情報処理学会論文誌 Vol.58 No.10 1578–1590 (Oct. 2017)

IXM：ロボット制御ソフトウェア向けプロセス間通信ミドルウェア

菅谷 みどり[1] 松原 豊[2] 住谷 拓馬[1,†1] 中野 美由紀[3]

受付日 2016年12月23日, 採録日 2017年7月4日

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．我々は，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ェ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ェアが用いられる．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ることことから急速に普及している．その一方で，Linux が動作する高性能コンピュータを用いることを前提としているので，性能が限られる組込み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，リアルタイム性の向上，安全性の向上を実現する 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ェアの通信ミドルウェアとして適することを示す．

キーワード：ロボット，ミドルウェア，ROS，Robot Operating System，プロセス間通信，ロボットミドルウェア

IXM: Rapid Inter-process Communication Middleware for Robotics Software

Midori Sugaya[1] Yutaka Matsubara[2] Takuma Sumiya[1,†1] Miyuki Nakano[3]

Received: December 23, 2016, Accepted: July 4, 2017

Abstract: In recent years, autonomous robots, such as service robots, rescue support robots and drones, are becoming more advanced and complicated. We are developing an elderly watching robot that tracks the behavior of elderly people and detects falls. In the development of robot control software, middleware is used to develop different programs for each function, and support data such as sensor values and control instructions to communicate with each other so that they can cooperate with each other. The ROS (Robot Operating System) has been rapidly spreading because the communication function is simple and there are a lot of supported devices. On the other hand, since a high-performance computer running Linux is required, there are problems of performance and safety when operating in an embedded system with limited performance. In this paper, we focus on communication between multiple processes on a single computer and propose a middleware IXM (Information eXchange Middleware) that realizes high-speed communication function, improved real-time property, and a mechanism to reduce an influence of software bugs. It realizes high-speed, real-time communication with shared memory. Evaluation from the viewpoint of function and communication performance, we show that it is suitable as communication middleware of robot control software for embedded systems.

Keywords: robot, middleware, ROS, interprocess communication, data sharing

1. はじめに

近年，少子高齢化に向けてAIを搭載するロボットが医療，介護，農業，建設など様々な分野で活躍することが期待されるなか，ロボットとICT技術との連携が必須となるという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ることから，ロボットは，接続される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．分散システムにおいて，コンピュータが通信を介して接続する場合，外部ネットワークを介して別のコンピュータと接続する場合と，制御システム内のコンピュータ間で接続する場合がある．後者は，自動車やロボットなど，高度に密結合したシステムで採用されているが，コストや重量，物理的な空間の制約があることから，コンピュータ数，センサやアクチュエータの配置に制約があり，これらを十分に考慮した設計が求められる．

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．データ共有においては，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．これらのコンポーネントの制御や，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，データ中心指向の考え方の基で，非同期な通信方式が採用されることが多く，これまでに，RTミドルウェア [3] やROS [2], [10] などのロボットに特化した通信ミドルウェアが提案されている [11]．

RTミドルウェアは，通信を実現するための標準インターフェイスの仕様を規定しているが，その実装方法には自由度があり，ハードウェアへの依存性や，既存のソフトウェア資産が少ないことが問題となっている．それに対して，ROSは，ハードウェアの依存性が低く，オープンソースの資産と，スケーラビリティに優れていることから，多くのロボットで採用されている．ROS [10] は，自律ロボットソフトウェアの効率的な通信を目指しており，Publisher/Subscriber（以降，Pub/Sub）通信モデルを採用し，ロボットに特有の複数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．しかしながら，ROSでは，汎用性を重視して，単一コンピュータ内での通信においても，コンピュータ間通信と同じように，ソケットで実装されている．そのため，単一コンピュータ内でのデータ通信性能とリアルタイム性が低下するという問題がある．この問題に対し，ROSでは，拡張実装として，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，異なるアプリケーションを同一プロセスで実現するので，送信側アプリケーションと受信側アプリケーションの独立性を維持するのが難しくなる．たとえば，本来は独立して動作を継続することが期待される，送信側と受信側のアプリケーションが，親プロセスの異常終了によって同時に影響を受けたり，片方のアプリケーションのバグによって，共有するメモリ空間内のデータへのアクセス，破壊が発生する可能性がある．これらの事象は，システム全体の制御機能や安全性に影響を及ぼす深刻な問題となる．

本研究では，単一コンピュータ上に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ェ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以下の3要件を定め，これらを満たす通信ミドルウェアIXM（Information eXchange Middleware）を設計，実装し，評価することを目的とする．

開発したIXM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．本論文の内容は，当初は，独立したミドルウェアとして提供することを想定していたが，実用を考慮すると，ROSとの連携や，ROS自体の実装を改善する際の有益な情報を提供するものであると考える．

本論文の構成は以下のとおりである．まず，2章にて，関連研究について述べる．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．4章では，提案する通信ミドルウェアIXMの詳細について述べ，5章では，性能測定として，通信性能の測定結果およびnodeletとの比較を詳細に分析する．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ェア

2.1 RTミドルウェア

RTミドルウェア [5] は，カメラやセンサ，モータなどを制御するソフトウェアコンポーネントの通信規格を定めたものである．この規格に準拠するミドルウェアが実装され，公開されている．RTミドルウェアというのは，規格

[1] 芝浦工業大学
Shibaura Institute of Technology, Koto, Tokyo 135–8548, Japan
[2] 名古屋大学
Nagoya University, Nagoya, Aichi 464–8603, Japan
[3] 産業技術大学院大学
Advanced Institute of Industrial Technology, Shinagawa, Tokyo 140–0011, Japan
[†1] 現在，東洋電装株式会社
Presently with Toyodenso Co., Ltd.

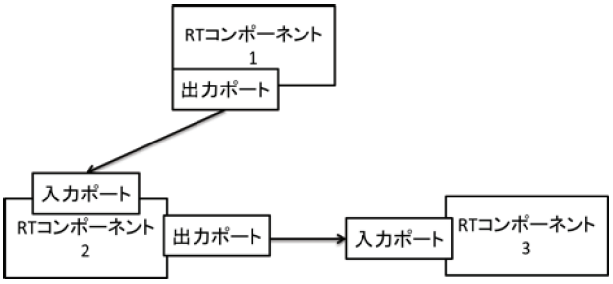

図 1 RT ミドルウェアの通信モデル
Fig. 1 Communication model of RT-Middleware.

に準拠して開発されたものの総称である．

RT ミドルウェアの通信モデルを**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 ミドルウェアの実装の 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．ハードウェ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間で統一的にデータを送受信する仕組みについては，開発者自身が，効率を考慮してデータ処理を行う必要がある．たとえば，コンポーネント間で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，開発者の自由度が高い反面，アプリケーション開発に専念したい場合には開発負担も大きい．

RT ミドルウェア実装の一部は，通信プロトコルに TCP/IP を利用した CORBA 実装により，分散オブジェクトの機能を実現しており，従来よりリアルタイム性能を十分に実現できていない問題が指摘されている [14]．これに対して，通信経路に CAN（Controller Arage Network）を利用した RT ミドルウェア [17] や，OpenART-aist をリアルタイム OS である ARTLinux 上に移植し，コンポーネントをリアルタイムで実行させる提案がなされている [18]．金広らは，ヒューマノイドロボット HRP において ART-Linux [15] を採用し，同時に Ethernet を通じて，分散したノードの情報を統合するための分散 I/O システムとして，RTOS の ART-Linux を用いて 1 [ms] 周期のセンサ情報を収集するタスクと，5 [ms] 周期の全身の運動制御タスクのデータ送受信を 1 [ms] で完了する実時間 CORBA を開発した [15]．しかし，継続的なソフトウェア提供は十分行われておらず，最新のドライバの提供，汎用 CPU，汎用 OS 上で，既存のソフトウェア資産を利用しつつ新しいロボットを開発する目的には，先に述べたように十分に対応できていない．

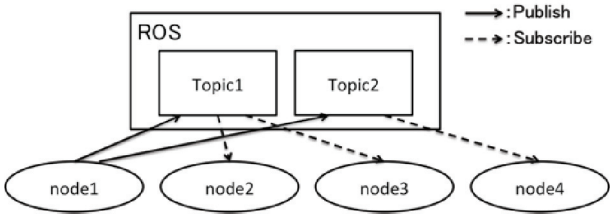

図 2 Publisher/Subscriber モデル
Fig. 2 Publisher/Subscriber model.

本論文では，単一コンピュータ内での複数アプリケーション間のデータ共有に着目しており，また，汎用的な OS 仕様により実現することを目指している．これらの理由から，RT ミドルウェアをそのまま適用するのは困難であると判断した．

2.2 ROS（Robot Operating System）

ROS は，ロボット制御ソフトウェ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ェアである [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．第 1 節で述べたように，ROS は，自律ロボットソフトウェアの効率的な通信を目指しており，データ共有における Publisher/Subscriber（以降，Pub/Sub）通信により，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．本論文では，特に 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通信する際に，その応答性能を要求する場合，ROS の提供するソケット通信は，データ共有時の通信オーバヘッド [7] がボトルネックとなる．この問題に対して，ROS では，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ただ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，通信ミドルウェ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，データ共有の仕組みの汎用性を重視した結果，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，データ共有時の処理オーバヘッドが大きいという問題がある．これについては，本論文でも評価する．安全性については，各通信ノードがスレッドで実装されるので，スレッド間でのメモリアクセスに制限がなく，意図しないアクセスを防止するのが困難である．

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，本論文で述べている問題提起と提案内容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．たとえば，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とはいい難い状況である．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うえでは，1 つの問題となっている．DDS の仕様は膨大であり，実装によって提供される機能の範囲，性能も異なると思われるので，提案方式との比較については，今後の課題であると考える．

3. 課題

3.1 課題の概要

1 節に述べたように，ソーシャルロボットにおいてはインタラクティブな性質上，リアルタイム性能が重要である [12]．こうしたリアルタイム性能を考慮する際に重要な点として，ロボットを構成するプログラムは，複数のセンサからの入力情報を扱う必要上，複数のプログラムが非同期的にデータを共有することが重要であり，このことから複数の通信先とデータの共有操作を保証するミドルウェアの役割が重要となる．ミドルウェアは，単一コンピュータ内での複数の処理プログラムをデータ共有を支援する必要がある上，インタラクションを支援するため，人への応答というリアルタイム制約を満たす必要がある．リアルタイム制約を満たすためには，データ共有支援機構となるミドルウェアにおいて，できるだけ処理のオーバヘッドが少なく，高性能な支援が求められる．また，様々なロボットのアーキテクチャへ対応するために，多様なハードウェアアーキテクチャへの対応が要求される．

本研究で想定した見守りシステムのロボットのインタラクティブなサービスでは，人への自然な反応のために 1 つのロボットの反応動作（サービス）が，1 秒以下で実現される必要がある．実現のためには，ハードウェアのセンシング処理，ミドルウェアの通信，制御プログラムの判定，更新，制御の指示，フィードバックなどの一連の動作を本時間内で実現する必要がある．特に，見守りシステムなどのロボットサービスを実現するには，再利用生などの観点から，これらの処理機能が独立したコンポーネントとして非同期に動作し，分散化した状態でデータ共有する形態が望ましい．通信によるデータ共有支援をミドルウェアが行うことで，開発コストを最小限にすることができる．ミドルウェアは，独立したコンポーネントの動作をつなぐ通信を支援するというアプリケーションを支援する機能であることから，最小限の時間で行うことが求められる．特に，通信対象のノード数が増える場合，こうした 1 回あたりの通信が安定した精度で，1 ms 未満の時間で行われることが重要である．

このことから，我々は本研究開発の目的を次の要件を満たすこととした．

- 要件 (1)：単一コンピュータ内でのデータ共有通信をサポートすること．
- 要件 (2)：通信におけるオーバヘッドを抑えること．性能が限られる組込みシステムにおいては，オーバヘッドを最小限と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，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（セキュリティの脅威）は，対象外とする．

3.2 目的

汎用性を実現するために，本研究では汎用 CPU，OS 上で，既存のソフトウェア資産を利用しつつ，新しいロボットを開発するミドルウェアとして IXM を提案する．従来の RT ミドルウェアは最新のドライバの提供，汎用 CPU，OS 上で，既存のソフトウェア資産の利用における課題や実現方法の課題，また，ROS は通信の安全性，安定性および性能の課題があると考え，新しいミドルウェアを提案するものとした．

IXM では，上記の要件を満たし，高速な通信を実現する方法として，共有メモリによるデータ通信を提案する．すでに，要件 (1) を満たし，(2) を目指すミドルウェアとして ROS が提案されている．ROS は，汎用性の高いソケット方式を採用している．ソケットは，スケーラブルなマルチタスク環境を想定した汎用性の高い仕組みである一方，

ロボット内部の分散化I/O処理を対象としたサービスの場合には，コストが大きいことから，要件 (2) について十分ではないと考えられる．また，通信コストを最小限にする共有メモリ方式との比較上のメリットが明らかにされていない．

共有メモリは，複数のプロセスが同時並行的にアクセスが可能でかつRAM上の領域をスタティックに確保することから高速にデータ共有する方法として有力である．同じRAM上のメモリの利用であっても，ヒープ領域などの利用と比較して，OSのメモリ取得のアルゴリズムやメモリ使用状態に依存せず，固定的にアクセスできるメリットがあり，これを中心として構成するミドルウェアにより，性能の安定性とリアルタイム性能の支援ができると考えた．特に，本研究で想定するようなロボットは，ロボット内部処理としてプロセスの並行実行の際の性能および，その安定性が保証されることによるリアルタイム性能の支援が重要であり，外部のコンポーネントと分散して通信する必要性は低い．本提案は，こうした要求に対応するものであると考えた．

本研究ではまず，実際に，共有メモリ方式を実装し，通信時間の高速化の程度を確認し，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比較を行う．次に，これを実現したIXMモジュールを開発しロボット内部処理向けのミドルウェアとして提供できるものとした．

3.3 予備実験

本節では，ソケット方式と共有メモリによる，データ共有における処理時間の比較を目的として予備実験を行う．本調査により，提案方式への実装を考察するものとした．

3.3.1 実験内容

データ送信プログラムとデータ受信プログラムの2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，ポインタのサイズを越えるサイズとして，24 byte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の差をデータ共有時間として算出した．これを通信1回分として，合計で1,000回計測した．

3.3.2 実験条件

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，各プログラムは，そのファイルを使ってポーリングによって送受信タイミングを合わせた．受信待ち処理では，処理時間が短い場合に，ポーリングを使用することは許容されるが，データ量が大きく処理時間が長いケースでは，プロセッサを無駄に使用することが問題となる．今回は，想定している見守りロボットにおいて，センサからの入力値が，座標軸情報など数バイトの小さなサイズであることを想定して，ポーリングでも問題ないと考えた．ソケット通信によるデータ共有では，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プロセスが待機状態となるので，共有メモリで実装したようなポーリングは行わず，recvを呼んだ後にsendを呼ぶものとした．

表 1 データ共有における処理時間の比較（単位：μs）
Table 1 Data sharing time (Unit: μs).

方法	平均値	中央値	標準偏差値
ソケット	22	20	7
共有メモリ	14	14	10

3.3.3 実験環境

ノート型PC（Intel Core i5 2.6 GHzの4コア搭載，メモリ4 GB）で，OSはubuntu12.04LTS（32 bit）を用いた．IXMがターゲットとする見守りロボットは，画像認識やメール配信が必要なので，ノートPCを実験環境として採用した．CPU性能の実行時の変化が通信性能に影響することを防止するため，CPUのパフォーマンス設定を行うことできるcpufreqを使って，CPUへの処理負荷に関わらず，最高クロック周波数で動作するよう設定した．今後，消費電力が低く，性能が限定される組込みボードを採用する場合，本論文の測定結果は，絶対値は変わってしまうが，その傾向を把握するうえで非常に有益であると考えている．

3.3.4 実験結果

ソケットと共有メモリによる，データ共有における処理時間の比較（平均値，中央値，標準偏差値）を**表 1** に示す（単位はマイクロ秒）．

表1のように，ソケットによるデータ共有時間は，平均値と中央値に差があり，標準偏差値はほぼ同等であるという結果が得られた．平均値と中央値の差がないことは，データ共有時間の絶対的な平均処理時間と安定性においては共有メモリの方が優れていることを示している．このことから，IXMの設計においても，動作の通信速度と安定性の向上の要求を満たすために，共有メモリを採用することとした．

4. ロボット通信支援ミドルウェアIXM

4.1 概要

本論文では，3つの要件を満たすロボット制御ソフトウェア向けプロセス間通信ミドルウェアIXM（Information eXchange Middleware）を提案する．IXMを使用する際のソフトウェア構成を**図 3** に示す．IXMは，ubuntuなどLinuxベースのOS上で動作する．データを送受信するプロセスは，IXM上で動作し，IXMが提供するAPIを利

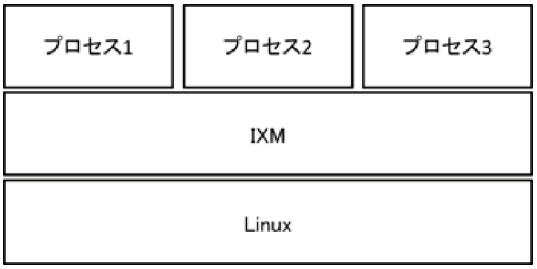

図 3 IXM のソフトウェア構成
Fig. 3 Software configuration of IXM.

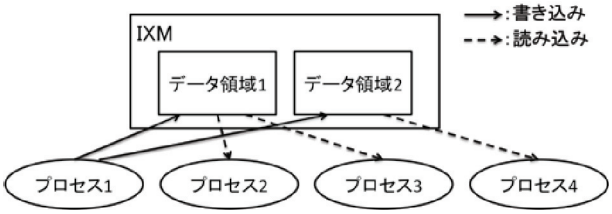

図 4 IXM の通信モデル
Fig. 4 Communication model of IXM.

用して通信する．

IXM の通信モデルを図 4 に示す．IXM では，データを送受信するための共有メモリ領域を管理する．システムの設計段階で，共有メモリ領域の識別番号（IXM キーと呼ぶ）とサイズを決定し，システムの動作開始段階で，IXM が設計情報に基づいて，実際に共有メモリ領域を確保，管理する．プロセスは，IXM キーを用いて，操作対象の共有メモリ領域に対して，データの書込みと読込みの 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．各プロセスは，設計段階で決定した操作対象の共有メモリ領域のみを把握しており，そこからデータを受信するプロセスを把握する必要はない．この通信モデルを採用することで，要件 (1) を満たすと考える．

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 プロセスは，共有メモリの管理と，共有メモリを使用してデータを送受信するプロセスを管理する．具体的には，プロセス情報リストと共有メモリ情報リストの構造体を生成する．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）が格納される．

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．このとき，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

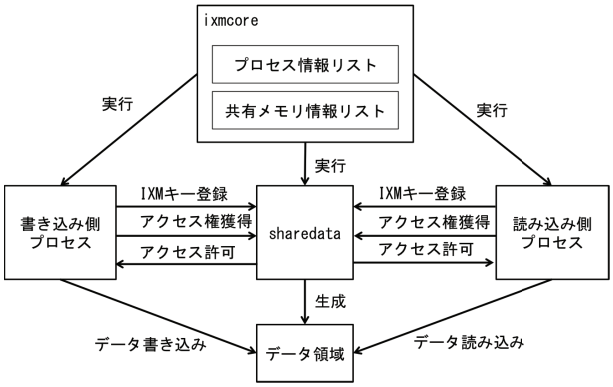

図 5 IXM のプロセス設計
Fig. 5 Design and process of IXM.

側プロセスは，まず自身が使う IXM キーを sharedata に登録する．正常なキーが登録されると，sharedata は共有メモリ用データ領域を生成する．書き込み側プロセスと読み込み側プロセスは，登録した IXM キーを使って，sharedata を経由してデータ領域にアクセスできるようになる．データ領域へのアクセスには，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．なお，ここでは，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，それらを明確に分ける必要はなく，設計情報にプロセスを登録すれば，読み書きできる．

ロボット制御アプリケーションにおいて，アプリケーションが（たとえば，OS のサービスコールを使用して）共有メモリを直接操作する場合，バグによって，別のプロセスの共有メモリ領域を破壊してしまう可能性がある．そこで，アプリケーションが直接共有メモリを操作することを禁止し，プロセス間の通信を IXM に統一して設計・実装するという考え方にすることで，共有メモリをアプリケーション自体で操作することがなくなり，IXM キーによる簡易的なアクセス管理によって，低オーバヘッドで，かつバグによる共有メモリの破壊を低減できると考えられる．

以上の仕組みよって，要件 (2) と (3) を満たすと考える．なお，共有メモリ操作以外のメモリ操作のバグ（たとえば，メモリ上にマップされている周辺回路を操作する際に，そのアドレス指定を大きく間違えて共有メモリを破壊する）や，意図する破壊（セキュリティの脅威）は，今回は対象外とする．このようなバグや脅威による破壊を防止するためには，MMU（Memory Management Unit）や MPU（Memery Protection Unit）などの専用ハードウェアとそれらを使った保護機能を導入することが考えられる．しかしながら，保護機能を使いこなすためには知識や経験が必要で，導入の敷居が高く，アプリケーション開発自体の負担が大幅に増加してしまう．よって，保護できる範囲は限定されるものの，容易に導入可能である点において，IXM の有用性は高いと考える．

4.2.1 API

IXM の要求に従い提供する API を下記に定義する．

- IxmInit()
  本ミドルウェアの起動，プロセス情報，共有メモリ情報の初期化，共有データ領域の確保
- IxmWrite(key, data_pointer)
  key の登録とアクセス権を取得後，データ領域にデータを書き込む
- IxmRead(key, read_buffer)
  key の登録とアクセス権利を取得後，データ領域から key で識別されるデータを read_buffer に読み込む
- IxmFree(key)
  サービス終了後，登録した key 番号の領域を解放する

以上の API を用いて IXM を利用できるものとする．大まかな流れとしては，IxmInit() により，ixmcore サービスを行うマネージャーが起動され，その後のプロセス同士のデータ共有時の動作を行うためのプロセスの生成，メモリ領域の確保と初期化などの操作を行ったのち，IxmWrite()，IxmRead() 関数により，その領域を用いたデータ通信を可能とする．key は，通信ごとのユニークな ID を作成する．IxmFree() は，key により特定されている部分を解放する．特にデータ共有時の動作については，次節に示す．

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** に示した．多対多通信の場合の手続きも，基本的には同様である．図 6 に，(1)–(12) の番号順に通信の流れと，そのとき使用する 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の破棄を 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．

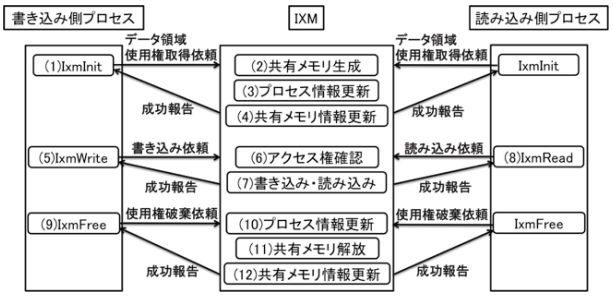

図 6 IXM における 1 対 1 通信のシーケンス図
Fig. 6 Sequence diagram for one-to-one communication in IXM.

5. 評価

本提案する IXM が 3 章の課題に示した要件を満たし，共有メモリによるデータ通信により高速な通信を実現することができていることを，評価により検証する．比較対象としては，ROS を用いて行う．ROS は，汎用性の高いソケット方式を採用している．ソケットは，スケーラブルなマルチタスク環境を想定した汎用性の高い仕組みである一方，ロボット内部の分散化 I/O 処理を対象としたサービスの場合いには性能のコストが大きい問題があることについてはすでに述べた．このことから，本節では主に IXM と ROS:nodelet を含めた比較を行う．また，結果と考察から，要件に示したデータ共有のオーバヘッド，保護機能の観点から結果を考察するものとした．

5.1 IXM と ROS の機能比較

まず，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

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(32 bit) float(64 bit)	int(8 bit-64 bit) float(32,64 bit)	int(8-64 bit) float(32,64 bit)

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．IXM ではプロセスレベルで共有メモリシステムコールでまずは共有する方式をとっていることから，メモリ不足環境ではメモリ空間がスワップアウトされる恐れがある．現状の IXM の適用対象となるロボットサービスは，ロボットの内部処理の高速化を目的としていることから，大量のデータ利用を想定していない．このことから，システムコールレベルでの共有で十分であると考えられる．ただし，今後内部処理の対象となるプロセス数が増大することが想定される場合には，ミドルウェア内部でアクセスのシリアライズや制御などを検討する必要があると考えられる．

一方，(3) について，ROS:socket は，TCP/IP によるソケット通信により実装されている．ソケット通信はデータ構造，メモリコピー，スタック制御などが OS 内部のプロトコルスタックで階層的に行われることから，一般的に処理オーバヘッドが大きい．しかし，ROS では，ネットワークを介し他のコンピュータと通信ができるメリットを重視しており [10] こうした方法が採用されている．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 となっており，ROS の方がオブジェクト志向言語を取り入れることで，再利用生や開発効率が高いといえる．一方，これらの言語を使いこなせない場合には，拡張が難しい問題がある．これに対して，C は現在でも最も利用されている言語であり，標準ライブラリが多く，他の言語との連携や OS との標準インターフェイスとの互換性も保証されており，広く拡張が容易であるといえる．

また，データ型の対応範囲が多い場合に備え，実装上にアーキテクチャやコンパイラにより拡張可能なマクロを用いることで，(5) の問題および，今後の拡張に対応できると考えられる．

要件と対比して結論を述べる．(1) より，要件 1 にある単一コンピュータ内でのデータ共有通信については，ROS と IXM ともに機能を満たしている．一方，要件 (2) のオーバヘッドの最小化については，ROS:socket はオーバヘッドが大きいが，ROS:nodelet のスレッドは性能が良いと想定される．しかし，要件 3 に示した保護機能が十分ではない問題がある．通信の独立性のについては，CPU 保護機能を利用しているプロセス実装の点で，IXM の方が安全性が高い．さらに，(4) は多様なアーキテクチャへ継続的に対応可能という点では，いずれも汎用性の高い言語を利用している．特に，両者も標準 C 言語ライブラリを利用可能であることから，多様なアーキテクチャへの対応条件を満たしていると考えられる．また，(5) についても，両者は C，C++という汎用性の高い言語の利用による拡張は容易であることから，両者にほぼ差はない．以上より，機能比較の観点からは，要件の保護機能の差において IXM の方が優れていると結論できる．

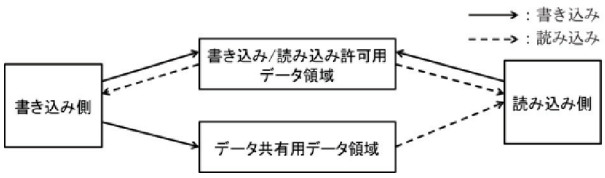

図 7 IXM におけるデータ共有方法
Fig. 7 Data sharing in IXM.

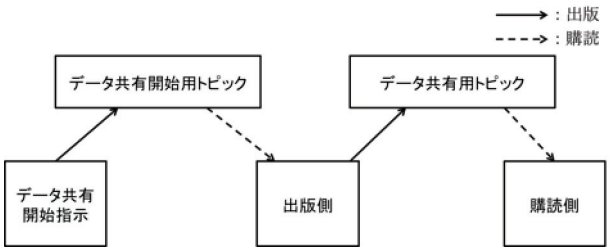

図 8 ROS:nodelet のデータ共有方法
Fig. 8 Data sharing mechanism in ROS:nodelet.

5.2 データ共有時間の比較実験

5.2.1 実験方針

図 7，**図 8** に示したように，IXM，ROS:nodelet の 2 つのデータ共有方式は，通信時にデータを送る側（生産者）と，そのデータを受け取る側（消費者）の 2 つの処理単位により構成されているという点では同一である．しかし，IXM では，データ共有に際して，データ管理領域のアクセスをこれらの 2 つ実際に行っているのに対して，ROS は，これら以外の処理単位と，非同期的にバックグラウンドで通信しつつ実行しているなど [19]，共有を実現するための実装に差が存在する．そのため，厳密な意味での通信回数での比較は困難である．そこで，条件を最低限揃えるために，少なくとも 1 対 1 通信において，共有操作にかかる開始から終了までの時間を対象として計測し，比較評価するものとした．

5.2.2 実験方法

実験方針に従い，基本的な性能比較を行うために，IXM，ROS:nodelet において，1 対 1 のプログラム間のデータ共有時間を比較する．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 2 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（64 bit）型の 8 byte 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 1 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 および 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 10 μs 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，Publisher 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．条件を揃えるために，本実験では，データ共有開始用 Topic を用意し，その Topic への書き込みをイベントとして動作するコールバック関数を，Subscriber プログラム内に作成した．コールバック関数内の処理では，データ共有用 Topic に書き込まれたデータを，Subscriber 側プログラムが読み込むものとした．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，通信終了ごと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．

5.2.3 実験結果と考察

IXM と ROS:nodelet について，1,000 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) CPU を固定（affinity）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と考えられる．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) CPU の固定が優れており，最悪値においては (1) が優れた結果となった．これは，コアのマイグレーションにより，CPU の占有率が高くなった結果であると考えられる．IXM と ROS:nodelet について，(2) と (5) で比較をすると，IXM の方が ROS:nodelet に比べ平均値は 56%，最悪値は 36%早い結果となった．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．このことから，共有メモリを実装した IXM の方がミドルエアとして，速度においてメリットがあるといえる．

データ共有時間のヒストグラムを**図 10** に示す．グラフより，最大発生回数については，IXM は約 100 μs が約 90 回発生している．ROS は約 200 μs が約 16 回，150 μ に集中して発生していることから，IXM の方が高速で，データ共有時間のばらつきが少なく，安定性に優れているといえる．リアルタイム性能を考慮する際に，処理のばらつきが少ないことは，予測可能性の性質を満たすうえで重要である．このことから，IXM の方がリアルタイム性を必要とするシステムに適しているといえる．

さらに，本研究では，IXM と ROS:nodelet においてデー

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	平均値	標準偏差値	最悪値
IXM	(1) なし	164	43	236
	(2) CPU を 1 つに固定	93	16	181
	(3) プロセス優先度最高	163	43	243
ROS:nodelet	(4) なし	210	23	284
	(5) CPU を 1 つに固定	155	14	357
	(6) プロセス優先度最高	351	54	620

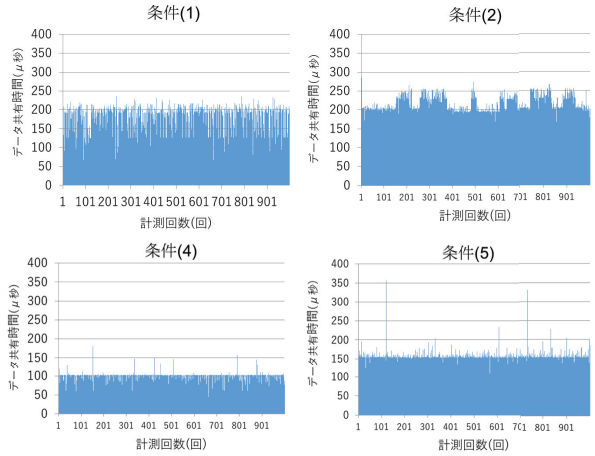

図 9 データ共有時間の変動
Fig. 9 Fluctuation of data sharing time.

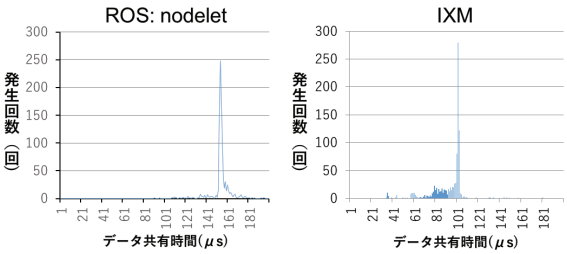

図 10 IXM と ROS:nodelet のデータ共有時間のヒストグラム
Fig. 10 Histgrams of data sharing time in IXM and ROS:nodelet.

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

タ共有時の内部処理の差異を明らかにするために，処理ごとに，処理時間を調査するマイクロベンチマーク実験を行った．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を計測した．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．

5.2.4 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
(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)
(g) 共有メモリ上のデータを変数に格納：
returndata = &tempdata; ※ tempdata は，共有メモリ上のデータへのポインタ

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 1,000 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．

5.2.5 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

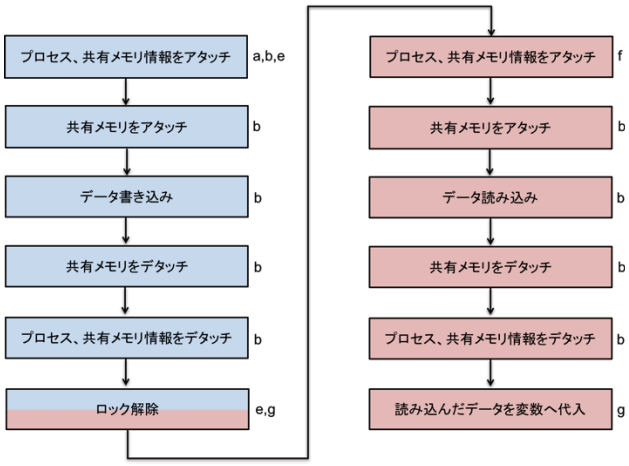

図 11 IXM の内部処理時間
Fig. 11 Processing time in IXM.

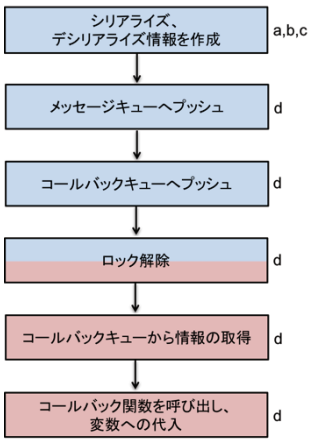

図 12 ROS:nodelet の内部処理時間
Fig. 12 Pprocessing time in ROS:nodelet.

表 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	2,258

のそれぞれの書込み側，読込み側の内部処理時間を，**表 5**，**表 6**，**表 7**，**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，159 μs 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，63 μs であった．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

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	1,635

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

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 2,258 μs 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分かる．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．

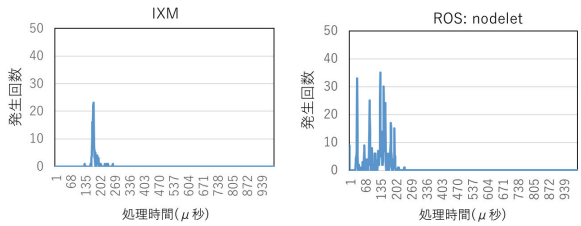

図 13 ロック解除の処理時間のヒストグラム
Fig. 13 Histogram of lock release processing time.

本節では，通信部分だけでなく，アプリケーションから通信機能を容易に利用できるようにするためのミドルウェアとして，IXM と ROS:nodelet と比較した．通信部分以外の処理において，IXM の処理は，データ共有に必要な共有メモリアクセス制御に関する最低限の関連手順により実現しているのに対し，ROS:nodelet はデータのシリアライズ/デシリアライズ，コールバック処理など，データ共有を汎用的に行うための処理が行われている．その結果，ミドルウェア全体の通信時間の評価結果としては，IXM の方が高速かつ通信時間の安定性が高いという結果をとなり，IXM の有効性を確認することができた．

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ェア向けミドルウェア 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ェ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．

今回，本研究では，主にデータ共有における安全性と，高性能な通信支援について実現を行った．しかし，IXM 自身は，ROS や RT ミドルウェアのように，コンポーネントそのものを提供するフレームワークや仕組みなどを提供していない．このことから，今後は実際の IXM を適用する際には目的に特化した形で利用するか，もしくは，ROS

との統合などの方法により実現することが重要である．現在我々は，IXM を ROS のデータ共有部分を高速に支援するコンポーネントとして利用可能とすることを検討しており，拡張可能である IXM を利用することでリアルタイム性能が必要となる見守りシステムなどにおいて，データ共有部分を安全かつ高性能に実現できると考えられる．

謝辞 本研究は JSPS 科研費 15K00105 の助成を受けたものです．研究費により，多くの議論やロボット購入の機会を得ることができました．研究を推進できましたこと，ここに改めて感謝申し上げます．

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 (2015).
[2] Quigley, M. et al.: ROS: An open-source Robot Operating System, ICRA Workshop on Open Source Software (2009).
[3] Quigley, M. et al.: Programming Robots with ROS 1st ed, O' Reilly, pp.31–49 (2015).
[4] Fernandez, E. et al.: Learning ROS for Robotics Programming 2nd ed., Packt Publishing, p.35 (2015).
[5] ROS on DDS, available from ⟨http://design.ros2.org/⟩ (accessed 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，Vol.113, No.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] グレイグ・ハント，村井 純，安藤 進：TCP/IP ネットワーク管理第二版，pp.3–24 (1998).
[9] OpenRT-aist, available from ⟨http://www.openrtm.org/⟩ (accessed 2016-10-28).
[10] Thomas, D. and von Stryk, O.: Efficient communication in autonomous robot software, *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems* (*IROS*), pp.1006–1011 (2010).
[11] Mohamed, N., Al-Jaroodi, J. and Jawhar, I.: Middleware for Robotics: A Survey, *2008 IEEE Conference on Robotics*, Automation and Mechatronics, Chengdu, pp.736–742 (2008).
[12] Mohammad, Y. and Nishida, T.: Toward combining autonomy and interactivity for social robots, *Journal of AI and Society*, Vol.24, No.1, pp.35–49, Springer-Verlag (2009).
[13] 金広文男，石綿陽一，齋藤 元，赤地一彦，宮森 剛，五十棲隆勝，金子健二，比留川博久：実時間 Ethernet を用いたヒューマノイドの分散 I/0 システム，日本ロボット学会誌，Vol.25, No.3, pp.466–477 (2007).
[14] 池添明宏，村永和哉，中本啓之，長瀬雅之：リアルタイム OS 向けの RT ミドルウェアの研究開発，日本機械学会ロボティクスメカトロニクス講演会（ROBOMEC2008），IPI-E05 (2008).
[15] 石綿陽一，松井俊浩：高度な実時間処理機能を持つ Linux の開発，第 16 回日本ロボット学会学術講演会予稿集，pp.355–356 (1998).
[16] 加賀美聡：ロボット研究のための PC/AT 互換機上のリアルタイム OS，日本ロボット学会誌，Vol.16, No.8, pp.1036–1041 (1998).
[17] 土屋 裕，水川 真，安藤吉伸，末廣尚士，安藤慶昭，中本啓之，池添明宏：RT ミドルウェアを用いた CAN モジュール構成型ロボットの構築，日本機械学会ロボティクス・メカトロニクス講演会 2006，1P1-C31 (2006).
[18] Yoon, W.-K., 安藤慶昭，末廣尚士，北垣高成，神徳徹雄：OpenRTM_aist による実時間制御を考慮した RT コンポーネント，ロボティクス・メカトロニクス講演会 2007，1P1-A01 (2007).
[19] ROS.org: Master, available from ⟨http://wiki.ros.org/Master⟩ (accessed 2017-04-28).

菅谷 みどり （正会員）

2004 年早稲田大学理工学研究科電子・情報通信学専攻修了．2010 年早稲田大学大学院理工学研究科情報・ネットワーク専攻修了．博士（工学）．同大学助手，科学技術振興機構（研究員），横浜国立大学を経て 2013 年芝浦工業大学工学部情報工学科科准教授（現職）．電子情報通信学会，IEEE，ACM 各会員．本会シニア会員．

松原 豊 （正会員）

1982 年情報処理大学理学部情報科学科卒業．名古屋大学大学院情報学研究科附属組込みシステム研究センター助教．2006 年名古屋大学大学院情報科学研究科博士前期課程修了．同大学大学院情報科学研究科附属組込みシステム研究センター研究員，特任助教を経て，2013 年より現職．組込みシステム向けのリアルタイム OS 等の研究に従事．博士（情報科学）．IEEE，USENIX，各会員．本会シニア会員．

住谷 拓馬

2016 年芝浦工業大学大学院理工学研究科修士課程修了．同年東洋電装株式会社入社．現在に至る．同社開発本部第四開発部第一 BL 所属．主に，自動車のスイッチおよびセンサのソフトウェア開発に従事．

中野 美由紀 （名誉会員）

東京大学理学部情報科学科卒業．博士（情報理工学）．富士通（株）勤務後，1985 年東京大学生産技術研究所助手（2004 年助教）．2008 年特任准教授．2013 年芝浦工業大学特任教授．2016 年産業技術大学院大学教授．データベースシステム，ストレージシステム，データ工学の研究に従事．IEEE，電子情報通信学会，ACM，日本データベース学会各会員．本会終身会員．