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の設計と実装

住谷 拓馬[1]　松原 豊[2]　中野 美由紀[3]　菅谷 みどり[4]

概要：近年，接客ロボット，救助支援ロボットやドローンなど，自律的に行動する自律型ロボットの高機能化，複雑化が著しい．我々も，高齢者の行動を追従して，転倒を検出する高齢者見守りロボットを開発している．ロボット制御ソフトウエアの開発においては，機能ごとに別々のプログラムを開発し，それらのプログラム間でセンサ値や制御命令などのデータを通信することで，互いに連携できるよう支援するミドルウエアが用いられる．ROS（Robot Operating System）は，通信機能が単純で，サポートされる機器も豊富であり, 普及している一方，高性能な PC の Linux 上で動作することを前提としており，性能が限られるシステムで動作させる際には性能と安全性の課題がある．本論文では，単一コンピュータ上における複数プロセス間の通信に着目し，ROS の提供する通信機能の高速化，リアルタイム性向上や安全性向上を実現する IXM（Information eXchange Middleware) を提案する．IXM は，共有メモリを用いて，高速かつリアルタイム性の高い通信を実現する．機能と通信性能の観点で評価し，IXM が，組込みシステム向けのロボット制御ソフトウエアの通信ミドルウエアとして適することを示す．

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

1. はじめに

近年，少子高齢化に向けて AI ロボットが医療，介護，農業，建設など様々な分野での活躍が期待される中，ICT 技術との連携が必須との認識が高まっている [1]．AI などを搭載する高機能なロボットでは，データ解析やモータによるアクチュエータ制御などの機能を並行して行うことで，動作を効率化させることが一般的である．これらの機能は主にマイクロコンピュータなど計算機で実現されていることからロボットは，基本的には接続された複数のコンピュータが互いに通信する分散システムといえる．分散システムでコンピュータが通信を介して接続する場合，外部のネットワークを通じた接続をする場合と，コンピュータ内部で接続する場合がある．後者は，車やロボットなど，高度に密結合したシステム内部で利用される．そのため，コンピュータ数の制約や，センサやアクチュエータの配置等を考慮した設計が必要である．

特に，インタラクティブな動作を期待されるロボットでは，センサからの環境情報の入力と，それに対する動作（アクチェーション）を実現するには，入力ソースと出力制御の間でデータを共有する必要がある．データを共有するためには，その過程におけるコンポーネント間のデータ通信が発生する．これらの部品の制御や，制御に必要なデータ通信においては，応答性能，低リソース消費，ノード数増加に対するスケーラビリティが求められる．これらを満たすために，データ中心指向で非同期の通信が行われることが多い．こうした要求を満たすために，従来 RT ミドルウエア [3] や ROS[2] などのロボットに特化した通信ミドルウエアが提案されている．

ハードウエアの汎用性が高く，標準的なインターフェイスの仕様を満たせば通信が可能な RT ミドルウエアに対して，オープンソースの資産と，スケーラビリティに優れた ROS は多くのロボットで採用されている．特に，ROS はデータ共有における Publisher/Subscriber（以降，Pub/Sub）通信を用いる事で，ロボットに特有の複数のアプリケーションによるデータ共有を効率的に行える利点がある．しかし，ROS の実装は汎用性を実現するため，コンピュータ間での通信と同じようにソケットで実装されており，ロボット内部のデータ通信には，応答性能が劣る問題がある．この問題に対し，ROS では，送受信ノードを同一プロセスの別スレッドで実装する nodelet[3] の仕組みが提供されている．しかし，nodelet は，処理のオーバヘッ

[1] 芝浦工業大学（現在，東洋電装 (株)）
[2] 名古屋大学
[3] 産業技術大学院大学
[4] 芝浦工業大学

ドが大きく，異なるアプリケーションを同一プロセスで実現するために，送信側と受信側の安全性を維持するのが難しい問題がある．例えば，親プロセスが異常終了した場合に，送受側と受信側の双方のノードに影響することや，スレッド間でのメモリ空間の共有は，バグによる意しないメモリアクセスの防止が困難である．このことは，システムの基本機能や安全性に影響を及ぼす深刻な問題である．

本研究では，単一コンピュータ上で実装されるロボット制御システムを対象に，単一コンピュータ内における通信ミドルウエアの設計と実装を提案する．具体的には，ロボット開発を通じて得られた経験を元に，コンピュータ内通信に対する 3 要件を定め，これらを満たす通信ミドルウエア IXM（Information eXchange Middleware）を設計，実装し，評価することを目的とする．

- 要件 (1)：単一コンピュータ内での Pub/Sub 通信をサポートすること．シングルコア上での通信と，マルチコア間での通信の 2 種類をサポートする．
- 要件 (2)：出来る限り，通信におけるオーバヘッドやリソース消費を抑えること．性能が限られる組込みシステムにおいては，オーバヘッドを出来る限り小さくすることが求められる．
- 要件 (3)：送信側と受信側の独立性を維持すること．片方のプロセスの停止や，（意図しない）バグなどによって，両方が停止し，システムの安全性に影響を及ぼさないようにすること．意図する破壊は対象外とする（セキュリティは考えない）．

開発した IXM では，設計段階で指定したメモリ領域のみを，送信側と受信側の間で共有して通信する，共有メモリベースのデータ通信機構を提供する．提案するデータ通信機構を実装し，ROS，nodelet と機能，応答時間の観点から比較，評価する．

本論文の構成は以下の通りである．まず，第 2 章にて，関連研究について述べる．第 3 章では，コンピュータ内通信の基本性能を明らかにするために，共有メモリを用いた通信と，ソケットを用いた通信の通信時間を測定した結果について述べる．第 4 章では，提案する通信ミドルウエア IXM の詳細について述べ，第 5 章では，性能測定として，通信性能の測定結果及び nodelet との比較を詳細に分析する．第 6 章では，まとめと今後の課題について述べる．

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] は，カメラやセンサ，モータなどを制御するソフトウエアコンポーネントの通信規格を定めたものである．そのため，この規格のもとに開発されたミドルウエア実装が利用される．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 ミドルウエアの通信モデルを図 1 に示す．RT コンポーネントの規格上，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分散システムを前提としている．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 ミドルウエアの実装の 1 つである OpenRTM-aist[9] は，CORBA を用いて実装されており，様々な OS の実行環境で動作する．これを用いれば，RT コンポーネントを比較的容易に利用できる．ハードウエアの入力をコンポーネントからアクセスし，出力に渡す処理は，read，write の入出力で行うことができ, 必要に応じてコールバック関数により拡張が可能である．しかし，複数の接続コンポーネントに統一的にデータを送付する仕組みについては，ROS のような pub/sub モデルなどが考慮されていないことから，効率を考慮したデータ処理を行う必要がある．また，コンポーネント間のデータ共有通信に関しては，仕様上の制限がないことから，個別に実装する必要がある．仕様上開発者の裁量が大きい反面，アプリケーション開発に専念したい場合には負担も大きい．本稿では，複数のデータ共有を目的とするので，RT ミドルウエアは適用できないと判断した．

2.2 ROS（Robot Operating System）

ROS は，ロボット制御ソフトウエア開発のためのツールや，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 [6]．具体的には，ハードの抽象化，デバイスドライバ，ライブラリ，視覚化ツール，メッセージ通信，パッケージ管理機能などが提供されている．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 では，図 2 に示すように，プログラム間を Pub/Sub 通信で実現する．あるアプリケーションの機能をノード（Node）として実現する．ノードは，トピック（Topic）を介して通信を行う．トピックとは，複数のノードが共有するデータ領域である．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，トピックを講読（Subscribe）を行うことで通信が成立する．図 2 においては，Node1 は，Topic1 と Topic2 に対しデータを出版し，Node2 と Node3 は，Topic1 から講読する．Node3 と Node4 は，Topic2 を購読する．このように，トピックを介し, 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，操作対象のトピックのみ把握すればよく，通信相手のノードを意識せず通信することができる．

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ヘッド [7] がボトルネックとなる．これに対し，ROS は，nodelet という機能を提供している．nodelet は，ノードをスレッド単位で実行し，ノード間で

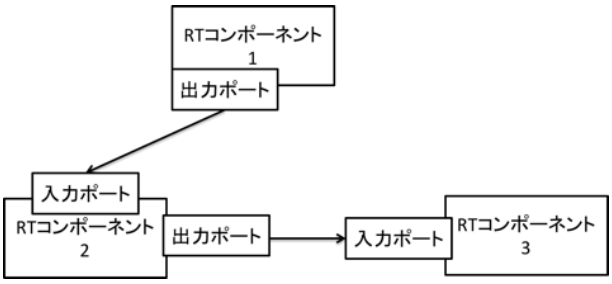

図 1　RT ミドルウェアの通信モデル
Fig. 1　Communication model of RT-Middleware.

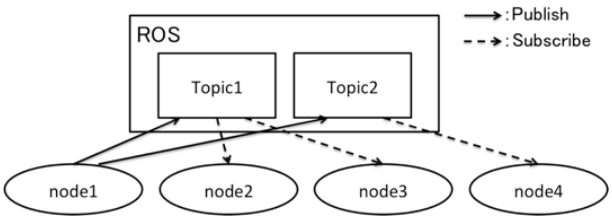

図 2　Publisher/Subscriber モデル
Fig. 2　Publisher/Subscriber model.

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，ソケット通信も不要なためデータ共有速度が速い．

しかし，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ヘッドが発生する問題がある．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．

ROS のコミュニティにおいても，コンピュータ内通信におけるオーバヘッドの大きさは問題視されており，次期バージョンである ROS 2.0 では，OMG によって仕様が策定されている DDS（Data Distribution Service）[4] を ROS の通信機構として採用することが検討されている．この方向性は，我々の問題意識と提案の正当性を裏付けるものである．現時点においては， ROS と DDS との連携は，開発段階である．例えば，RTI 社 RTI Connext は，共有メモリによる通信をサポートする DDS 実装の 1 つであるが，ROS との連携機構は開発段階であり，標準的に使用されている状況ではない．また，RTI Connext 自体が商用であることも，オープンソースをベースとする開発を進める上では，1 つの問題である．

3. 予備実験

3.1 目的

本研究では，ROS の nodelet に代わる通信機能として，共有メモリによるデータ通信を用いる方法を検討する．共有メモリによってどれだけ通信時間の高速化が見込めるかを確認するため，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を測定し，比較した．

3.2 方法

データ送信プログラムとデータ受信プ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 を呼んだ直後とした．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 とした．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 1,000 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．

実験環境として，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．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．単位は，マイクロ秒である．表 1 のように，全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．また最悪値も遅いことから，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，共有メモリは 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．

表 1　データ共有時間比較（単位：μs）
Table 1　Data sharing time (Unit: μs)

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

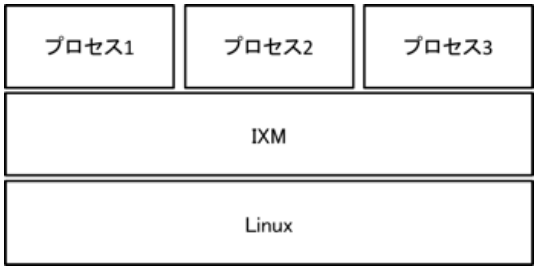

図 3　IXM のソフトウエア構成
Fig. 3　Software configuration of IXM.

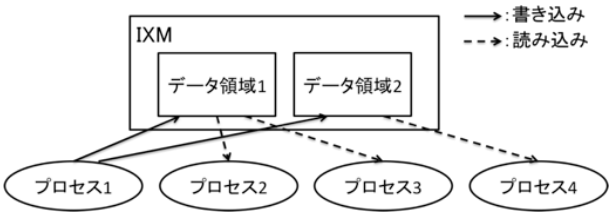

図 4　IXM の通信モデル
Fig. 4　Communication model of IXM.

4. ロボット制御向けミドルウエア IXM

4.1 概要

データ共有処理時間の高速化と，安全性の向上を目的とし，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．IXM を使用する際のソフトウエア構成を図 3 に示す．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．

IXM の通信モデルを図 4 に示す．IXM は，データを送受信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を持たせ，それを IXM キーとした．プロセスはその領域に対してデータの書込み，読込みの 2 つの操作が可能である．1 つのデータ領域に対し複数のプロセスが非同期で操作を行うことができるものとした．各プロセスは自身が操作するデータ領域のみを把握し，共有メモリを介した受信先プロセスを把握する必要はないものとした．

4.2 設計

IXM を構成するプロセスの設計を図 5 に示す．IXM の核となる ixmcore プロセスを設計，実装し，このプロセスが共有メモリを管理するプロセスや，ロボット制御プロセスの実行を行うものとした．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 キーを保持する．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，サイズ（要素数）を格納する．

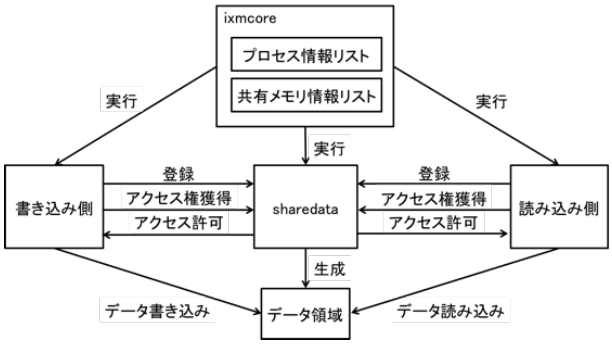

図 5　IXM のプロセス設計
Fig. 5　Design and process of IXM.

共有データを管理する sharedata プロセス，データの書き込み側プロセス，読み込み側プロセスは，それぞれ ixmcore の子プロセスとして実行される．この時，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．

4.3 データ共有時の動作

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動作例を図 6 に示す．多対多通信の場合も手続きは同様である．図 6 に，(1)-(12) の番号順に通信の流れと，その時使用する API を示した．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サイズ（要素数）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

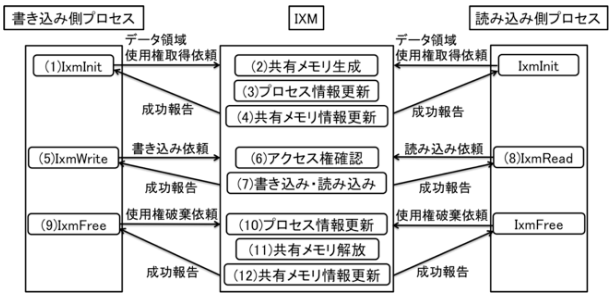

図 6 IXM における 1 対 1 通信のシーケンス図
Fig. 6 Sequence diagram for one-to-one communication in IXM.

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) データ領域が全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．(1) について，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込みと読込みを行う．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，更新されたタイミングでコールバック関数を呼び出す方法と，購読側プロセスの任意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 2 つが提供されている．ROS:nodelet では，前者の方法のみ提供されている．

(2) については，ROS:socket は，プロセス間で TCP/IP によるソケット通信行うため，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，スレッドで実行され，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は速い．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．

表 2 IXM と ROS の機能比較
Table 2 Feature comparison table of IXM and ROS.

	IXM	ROS:socket	ROS:nodelet
(1) 同期／非同期	非同期	非同期	非同期
(2) 実装方法	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) コンピュータ間通信	×	○	×
(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)

その他，(3) について，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 と ROS: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，コンピュータ間でのデータ共有には対応していない．(4) については，IXM が C，ROS が C++，Python となっている．(5) では，IXM の方が少ないが，増やすよう拡張することは容易である．

5.2 データ共有時間の比較実験

5.2.1 実験方法

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．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，データの書込み側プロセスと読込み側プロセスの間で，それらの操作を許可するための許可用データ領域を生成し，ポーリングを 10μs 周期で行った．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．

ROS:nodelet のプログラムは，基本的に購読者を実装する仕組みになっており，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

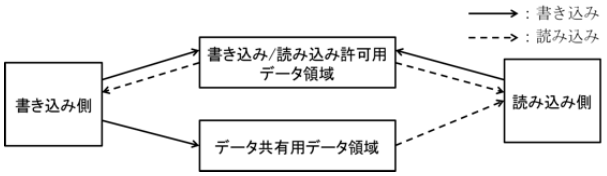

図 7　IXM におけるデータ共有方法
Fig. 7　Data sharing in IXM.

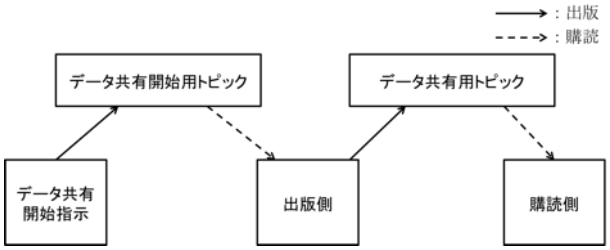

図 8　ROS:nodelet のデータ共有方法
Fig. 8　Data sharing mechanism in ROS:nodelet.

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，通信終了毎に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．

5.2.2　実験結果と考察

IXM と ROS:nodelet について，1000 回計測したデータ共有時間の平均値，中央値，標準偏差値，最速値，最悪値を表 3 に示す．さらに，図 9 に各条件のデータ共有時間の分布を示す．まず，(1)〜(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) と (3) の結果が変わらないので，プロセスの優先度は，データ共有時間に関係していないと考えられる．(4)〜(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) と (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，安定性を保つと同時に，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 10 に示す．グラフ

表 3　IXM と ROS におけるデータ共有時間比較 (単位:μs)
Table 3　Comparison of Data sharing between IXM and ROS. (unit: μs)

ミドルウェア	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

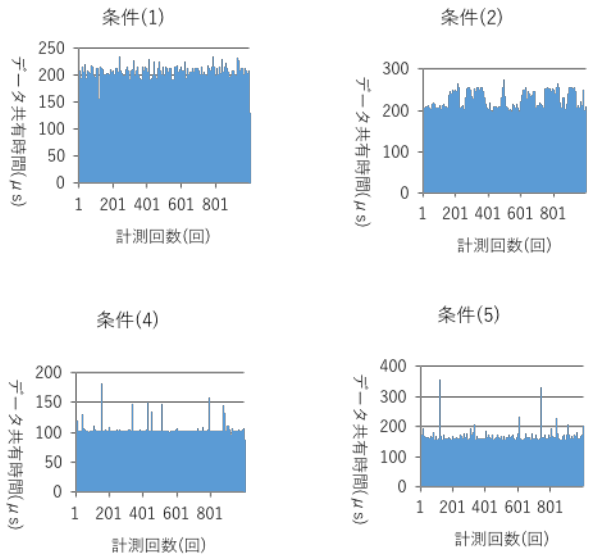

図 9　データ共有時間の変動
Fig. 9　Fluctuation of data sharing time.

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．

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．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．

実験環境は予備実験と同様である．プロセスやスレッドのマイグレーションによる影響を防止するため，プロセスやスレッドが動作する CPU を 1 つに固定した．さらに，変数への代入やファイル出力などの計測オーバヘッドを算出し，計測結果から差し引いた．

5.2.3　オーバヘッドの算出

各オーバヘッドの種類を以下に示す．

(a) 変数への代入：a = b
(b) 演算を含む変数への代入：a = b*c + d*e

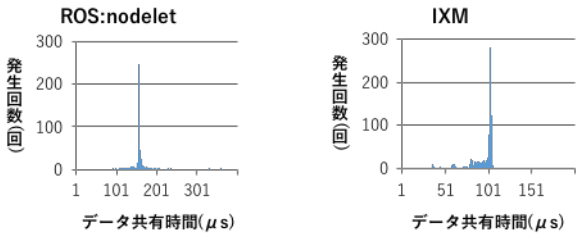

図 10　IXM と ROS:nodelet のデータ共有時間のヒストグラム
Fig. 10　Histgrams of data sharing time in IXM and ROS:nodelet.

表 4　各オーバヘッドの処理時間 (単位:μs)
Table 4　Processing time of each overhead. (unit: μs)

	a	b	c	d	e	f	g
平均値	0	0	7	8	1	2	14

表 5　IXM の書込み側内部処理時間 (単位:μs)
Table 5　Processing time of writing messages in IXM. (unit: μs)

処理	中央値	標準偏差値	最悪値
プロセス, 共有メモリ情報をアタッチ	20	3	59
共有メモリをアタッチ	11	1	35
データ書き込み	4	1	18
共有メモリをデタッチ	8	1	29
プロセス, 共有メモリ情報デタッチ	10	1	28
ロック解除	159	67	2258

(c) 標準出力：printf(“%.3lf\n”, a)
(d) 演算を含む標準出力：printf(“%.3lf\n”, a*b + c*d)
　ファイル出力：fprintf(fp, “%.3lf\n”, a)
(e) 演算を含むファイル出力：
　fprintf(fp, “%.3lf\n”, a*b + c*d)
(f) ファイルオープン, ファイル出力, ファイルクローズ：
　・fopen(fp), printf(fp, “%.3lf\n”,a), fclose(fp)

以上のオーバヘッドの時間を表 4 に示す．時間の算出方法は, 以上の処理の時間計測を 1000 回行い, 結果の平均値と中央値を取った．すべての処理において平均値と中央値が近いため平均値を用いるものとした．

5.2.4　実験結果と考察

図 11，図 12 に IXM と ROS のそれぞれの内部処理を示す．青色は，書込み側，赤色は読込み側を示す．各処理の横にオーバヘッドの種類を示す．IXM と ROS:nodelet のそれぞれの書込み側，読込み側の内部処理時間を，表 5 から表 8 に示す．標準偏差値が大きく平均値と中央値に差が見られたため，中央値を使用した．

IXM について中央値を見ると，ロック解除処理が最も長く，159μs であった．ロック解除の判断がポーリングであることが，遅延の原因になっていると考えられる．次は，読み込んだデータを変数へ代入する処理で，63μs であっ

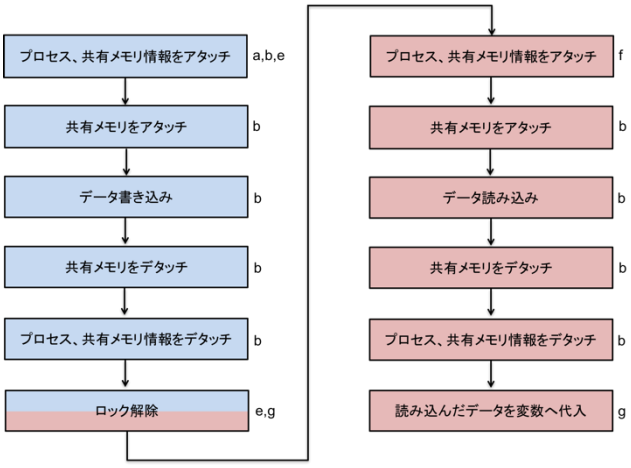

図 11　IXM の内部処理時間
Fig. 11　Processing time in IXM.

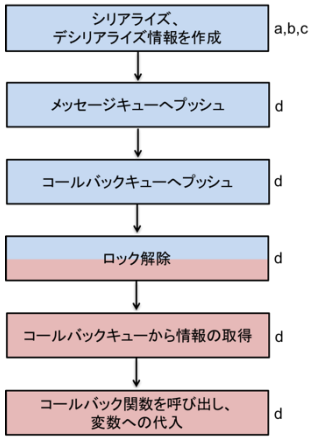

図 12　ROS:nodelet の内部処理時間
Fig. 12　Pprocessing time in ROS:nodelet.

表 6　IXM の読込み側内部処理時間 (単位:μs)
Table 6　Processing time of reading messages in IXM. (unit: μs)

処理	中央値	標準偏差値	最悪値
プロセス, 共有メモリ情報をアタッチ	18	3	48
共有メモリをアタッチ	10	2	18
データ読み込み	3	1	5
共有メモリをデタッチ	6	2	18
プロセス, 共有メモリ情報をデタッチ	10	3	22
読み込んだデータを変数へ代入	63	415	1635

表 7　ROS:nodelet の書込み側内部処理時間（単位:μs）
Table 7　Processing time of writing messages in ROS:nodelet. (unit: μs)

処理	中央値	標準偏差値	最悪値
シリアライズ, デシリアライズ情報を作成	131	27	224
メッセージキューへプッシュ	13	9	86
コールバックキューへプッシュ	30	14	124
ロック解除	127	50	234

表 8　ROS:nodelet の読込み側内部処理時間 (単位:μs)
Table 8　Processing time of reading messages in ROS:nodelet. (unit: μs)

処理	中央値	標準偏差値	最悪値
コールバックキューから情報の取得	49	18	169
コールバック関数を呼び出し，変数への代入	55	20	134

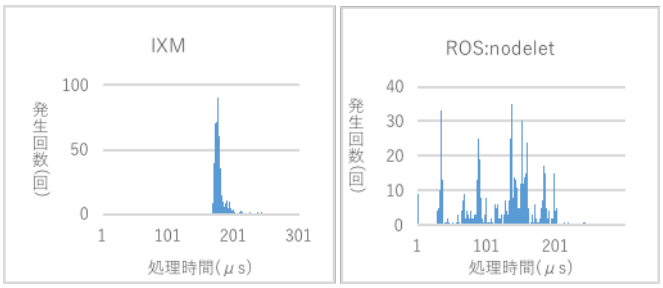

図 13　ロック解除の処理時間のヒストグラム
Fig. 13　Histogram of lock release processing time.

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．これに対して，ROS:nodelet の中央値は，シリアライズ，デシリアライズ情報作成処理，ロック解除処理に時間がかかっている．前者は，オブジェクトの生成と，生成したオブジェクト操作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．これは，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 の中央値について比較をすると，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，ポーリング処理と，処理内に計測結果を出力する処理が影響していると考えられる．

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たと考えられ，またそれが標準偏差値と最悪値に影響していると考えられる．IXM 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，その影響が顕著に現れたと考えられる．

IXM，ROS:nodelet のロック解除の処理時間のヒストグラムを図 13 に示す．外れ値である IXM の最悪値 2258μs は，除外した．IXM は，表を見ると最悪値が大きく外れているため標準偏差の値が大きくなってしまっているが，図を見ると，ばらつきは小さいことがわかる．処理時間の中央値を比較すると大きな差はないので，ポーリングにより CPU 使用率を消費するよりも，ミューテックスオブジェクトを獲得するまで待機して，その間別の処理を行う方が効率は良いと考えられる．

6. まとめ

本研究では，複数プログラム間でのデータ共有処理の高速化を図るとともに，バグによる影響を防止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．IXM と ROS とで機能比較を行い，IXM が 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することで IXM の有効性を示した．

今後の課題としては，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] 総務省 情報通信審議会 情報通信技術分科会 技術戦略委員会 重点分野 WG, ”人工知能 ロボット アドホックグループ検討結果とりまとめ”, 4 月, 2015.
[2] Morgan Quigley, et al., ”ROS:an open-source Robot Operating System”, ICRA Workshop on Open Source Software, 2009.
[3] Morgan Quigley, et al., ”Programming Robots with ROS 1st ed”, O’ Reilly, pp31-49, 2015.
[4] Enrique Fernandez, et al., ”Learning ROS for Robotics Programming 2nd ed.”, Packt Publishing, pp.35, 2015.
[5] ROS on DDS, http://design.ros2.org/, (参照 2016-10-28).
[6] 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, 113(248), pp.15-20, 2013.
[7] 小倉崇, ROS ではじめるロボットプログラミング, 工学社, 2015.
[8] Graig Hunt, 村井純, 安藤進, ”TCP/IP ネットワーク管理 第二版”, O’ Reilly, p3-24, 1998.
[9] OpenRT-aist, http://www.openrtm.org/, (参照 2016-10-28)