Vol.2016-ARC-219 No.5
Vol.2016-SLDM-175 No.5
Vol.2016-EMB-40 No.5
2016/3/24

IXM:ロボット制御ソフトウエア向けプロセス間通信ミドルウエア

住谷拓馬[†1]　松原豊[†2]　菅谷みどり[†1]

概要：近年，ロボットの利用において，複数の機能を状況に応じて切り替えられ，より複雑な自律行動が可能なことが期待されている．このようなロボットを制御するにはセンサやアクチュエータなどを制御する複数の機能プログラム間でデータ共有のための通信を行うことが必要となる．そこで，ロボットの制御ソフトウエア開発において，複数プログラム間の通信を提供するミドルウエアは不可欠となっている．本稿では，提案する通信ミドルウエア IXM (Information eXchange Middleware) の設計，既存のミドルウエア ROS(Robot Operating System)との機能比較，データ共有時間の比較結果について述べる．

キーワード：ロボット，ミドルウエア，ROS，プロセス間通信，データ共有

IXM:Interprocess Communication Middleware for Robot Software

TAKUMA SUMIYA[†1]　YUTAKA MATSUBARA[†2]
MIDORI SUGAYA[†1]

Abstract: In recent years, switching a plurality of functions in accordance with the situation, the robot capable of complex autonomous behavior is expected. Such control the robot, it is necessary to perform the communication for data sharing between a plurality of functional program controlling the sensors and actuators. Therefore, in the control software development of the robot, middleware has become essential to communication between multiple programs. In this paper, the design of the proposed communication middleware IXM (Information eXchange Middleware), feature comparison with the existing middleware ROS (Robot Operating System), and the comparison result of data sharing time are described .

Keywords: Robot, Middleware, ROS, Interprocess Communication, Data sharing

1. はじめに

近年，ロボットに関する研究が盛んになっている．特にロボットの利用において，複数の機能を状況に応じて切り替えられ，より複雑な自律行動が可能なことが期待されている．このようなロボットを制御するにはセンサやアクチュエータなどを制御する様々な機能プログラムが必要である．またそのソフトウエア開発においては，複数の機能プログラム間でデータ共有のための通信を行うことが必要となり，容易に実現可能な開発環境が必要となる．こうしたロボットの制御ソフトウエア開発において，複数プログラム間のデータ共有のための通信を提供するミドルウエアは不可欠である．

ROS(Robot Operating System)[1]では，通信はソケット通信で，データ共有は publish/subscribe モデル[2]により実現している．しかしロボット内の単一コンピュータで，より応答性能を求める場合，ROS が採用するソケット通信はデータ共有時の通信のオーバーヘッドがボトルネックとなる．それに対し，ROS はローカルメモリを用いた nodelet[3]という仕組みを提供している．

しかし，ROS の nodelet には通信ミドルウエアの性能と安全性について課題がある．具体的には，性能については，通信のためのキューを複数用意し，マルチスレッドで実装している．そのため，キューの操作とスレッドの管理により処理数が多く，データ共有時に処理のオーバーヘッドがかかる．ROS でもこれら問題は認識されており，DDS(Data Distribution Service)[4]を ROS の通信機構として実装することが考えられている．安全性についてはスレッド実装であることから，スレッド間でメモリ領域が予期せずアクセスされデータが書き換えられる可能性がある．

今後，様々な機能実現が求められるロボット制御ソフトウエアでは，ロボットの利用特性に合わせて複数のプログラムを連携させる必要がある．また，それが安全に動作しなければならない．そこで本研究では，複数プログラム間でのデータ共有の高速化を図るとともに，安全性を考慮することを目的とする．そのために，ロボット制御ソフトウエア向けミドルウエア IXM(Information eXchange Middleware)を提案する．

具体的には，(1)複数プログラム間のデータ共有の高速化のために，データ共有時の内部処理数をできるだけ単純なものとすることで，処理オーバーヘッドの少ないデータ共有を目指す．(2)安全性のためには，制御ソフトウエアの予期しない誤動作を防ぎ，各機能を正しく動作させるためのデータ保護機能の実現を検討する．

†1 芝浦工業大学
SHIBAURA INSTITUTE OF TECHNOLOGY
†2 名古屋大学
Nagoya University

Vol.2016-ARC-219 No.5
Vol.2016-SLDM-175 No.5
Vol.2016-EMB-40 No.5
2016/3/24

実装方法は，データ共有の高速化については近年のロボットが持つメモリ容量が増大していること，スレッド間においても通信のオーバーヘッドが高いことに着目し，共有メモリを用いてデータ共有を行うものとした．安全性については，共有メモリによるデータ共有により通信コストの削減が大中に見込まれることから，プログラムの実行単位をプロセスとすることでメモリ領域を分け，またプロセスにデータへのアクセス権を持たせるものとした．

評価については IXM と ROS の nodelet とで機能比較とデータ共有時間の比較を行った．

2. 既存のロボットミドルウエア

2.1 RT ミドルウエア

RT ミドルウエア[5]とは，カメラやセンサ，モータなどを制御するソフトウエアを RT コンポーネントと呼び，それらをつなぐ通信規格である．そのため，この規格のもとに開発されたものが実際にはミドルウエアとして利用される．例えば，OpenRTM-aist が挙げられる．RT ミドルウエアは規格に準拠して開発されたものの総称である. RT ミドルウエアの規格，OpenRTM-aist ともに産業技術総合研究所が主体となって開発している．

RT ミドルウエアの通信モデルを図 1 に示す．

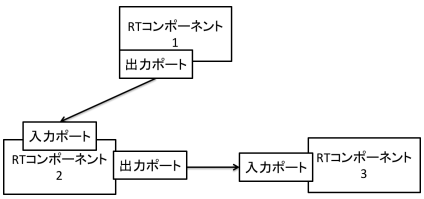

図 1　RT ミドルウエアの通信モデル

Figure1　Communication model of RT-Middleware.

RT コンポーネントは入出力ポートを持ち, そのポートを介してコンポーネント間で通信を行う．ポートには，データポートとサービスポートがあり，データポートはデータの送受信を行い，サービスポートは関数のリモート呼び出しを行う．

RT コンポーネントは入力に対するコールバック関数によって実装されており，入力があるとそれに対応した処理と出力を行うものとなっている．

2.2 ROS

ROS は，ロボット制御ソフトウエア開発のためのシミュレーションツールや，複数プログラム間で通信を行うためのライブラリを提供しているミドルウエアである．Kobuki など移動ロボットを制御するためのライブラリも充実している．WillowGarage により開発されている．本研究では，ROS の代表的な機能である ROS 通信ライブラリに着目し，提案する IXM との比較，分析を行う．

ROS はプログラム間の通信を Publish/Subscribe モデルにより実現している．通信モデルを図 2 に示す．

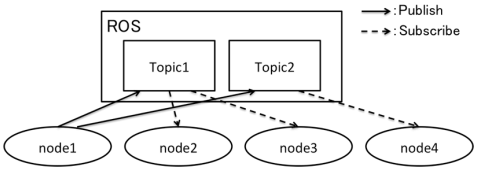

図 2　Publish/Subscribe モデル

Figure2　Publish/Subscribe model.

ROS ではプログラムはノード(Node)として扱われ，ノードは通信の名前を指し示すトピック(Topic)を介して通信を行う．データを送信するノードはトピックに対しデータの出版(Publish)を行い，受信するノードはトピックから購読(Subscribe)を行うことで通信が成立する．

図 1 では, Node1 は Topic1 と Topic2 に対しデータを出版し，Node2，Node3 は Topic1 から購読している．Node3，Node4 は Topic2 から購読している．

このように，トピックを介することで複数のノード間で通信を行うことができる．また，ノードはどのトピックに対して操作を行うか把握すればよく，通信相手のノードを意識せず通信することができる．

2.2.1 nodelet

ロボット内の単一コンピュータでソフトウエアの応答性能を求める場合，ROS の提供するソケット通信はデータ共有時の通信のオーバーヘッドがボトルネックとなる．

そこで ROS は nodelet という機能を提供している．これは，ノードをスレッド単位で実行し，ノード間でローカルメモリを共有してデータ共有を行う仕組みである．メモリ上のメッセージ用キューに更新があると直ちに関係するコールバック関数が呼び出される．ソケットを用いたデータ共有より，データを送信形式に整える処理であるシリアライズ，受信したデータを復元するための処理であるデシリアライズの処理コストが軽くなり，ソケット通信も行わないためデータ共有速度が速い．

2.2.2 nodelet における課題

ROS の nodelet には通信ミドルウエアのデータ共有時間と安全性について課題がある．具体的には，データ共有時間については汎用性を考慮し内包する機能が多いため，キューの操作やコールバック関数呼び出しなどの処理数が多く，データ共有時に処理のオーバーヘッドがかかる．安全性についてはスレッド実装であることから，スレッド間でメモリ領域が予期せずアクセスされデータが書き換えられる可能性がある．

3. 予備実験

3.1 目的と方法

共有メモリを用いることでデータ共有の高速化がどれだけ見込めるかを調査するために，ソケット通信と共有メモリを用いた場合のデータ共有時間の比較実験を行った．

実験方法として，データ送信プログラムとデータ受信プ

Vol.2016-ARC-219 No.5
Vol.2016-SLDM-175 No.5
Vol.2016-EMB-40 No.5
2016/3/24

ログラムの二つを作成し，そのプログラム間でのデータ共有時間の計測を行った．計測区間は，データ送信 API を呼ぶ直前からデータ受信 API を呼んだ直後までである．送信側でデータ送信 API を呼ぶ直前に時間計測 1 を行い，その値を送信する．値のデータサイズはソケット通信が 24byte，共有メモリが 8byte である．受信側は，データ受信 API を呼び出した直後に時間計測 2 を行い，時間計測 1 と 2 の差を算出する．この差をデータ共有時間とした．この計測を 1 回とし合計 1000 回計測した．計測区間の時間を測るためには送受信のタイミングを合わせる必要があるため，共有メモリでのデータ共有では，共有メモリアクセス権を記述したファイルを用意した．各プログラムはそのファイルを使ってポーリングするものとした．ポーリング周期は特に設定せずに行った．ソケット通信でのデータ共有では，データの送受信に send システムコールと recv システムコールを用いた．recv システムコールを呼び出すとデータが送信されるまで待機状態となるので，共有メモリで行ったようなポーリング処理は行わず recv を呼んだ後に send を呼ぶものとした．環境は Intel Core i5， 2.6GHz の 4 コア，メモリ 4GB の PC で，OS は ubuntu12.04LTS(32bit)を用いた．また CPU のパフォーマンス設定を行うことできる cpufreq をインストールし，負荷に関わらず最高クロック周波数で動作するよう設定した．

3.2 結果

ソケットと共有メモリのデータ共有時間の比較を表 1 に示す．データ共有時間の単位はマイクロ秒である．

表 1　ソケットと共有メモリのデータ共有時間比較
(単位:μs)

Table1　Data sharing time comparison of socket and shared memory. (Unit: μs)

方法	平均値	中央値	標準偏差値	最悪値
ソケット	74	37	69	241
共有メモリ	14	14	2	51

表 1 のように，すべての値において共有メモリの方がデータ共有時間が短い結果となった．ソケットについては，平均値と中央値に差があり，標準偏差値も共有メモリに対して大きい値となった．また最悪値も遅い値となったことから，データ共有時間の安定性と最悪時間において共有メモリの方が優れている．このことから，IXM の設計として考慮しなければならない，動作の安定性と最悪応答時間の向上を期待できると考えた．

また，本実験においてマイクロ秒単位で差が見られたため，以降の実験でも整数値に丸めるものとした．

4. IXM

データ共有時の処理の高速化安全性の考慮を目的とし，共有メモリを用いたロボット制御ソフトウエア向けプロセス間通信ミドルウエア IXM(Information eXchange Middleware)を提案する．以下にソフトウエア構成，通信モデル，ソフトウエア構成，プロセス関係，データ共有時のシーケンスについて述べる．

4.1 ソフトウエア構成

IXM を使用する際のソフトウエア構成を図 3 に示す．

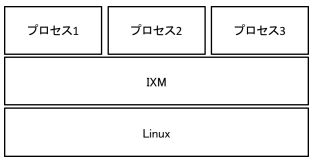

図 3　IXM のソフトウエア構成

Figure3　Software configuration of IXM.

IXM は ubuntu など Linux ベースの OS 上で動作する．各プロセスは IXM 上で動作し，IXM が提供する API を利用して通信を行う．

4.2 通信モデル

IXM の通信モデルを図 4 に示す．

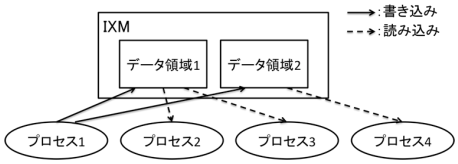

図 4　IXM の通信モデル

Figure4　Communication model of IXM.

IXM はデータをやり取りするためのデータ領域を持つ．データ領域は共有メモリで構成され任意に増減できる．データ領域には識別番号があり，それを IXM キーと呼ぶ．プロセスはその領域に対してデータを書き込む，読み込むという 2 つの操作を行う．1 つのデータ領域に対し複数のプロセスがそれぞれのタイミングで操作を行うことができる．プロセスは自身がどのデータ領域を操作できるかのみを把握しており，共有メモリとデータをやり取りする相手のプロセスは把握していない．

4.3 プロセス関係

プロセスの関係を図 5 に示す．

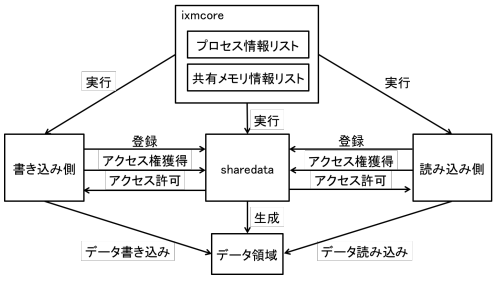

図 5　IXM のプロセス関係

Figure5　Process relationship of IXM.

IXM には核となる ixmcore というプロセスが存在し，このプロセスが共有メモリを管理するプロセスの実行や，ロボットを動作させるプロセスの実行を行う．ixmcore はプロセス情報リストと共有メモリ情報リスト構造体を生成する．プロセス情報リストは，実行中プロセスの名前，プロ

セス ID, IXM キーがリストされている. 共有メモリ情報リストには，IXM キーとそれに対応する共有メモリアドレス，データ型，(要素数)サイズがリストされている．sharedata，データの書き込み側，読み込み側プロセスは ixmcore の子プロセスとして実行される．この時にそれぞれのプロセス ID をプロセス情報リストへ保存する. sharedata は実行されると各リストを共有し，プロセスからのアクセス待ち状態となる．データ書き込み側，データ読み込み側プロセスはまず自身が使う IXM キーを ixmcore へ登録する．登録されると sharedata はデータ領域を生成する．そして，その IXM キーを使ってデータ領域へのアクセスを行う．アクセスには IXM から許可をもらう必要があり，その際に IXM は各リストから許可の判断をする．

ここでは，説明のためプロセスを書き込み側と読み込み側に分けているが，実際は分ける必要はなくプロセスは登録さえすれば読み書きを行うことができる．

4.4 データ共有時のシーケンス

IXM が提供する API を用いて，基本的な一対一通信を行う場合のシーケンスを図 6 に示す．多対多通信の場合も手続きは同様である．

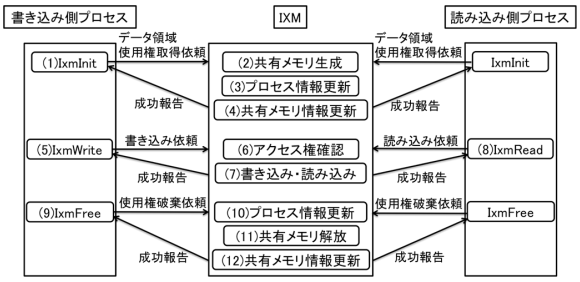

図 6　IXM の一対一通信時のシーケンス図

Figure6　Sequence diagram in case of one-to-one communication in IXM.

図 6 に示した番号順に通信の流れと，その時使用する API を以下に示す．読み込み側プロセスの記述のない箇所は書き込み側プロセスと同様である．

(1)書き込み側プロセスは IxmInit によりデータ領域使用権取得を IXM に依頼する．(2)IXM はプロセスに指定された型と(要素数)サイズで共有メモリを生成する．すでに生成されている場合は何もしない. (3)IXM はプロセスとデータ領域の対応関係を示したプロセス情報リストを更新する．(4)IXM はデータ領域と共有メモリアドレスの対応関係を示した共有メモリ情報リストを更新する．(5)書き込み側プロセスは IxmWrite によりデータ領域への書き込みを依頼する. (6)IXM はプロセス情報リストを参照しアクセス権を確認する．確認できた場合，IXM は共有メモリ情報リストを参照し共有メモリアドレスを取得する. (7)IXM は取得したアドレスの共有メモリにデータを書き込む．(8)読み込み側プロセスは，IxmRead によりデータ領域の読み込みを依頼する．処理は書き込みの場合と同様で成功するとデータ領域内のデータを取得する．(9)書き込み側プロセスは IxmFree によりデータ領域使用権破棄を IXM に依頼する．(10)IXM はプロセス情報リストを更新する. (11)データ領域がすべてのプロセスから使用権を破棄された場合，IXM はそのデータ領域に対応する共有メモリを解放する．(12)IXM は共有メモリ情報リストを更新する．

5. 評価

IXM と ROS の nodelet において，機能比較とデータ共有時間の比較を行った．

5.1 IXM と ROS の機能比較

機能比較の結果を表 2 に示す．

表 2　IXM と ROS の機能比較

Table2　Feature comparison table of IXM and ROS.

	IXM	ROS : socket	ROS : nodelet
(1)通信方式	非同期	非同期	非同期
(2)実現方法	共有メモリ (プロセス間)	ソケット (プロセス間)	ポインタ渡し (スレッド間)
(3)ネットワーク通信	×	○	×
(4)実装言語	C	C++, Python	C++, Python
(5)対応データ型	int (32bit) float (64bit)	int (8bit – 64bit) float (32, 64bit)	int (8 – 64bit) float (32, 64bit)

各項目について比較を行う．

(1)IXM はデータ領域を介してプロセス間で非同期にデータの書き込みと読み込みを行う．ROS はトピックを介してノード間で非同期にデータの出版と購読を行う．送受信のタイミングを合わせる場合，IXM はポーリングなどの処理を API 呼び出しの前後で記述する必要がある．ROS:socket では，トピックが更新されるまで待機し，更新されるとだたちにコールバック関数を呼び出す方法と，購読側プロセスのタイミングでトピックにアクセスし，コールバック関数を呼び出す方法の二つが提供されている．ROS:nodelet では，ROS:socket の前者の方法のみである．

(2)ROS:socket はプロセス間で TCP/IP によるソケット通信を行うため，信頼性のある通信を行うことができるが，通信速度に問題がある. ROS:nodelet はスレッド単位でありスレッド間でのポインタ渡しによってデータを共有しているため通信速度が速い．しかし，スレッド間でメモリ領域が予期せずアクセスされる可能性がある．IXM はプロセス間通信を共有メモリにより行うことでプロセスから直接メモリにアクセスされることを防ぎつつ，データ共有を高速に行うことができる．

(3)ROS:socket は, TCP/IP によるソケット通信であるため，ネットワークを介し他のコンピュータと通信ができる．IXM，nodelet はともに単一のコンピュータ内での高速なデータ共有を目指しているため対応していない．

(4)IXM が C，ROS が C++，Python となっている．

(5)IXM の方が少ないが今後増やすことは容易である．

Vol.2016-ARC-219 No.5
Vol.2016-SLDM-175 No.5
Vol.2016-EMB-40 No.5
2016/3/24

5.2 データ共有時間の比較実験

基本的な性能比較を行うために，IXM，ROS:nodelet において一対一のプログラム間のデータ共有時間を比較した．

5.2.1 実験方法

実験のために，データを書き込む(出版)プログラムと読み込む(購読)プログラムの二つを IXM と ROS:nodelet それぞれで作成し，そのプログラム間でのデータ共有時間の計測を行った．計測区間と環境と CPU のパフォーマンス設定は予備実験と同様である．データは float(64bit)型の 8byte である．また，本実験ではシングルコア環境も考慮し，プロセスを一つの CPU に固定した条件での計測と，プロセスの実行優先度による動作の影響を調査するために，プロセスの実行優先度を最高にした条件での計測を行った．プロセスの CPU への固定は taskset コマンドを用い，プロセスの実行優先度変更は nice 及び renice コマンドを用いた．それぞれのデータ共有方法について述べる．

非同期通信である IXM は計測区間の時間を測るためには書き込みと読み込みのタイミングを合わせる必要がある．そこで，データの書き込み側プロセスと読み込み側プロセスの間で，書き込みと読み込みを許可するための書き込み/読み込み許可用データ領域を生成し，ポーリングを 10 マイクロ秒周期で行った．そして，お互い許可が出たタイミングでデータ共有用データ領域にそれぞれ書き込みと読み込みを行うものとした．計測した時間はファイルに書き込むものとした．実験時における IXM のデータ共有方法を図 7 に示す．

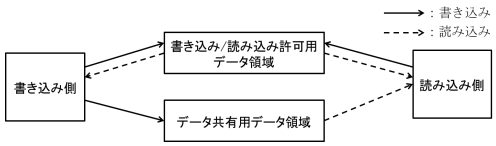

図 7　IXM のデータ共有方法

Figure7　Data sharing method of IXM.

nodelet のプログラムは基本的に Subscriber として実装するようになっており，main 関数は存在せずコールバック関数のみで構成される．そこで，本実験ではデータ共有開始用 Topic を用意し，そのトピックをきっかけとして動作するコールバック関数を出版側プログラム内に作成した．そして，そのコールバック関数内の処理でデータ共有用 Topic に出版をさせるようにし，そのトピックをデータ読み込み側プログラム内に作成したコールバック関数が購読するものとした．データ共有開始用 Topic への出版は，ROS から提供されているコマンドからトピックに対し操作が行える rostopic を用いた．計測時間の保存方法は，nodelet の場合プログラム内でファイルを開くとプログラムが終了されてしまうため，標準出力するものとした．図 8 に ROS:nodelet でのデータ共有方法を示す．

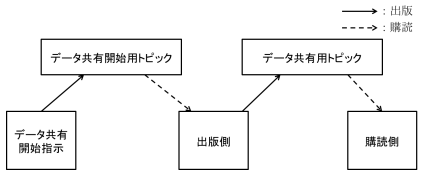

図 8　ROS:nodelet のデータ共有方法

Figure8　Data sharing method of ROS:nodelet.

5.2.2 実験結果と考察

IXM と ROS:nodelet について，1000 回の計測時間の平均値，中央値，標準偏差値，最速値，最悪値を表 3 に示す．

表 3　IXM と ROS のデータ共有時間比較(単位:μs)

Table3　Data sharing time comparison of IXM and ROS. (unit: μs)

ミドルウエア	条件	平均	標準偏差	最悪
IXM	(1)なし	164	43	236
	(2)CPU を一つに固定	93	16	181
	(3)プロセス実行優先度最高	163	43	243
ROS:nodelet	(4)なし	210	23	284
	(5)CPU を一つに固定	155	14	357
	(6)プロセス実行優先度最高	351	54	620

(1)(2)(3)を比較すると，すべての値において(2)が優れた結果となった．IXM はポーリングを行っているため，プロセスの CPU 使用率が高い．(1)の場合はコアのマイグレーションが発生してしまい，そのオーバーヘッドで平均値，標準偏差値，最悪値が大きくなった．また，(1)(3)の結果が変わらないことから，プロセスの優先度はデータ共有時間に関係していない．

(4)(5)(6)を比較すると，平均値，標準偏差値においては(2)が優れており，最悪値においては(1)が優れた結果となった．

IXM と ROS:nodelet について，(2)(4)で比較をすると，平均値は 56%，最悪値は 36%の向上を実現した．また，標準偏差値は変わらない値となっていることから，安定性を保ちつつ速度向上させることに成功した．

データ共有時間のヒストグラムを図 9 に示す．

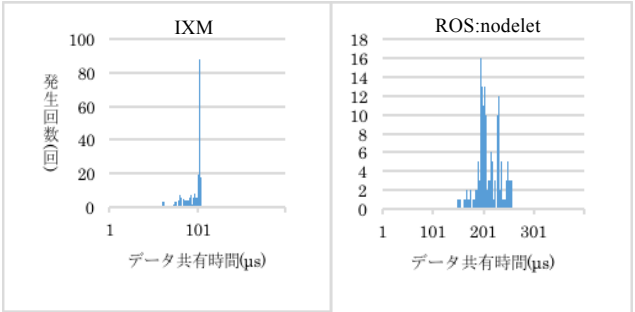

図 9　IXM と ROS:nodelet のデータ共有時間の発生回数

Figure9　Number of occurrences of data sharing time of IXM and ROS:nodelet.

グラフより最大発生回数については，IXM は約 100μs が約 90 回発生している．ROS は約 200μs が約 16 回発生して

Vol.2016-ARC-219 No.5
Vol.2016-SLDM-175 No.5
Vol.2016-EMB-40 No.5
2016/3/24

いる．IXM の方が速い速度で，データ共有時間のばらつきが少ない．

5.3 データ共有時における内部処理時間の比較実験

IXM と ROS:nodelet においてデータ共有時の内部処理の差異と，その処理時間を調査するために実験を行った．

5.3.1 実験方法

ROS のデータ共有時の内部処理を把握し，処理の流れとソースコードを特定した後，各処理の前後に時間計測点を組み込むことで処理時間調査を行った．IXM にも同様に通信 API 内の各処理の前後に時間計測点を組み込んだ．実験環境は 5.2 節の実験と同様であり，プログラムも 5.2 節のものをベースに書き換えたものを使用した．CPU は一つに固定した状態で行った．

また，計測にあたって変数への代入やファイル出力などでかかったオーバーヘッドを算出し，計測結果から引くものとした．

5.3.2 実験結果と考察

図 10, 11 に IXM と ROS のそれぞれの内部処理とその処理時間の中央値(単位:μs)を括弧内に示す．

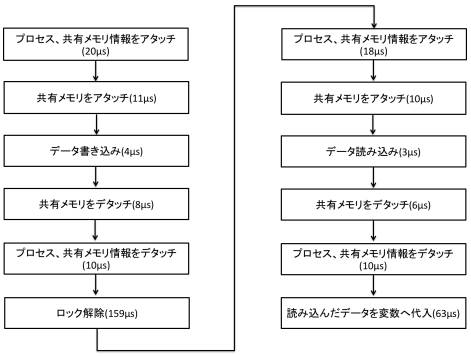

図 10　IXM の内部処理と処理時間

Figure10　Internal processing and the processing time of IXM.

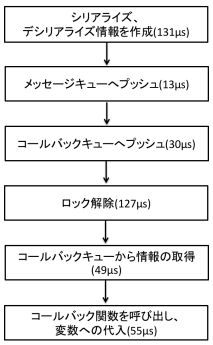

図 11　ROS:nodelet の内部処理と処理時間

Figure11　Internal processing and the processing time of ROS:nodelet.

まず IXM について, ロック解除処理が最も遅い結果となり，159μs かかった．ロックが解除されたかの判断をポーリングにより行っているため，遅延の原因になっていると考えられる．次に，読み込んだデータを変数へ代入する処理は 63μs かかった．また，共有メモリのデタッチよりアタッチに時間がかかっていることがわかった．

ROS:nodelet については, シリアライズ処理とデシリアライズ情報作成処理，ロック解除処理が遅い結果となった．シリアライズ処理とデシリアライズ情報作成処理については，オブジェクトの生成と，生成したオブジェクトの操作を行っているため，それらの処理に時間がかかっていると考えられる．ロック解除処理については，ROS は各スレッドの処理をミューテックスにより排他制御している．そのため，ミューテックスオブジェクトの獲得に時間がかかっていると考えられる．

IXM と ROS:nodelet の比較をすると, IXM は ROS:nodelet に対して速い傾向にあるが極端に遅い処理が存在した．これについては，ポーリング処理を行っていることと，処理内に計測結果を出力する処理が含まれていることが考えられる．

計測した時間の出力については，IXM はロック解除処理内と，読み込んだデータを変数へ代入する処理内でファイル出力している．ROS は各処理内で標準出力している．オーバーヘッドを考慮してもそれ以上の遅延が生まれてしまったと考えられ，IXM においては，ファイルのオープン，書き込み，クローズ処理を行っているため，顕著に現れてしまっていると考えられる．

6. まとめ

本研究では，複数プログラム間でのデータ共有の高速化を図るとともに，安全性を考慮することを目的とし，ロボット制御ソフトウエア向けミドルウエア IXM(Information eXchange Middleware)を提案した．評価としては， IXM と ROS とで機能比較を行い，IXM は ROS:nodelet に対して機能が大きく劣っていないことを示した．また，データ共有時間の比較を行い，IXM が高い性能を安定的に提供できることを確認し IXM の有効性を示した．

7. 今後の課題

まず実際にロボット制御ソフトウエアに適用した場合の全体の応答時間の計測，要件の再検討を行う必要がある．また，対応するデータ型の拡張や，ポーリング以外でのロック処理機能の提供の検討が必要である．

参考文献

1) Morgan Quigley, et al. “ROS:an open-source Robot Operating System”. ICRA Workshop on Open Source Software. 2009.
2) Morgan Quigley, et al. Programming Robots with ROS. 1st ed., O’Reilly, 2015, p31-49.
3) Enrique Fernandez, et al. Learning ROS for Robotics Programming.2nd ed., Packt Publishing, 2015, p35.
4) “ROS on DDS” . ROS 2.0 Design. http://design.ros2.org/, (参照 2015-2-22).
5) 安藤慶昭. ”ロボットミドルウエア標準「RT ミドルウエア」-RT コンポーネントフレームワークと OMG における標準化”. 電子情報通信学会技術研究報告 CNR クラウドネットワークロボット. 2013, 113(248), p15-20.