供する TCP/IP プロトコルスタックを用いた通信層を介してノード間のデータ通信を行う．Linux が搭載できない組込みマイコンにおいて ROS 通信層と通信する場合には，Linux 無しで動作できる TCP/IP プロトコルスタックを採用する必要がある．加えて，組込みデバイスからのホストデバイスとの通信処理，および，組込みデバイス上で動作するプログラム資源を管理する機能が必要である．

提案する実行環境である mROS を用いて提供される通信ライブラリは ROS と API 互換になるようにする．このことにより，ユーザは ROS ノードを記述するときと同様の API を使用するだけで，mROS アプリケーションを記述することが可能になる．

ROS1 と呼ぶことがある．本稿では ROS1 を対象とする．

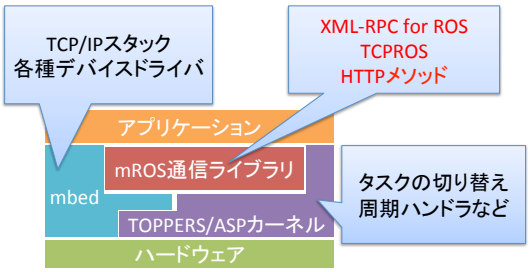

図 3 組込みデバイス上での ROS ノード実行環境のソフトウェア構成

Fig. 3 Running emvironment for ROS components on the embedded device

3.2 構成

mROS では，リアルタイム OS と TCP/IP プロトコルスタックを使用して mROS 通信ライブラリを提供することで ROS ノードの実行環境を実現する．通信処理の管理を担うリアルタイム OS としては，μITRON 仕様のリアルタイム OS である TOPPERS/ASP カーネル [2] を採用する．TOPPERS/ASP カーネルは，タスクと呼ばれるスレッドの生成を静的に行うため，組込みシステムに適しているという特徴がある．TCP/IP プロトコルスタックには，ARM mbed ライブラリに含まれる lwIP を用いる．mbed ライブラリは ARM プロセッサを搭載する多くの組込みデバイスで，周辺ペリフェラルを含む様々なデバイスの統一的なデバイスドライバが提供されている．また，様々なライブラリがオープンソースとして利用できるためプログラム設計が容易になることが期待できる．現在は，両者が移植済みである GR-PEACH [9] を対象として開発を進めている．GR-PEACH は Arduino 互換ボードであり，マイコンとして RZ/A1H が搭載されている．**図 3** に組込みデバイスにおける ROS ノード実行環境のソフトウェア構成を示す．開発中の実行環境では，mbed ライブラリと TOPPERS/ASP カーネルを活用したアプリケーション設計も可能である．なお，実装上は mbed ライブラリのラッパであるが，Arduino に準拠したライブラリ関数を活用することもできる．

3.3 mROS 通信ライブラリ

本節では，ROS システムとの通信を実現する機能を提供する mROS 通信ライブラリの内部実装について述べる．

3.3.1 実現する機能

組込みデバイス上のプログラムに対して ROS ノードとしての振る舞いを可能とするために，mROS では 2.2 で述べた通信プロトコルの動作をサポートする．以降，図 2 の番号を参照して説明を行う．(1) および (3) ではマスタや他のノードに対して手続きを呼び出す XML-RPC 通信のマスタとなる機能が必要となる．呼び出すメソッドの名前や引数の情報をユーザタスクから取得し，XML によるエン

© 2017 Information Processing Society of Japan

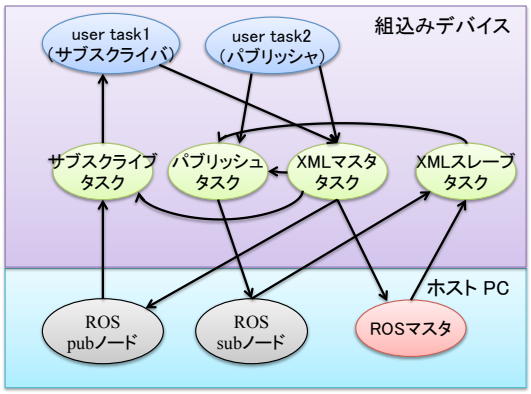

図 4 mROS におけるタスク構成とデータフロー
Fig. 4 Data flow on mROS

コード，およびリクエスト送信を行う．また，XML-RPC 通信のスレーブとなる機能も必要であり，通信相手および通信タイミングが不特定の要求を受け付けてレスポンスの返信を行わなければならない．mROS では周期ハンドラを用いて接続要求を確認することで XML-RPC 通信のスレーブとなる機能を実現している．

(2) および (4) では XML-RPC 通信のレスポンスを受信して処理する機能が必要になる．XML-RPC プロトコルでは手続き呼び出しにおいて必ずレスポンスデータが返信されるため，mROS では XML-RPC マスタとしての送信処理の後には必ず受信処理が行われる．受け取った XML データからデコードを行い，レスポンス結果によって必要な処理が行われる．

(5)，(6) および (7) で扱うデータは，TCPROS データプロトコルに合わせてデータをエンコードおよびデコードする必要がある．mROS では送信するデータに対してバイナリのデータ長を先頭に付与する機能を提供することで，ユーザからはデータ本体を引数として API を呼び出すだけとなっている．

ROS では，データ通信に TCP と UDP のどちらかを使用するが，本稿執筆時点での mROS では TCP のみをサポート対象として実装を行っている．

3.3.2 タスク構成

TOPPERS/ASP カーネルでは，アプリケーションはタスク単位で資源管理が行われる．mROS 通信ライブラリを使用する際には，ユーザが記述するユーザタスクに対して，サブスクライブタスク，パブリッシュタスク，xml マスタタスクおよび xml スレーブタスクの 4 つのタスクが用意され，ROS システムとの通信機能を提供する．**図 4** にタスクの構成とデータフローを示す．これらのタスクは mROS 通信ライブラリを読み込むだけで生成される．タスク間の通信は，μITRON で規定されているデータキューと共有メモリを使用する．データ転送は XML でエンコードされたデータを，共有メモリに格納し，その先頭アドレスとデータ長の情報をデータキューで他タスクへと通知することで

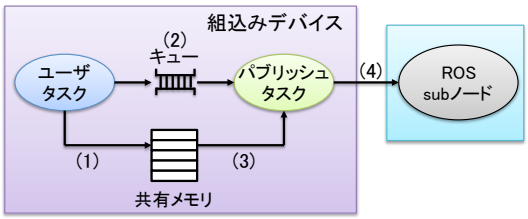

図 5 publish()API 実行フロー
Fig. 5 Task flow of publish

行う．mROS におけるデータキューのデータフォーマットは 32bit で，メッセージ ID と呼んでいる．メッセージ ID は 8bit のノード識別 ID，8bit のデータ格納先の共有メモリベースアドレス，16bit のデータ長で構成されている．データ長が 16bit であるため，現在の mROS で出版購読できるデータサイズの最大長は 64KB になっている．

ユーザタスクでは，ROS の API を使用してノード処理が記述できる．それぞれの ROS API に対応する機能は，4 つのタスクが提供する．

advertise()API はパブリッシャとして ROS マスタに登録を行う．引数に出版するトピック名をとり，xml マスタタスクのデータキューへ初期化を通知する．xml マスタタスクがノード識別 ID を割り当て，ROS マスタへ XML-RPC 通信，およびパブリッシュタスクへの登録を行う．xml スレーブタスクがサブスクラバーノードからの接続要求を受付け，パブリッシュタスクのポートを返送する．パブリッシュタスクでトピックの出版ソケットの接続が完了すると，返り値として Publisher 構造体を返す．

publish()API は advertise() で登録された Publisher 構造体のメンバ関数になっており，出版データを引数に取る．**図 5** に publish()API が呼び出された時の実行フローを示す．ユーザタスクがデータを共有メモリに書き込む (1)．メッセージ ID をパブリッシュタスクのデータキューに送る (2)．パブリッシュタスクが共有メモリからデータを取得し，TCPROS データプロトコルでエンコードを行う (3)．ノード識別 ID に対応するソケットからデータを出版する (4)．

subscribe()API はサブスクライバとして ROS マスタに登録を行う．**図 6** に subscribe()API が呼び出され，データ購読が行われるフローを示す．引数に購読するトピック名，およびトピックを購読した時に実行するコールバック関数をとる．xml マスタタスクのデータキューへと初期化通知を行う．xml マスタタスクがノード識別 ID が割り当てと ROS マスタへの登録 XML-RPC 通信を行い，サブスクライブタスクのデータキューへ登録通知を行う (1)．登録が行われると，サブスクライブタスクはトピックの購読リクエスト要求を xml マスタタスクに送る．xml マスタタスクはパブリッシャノードに対して XML-RPC 通信で購読要求を行い，ポート番号を取得し，サブスクライバタ

© 2017 Information Processing Society of Japan

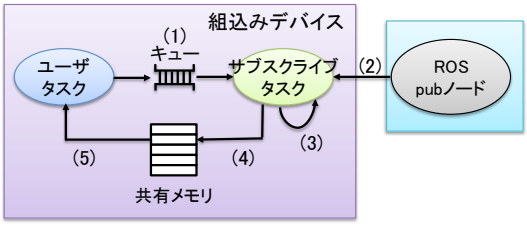

図 6 サブスクライブタスクフロー
Fig. 6 Task flow of subscribe

スクでは TCPROS プロトコルで接続を行う．サブスクライブタスクはユーザタスクとは独立して，周期ハンドラで起動されデータの購読を行う (2)．サブスクラバタスクのコンテキストでコールバック関数を実行する (3)．コールバック関数の結果をユーザタスクで使用する場合はサブスクライブタスクが共有メモリに書き込む (4)．ユーザタスクが共有メモリからコールバックの結果を取得する (5)．

これらは ros::NodeHandle クラスのメンバとなっており，ros::spin() や ros::Rate() もサポートしている．

3.4 プログラミングモデル

mROS では，前述の mROS 通信ライブラリが提供する API を利用することで ROS システムと通信を行うアプリケーションが設計できる．mROS 通信ライブラリが提供する機能は，すべて ROS の API と同一の名前で定義している．そのため，ユーザは ROS のプログラミングに必要な知識だけで mROS のアプリケーション記述し，実行することができる．オープンソースとして公開されている ROS パッケージの利用することも可能であると考えられる．

一方，アプリケーションのマルチタスク実現，および TOPPERS/ASP カーネルの API の利用やコンフィギュレーションの設定を行うことも可能になっている．このことにより，従来の組込みシステムの特性を利用できるため，リアルタイム性の保証が容易になる可能性がある．また，mbed のライブラリを活用することで様々なデバイスのアプリケーションが記述できる．

4. 評価

4.1 評価環境

ホストデバイスとして，Intel Core i7 2.4GHz, メモリ 16GB を搭載した NEC の LAVIE Hybrid ZERO を使用する．OS は Ubuntu 14.04LTS を搭載し，ROS indigo ディストリビューションを使用する．組込みデバイスは ARM Cortex-A RZ/A1H 400MHz を搭載した GR-PEACH を使用する．リアルタイム OS として TOPPERS/ASP カーネル，TCP/IP スタックとして mbed ライブラリを使用する．ホスト PC と GR-PEACH は同一イントラネットに接続し，IP アドレスは DHCP で取得する設定とした．

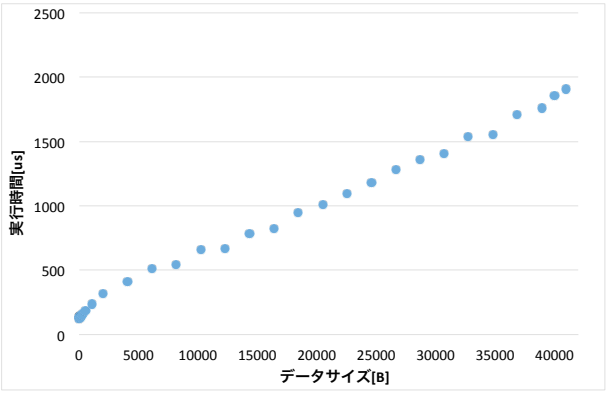

図 7 出版処理にかかる実行時間の平均値
Fig. 7 Average time of publishing data

4.2 実行時間

データ出版の実行時間は，ユーザタスク内からデータを出版する API を呼び出す直前から，データ出版が完了しユーザタスクが次の処理を行えるまでの時間を計測した．TOPPERS/ASP カーネルで用意されている get_utm() サービスコールを用いて測定区間の前後でマイクロ秒単位の時刻を取得し，その差を取ることで実行時間とした．

図 7 に 4B から 40960B までデータサイズを変えてそれぞれ 50 回出版したときの平均実行時間を示す．4B から 2048B までは，2 の冪乗ずつデータサイズを変え，2048B から 40960B までは 2048B ずつデータサイズを増やして実行時間の評価を行った．表 1 に 40KB のデータを 200 回出版したときの実行時間の分布を示す．

図 7 から，データサイズと実行時間には線形の相関があることがわかる．また，40KB 程度のデータであれば 2ms 以下で高速に出版を行うことができる．しかし，表 1 では同じデータサイズを出版するときの実行時間にばらつきがあることがわかる．極端に大きさが異なるデータを除いても，ばらつきが 2ms の間にあり，リアルタイム性を確保することが難しいと考えられる．この原因は究明中であり，リアルタイムシステムを構築することを容易にするために改善する必要がある．

データ購読の実行時間は，TCP ソケットからデータを取得した直後から，コールバック関数が呼び出された直後までを実行時間とし，計測を行った．**図 8** に 2000B から 52000B までを 2000B ずつデータサイズを変えてそれぞれ 5 回購読したときの平均実行時間を計測した．このときサ

表 1 40KB のデータ出版実行時間分布
Table 1 Distribution of execution of publishing 40 KB data

実行時間 [us]	サンプル数 [個]
900 以下	3
901 – 1500	29
1501 – 2500	161
2501 – 3100	2
10000 以上	5

© 2017 Information Processing Society of Japan

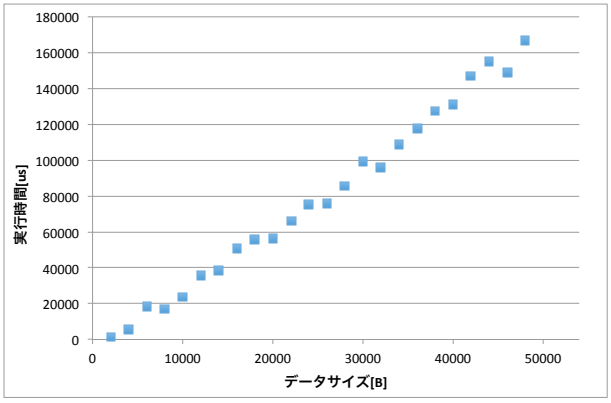

図 8 購読処理のかかる実行時間の平均値
Fig. 8 Average time of subscribing data

ブスクライブタスクは 100ms の周期で起動した．データ購読の場合にも，データサイズと実行時間に線形の相関があることがわかる．データサイズが大きくなると分割してデータを購読する現象が生じ，受信周期を短くするとデータが取れない場合があった．mROS を使用して実現するアプリケーションはエッジ端末として，カメラなどのセンサデータを処理して出版し，ホストデバイスからは制御用の命令データを購読する状況が想定される．この場合，出版データは大きなデータサイズを扱う必要があるが，購読するデータは小規模のものを扱えられれば十分であると考えられる．

今回は，mROS 単体の評価を行ったため，ネットワークの通信時間は含めていない．ロボットシステム全体の評価を行うためには，ネットワーク構成を含めて議論する必要がある．

4.3 プログラムのサイズ

mROS で必要となるオブジェクトファイルのデータサイズを size コマンドにより取得した．40KB のデータを出版する実行時間の評価の際に使用したプログラムのオブジェクトファイルのサイズを評価した結果を表 2 に示す．合計で 500KB 程度の大きさであり，我々が想定する組込みデバイスに搭載できる軽量なものであるといえる．なお，全体の 68%を占める mbed ライブラリは使用しないデバイスドライバもリンクされるため，使用しないファイルを除けばさらに軽量化が可能である．

表 2 mROS 環境のオブジェクトファイルサイズ
Table 2 Size of object files of mROS environment

name	text	data	bss	dec
カーネルライブラリ	82893	0	16396	99289
mbed ライブラリ	267038	52949	45719	365706
mROS ライブラリ	66271	28	166	66465
合計	416202	52977	62281	531460

5. まとめ

本研究では組込みデバイス向け軽量実行環境である mROS の提案した．TOPPERS/ASP カーネルと ARM mbed ライブラリを採用して mROS の開発を進めている．mROS 通信ライブラリで ROS の通信プロトコルをサポートすることで Linux が搭載できない組込みデバイス上で ROS ノードの実行が可能になる．実行時間と mROS 全体のサイズに関する評価を行ったところ，実行時間とデータサイズの大きさに線形関係があることが確認できた．また，mROS は実行環境全体として軽量なプログラムサイズであることが示された．

今後の課題として，ROS の通信方式の一つのサーバクライアント通信のサポートおよびすべての ROS API への対応がある．また，実行環境として TOPEERS/ASP カーネルと ARM mbed ライブラリを使用しているが，対応デバイスの拡張や実現環境の拡張を検討する必要がある．さらに，適用事例として，画像データの取得および変換処理を mROS の動作するエッジ端末で実行し，その結果をホストデバイスに出版処理するデモシステムの開発に取り組んでいる．

謝辞 本研究の一部は JSPS 科研費 16H02795 の助成による．

参考文献

[1] Quigley, M., et al.: ROS: an open-source Robot Operating System, *ICRA workshop on open source software*, No. 3.2, p. 5 (2009).
[2] TOPPERS プロジェクト：TOPPERS/ASP kernel, `https://www.toppers.jp/asp-kernel.html`.
[3] Maruyama, Y., et al.: Exploring the Performance of ROS2, *Proc. of EMSOFT*, pp. 1-10 (2016).
[4] Bezemer, M., et al.: Connecting ROS to a real-time control framework for embedded computing, *Proc. of ETFA*, pp. 1–6 (2015).
[5] Migliavacca, M., et al.: μROSnode: running ROS on microcontrollers, *ROS Developers Conference* (2013).
[6] Migliavacca, M.: The R2P framework for robot prototyping: methodological approach, hardware modules, and software components, PhD Thesis, Italy (2014).
[7] Bouchier, P.: Embedded ros [ros topics], *IEEE Robotics & Automation Magazine*, Vol. 20, No. 2, pp. 17–19 (2013).
[8] Crick, C., Jay, G., Osentoski, S., Pitzer, B. and Jenkins, O. C.: Rosbridge: Ros for non-ros users, *Robotics Research*, Springer, pp. 493–504 (2017).
[9] Open-source Software Platform Based on TOPPERS/ASP Kernel, mbed and Arduino Library for Renesas GR-PEACH.: `https://github.com/ncesnagoya/asp-gr_peach_gcc-mbed`
[10] Merrick, P., Allen, S. and Lapp, J.: XML remote procedure call (XML-RPC) (2006). US Patent 7,028,312.

© 2017 Information Processing Society of Japan