RT ミドルウェアのリアルタイム拡張

Real-Time Extension of RT-Middleware

○学　千代　浩之（慶大）　正　山崎　信行（慶大）
Hiroyuki CHISHIRO, Keio University, chishiro@ny.ics.keio.ac.jp
Nobuyuki YAMASAKI, Keio University

Modular component-based robot systems require not only an infrastructure for component management, but also scalability as well as real-time properties. Robot Technology (RT)-Middleware is a software platform for such component-based robot systems. Each component in the RT-Middleware, so-called “RT-Component” supporting particular robot functions, is based on Common Object Request Broker Architecture. Unfortunately, RT-Middleware cannot manage real-time information such as periods to guarantee real-time properties. This paper presents the real-time extension of RT-Middleware, which improves real-time performance than conventional RT-Middleware. Our real-time extended RT-Middleware manages the extended RT-Components, which can manage one or multiple tasks with same or different periods. Moreover, we add real-time information to General Inter-ORB Protocol packets for scheduling packets by priority. Experimental evaluations show that our real-time extended RT-Middleware improves schedulability than conventional RT-Middleware.

***Key Words*:** RT-Middleware, RT-Component, ART-Linux

1. 序論

近年，分散リアルタイム制御向けモジュール型ロボットの開発が盛んに行われている[1][2]．モジュール型ロボットはヒューマノイドロボット[3]と比較して，構成の変更が容易なため，ロボット開発者の多様な目的を実現することに適していると考えられる．このようなモジュール型ロボットを低コストで実現するためには，コンポーネントを基調としたミドルウェアが要求される．それゆえ，Ando らは Robot Technology (RT)ミドルウェア[4]を開発した．

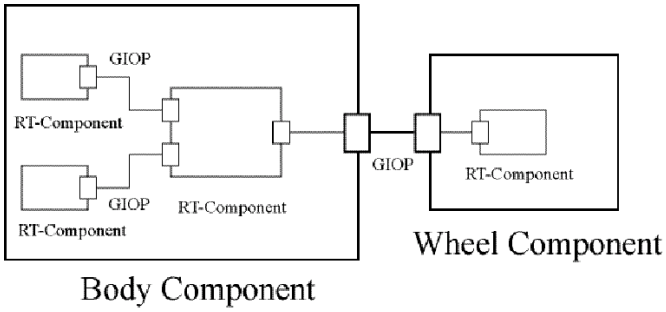

Fig. 1 RT-Middleware usage example for wheel robot

RT ミドルウェアは，モジュール型ロボットを開発するための共通のプラットフォームである．RT ミドルウェア上で実行するソフトウェアコンポーネントである RT コンポーネントを用いることで，ロボット開発者はロボットの開発コストを削減することが可能である．また，RT コンポーネント間のインターフェースの仕様として，Common Object Request Broker Architecture (CORBA) [5]を採用することで，CORBA 準拠のミドルウェアを利用しているロボット開発者は容易に RT ミドルウェアを用いたロボット開発に移行可能である．図 1 に車輪ロボット向け RT ミドルウェアの使用例を示す．車輪ロボットは胴体と車輪のコンポーネントで構成されている．胴体コンポーネントには 3 つの RT コンポーネント，車輪コンポーネントには 1 つの RT コンポーネントがある．各々の RT コンポーネント間は CORBA の抽象プロトコルである General Inter-ORB Protocol (GIOP)で接続する．

RT ミドルウェアにおいて RT コンポーネント間で通信を行う場合，リクエストキューで管理する GIOP パケットを FIFO で処理するため，優先度の高いリクエストのデッドラインミスが多発してしまう問題がある．我々は RT ミドルウェアの RT コンポーネント間の通信におけるリアルタイム性を向上させるため，RT ミドルウェアのリアルタイム拡張を行う．このリアルタイム拡張した RT ミドルウェア上で実行する拡張 RT コンポーネントは RT コンポーネントを基調とすることで，様々な恩恵を受けると考えられる．そこで，本論文では拡張 RT コンポーネントの性能評価を行う．

2. 背景

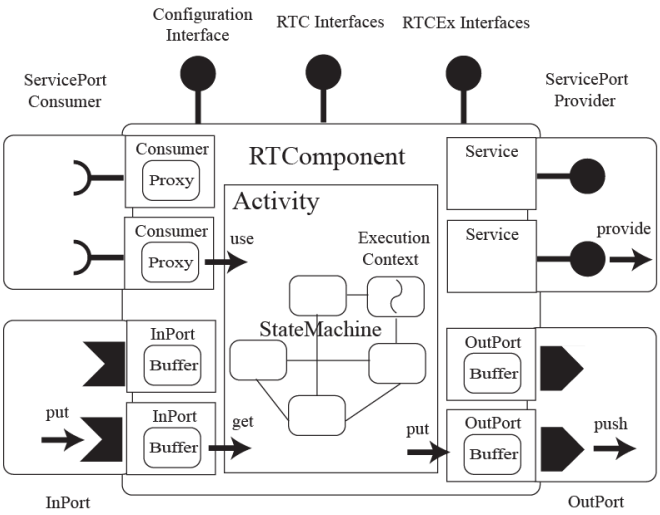

Fig. 2 Architecture of RT-Component

RT ミドルウェアを用いて，モジュール型ロボットを開発する場合，各々のモジュールは RT コンポーネントと呼ばれるソフトウェアコンポーネントを用いて開発し，各 RT コンポーネントを接続することで各モジュールをソフトウェアレベルで統合する．図 2 に RT コンポーネントのアーキテクチャを示す．RT コンポーネントは，タスクの状態を表す抽象スレッドである実行コンテキストを持つ．実行コンテキストは実行状態，待機状態等のように様々な状態に遷移する．また，RT コンポーネントは，データ指向の通信を行うデータポート，CORBA の Interface Definition Language で定義したインターフェースを用いて通信を行うサービスポートを持つ．ロボット開発者は，RT コンポーネントを用いてロボットを統合可能になるので，開発コストを削減可能になる．RT ミドルウェアは，基盤ミドルウェアである ADAPTIVE Communication Framework (ACE) [6]と CORBA 準拠のミドルウェアである omniORB [7]から構成される．

RT ミドルウェアの問題点として，リアルタイム性を保証す

るための情報を持たない．例えば，Rate Monotonic (RM)スケジューリング[8]は，周期を指標にしてタスクをスケジューリングすることでリアルタイム性を保証するが，RT ミドルウェアはこのような情報を持たない．それゆえ，RT ミドルウェアによる開発コストの削減だけでなくリアルタイム性の向上が要求される．

3. RT ミドルウェアのリアルタイム拡張

3.1 拡張 RT コンポーネントアーキテクチャ

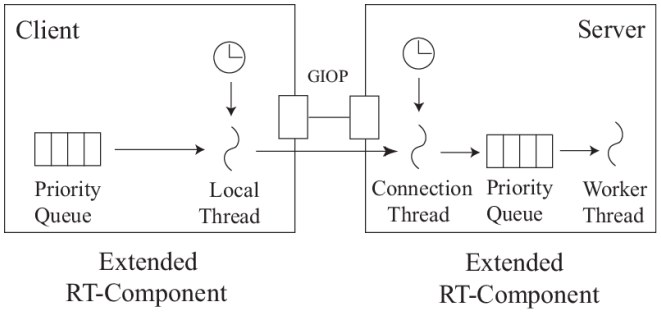

Fig. 3 Architecture of Extended RT-Component

図 3 に拡張 RT コンポーネントアーキテクチャを示す．従来の RT コンポーネントで設定可能な周期は 1 つである．しかしながら，図 1 に示すように，胴体コンポーネントが複数の RT コンポーネントを持つことがあり，これらの周期は異なる場合がある．そこで，拡張 RT コンポーネントは，複数の同じまたは異なる周期のタスクを持つことを可能にした．拡張 RT コンポーネント間の通信は従来の RT ミドルウェアと同様に GIOP で行うが，拡張 RT コンポーネント内のタスク間通信はグローバル変数等で行うことにより，オーバーヘッドを削減できる．拡張 RT コンポーネントはローカルスレッド，コネクションスレッド，ワーカースレッドの 3 種類のスレッドを持つ．ローカルスレッドは，ACE が管理し，RT コンポーネントにおける状態遷移を行う．すなわち，ローカルスレッドは，RT コンポーネントにおけるクライアント側の実行コンテキストに該当する．ローカルスレッドは，クライアントのリクエストを優先度順にサーバに送信する処理を行う．リクエストの優先度は RM と同様に，リクエストの周期が短いほど高いとする．この理由は，RM では，最短周期タスクのジッタが低いため，モジュール型ロボットの実現に適していると考えられるので，優先度をこのように設定した．コネクションスレッドは，omniORB が管理し，リクエストが到着しているか否かを一定周期毎にチェックする．リクエストが到着している場合，コネクションスレッドはそのリクエストをワーカースレッドに割り当てる．ワーカースレッドは，RT ミドルウェアが管理し，コネクションスレッドにより割り当てられたリクエストを実行する．リクエストがない場合，ワーカースレッドはリクエストが到着するまでスリープする．

3.2 拡張 GIOP パケット

拡張 RT コンポーネントの優先度キューの優先度は周期が短いほど高くなることを述べた．しかしながら，従来の RT コンポーネントは，周期に関する情報を送信できないので，リアルタイム性を保証できない．そこで，我々は CORBA の GIOP パケットに周期情報等を追加した．図 4 に拡張 GIOP パケットを

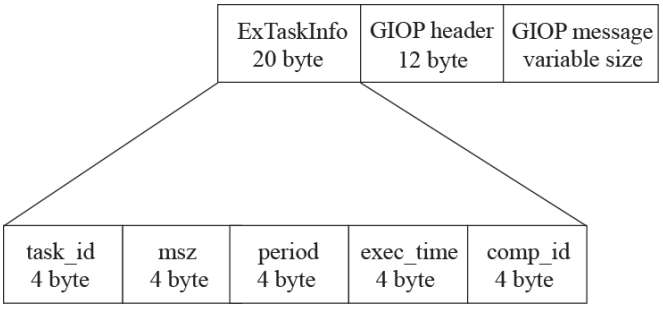

Fig. 4 Extended GIOP packet

示す．拡張 GIOP パケットは従来の GIOP パケットにリアルタイム性を保証するための情報 ExTaskInfo を追加する．ExTaskInfo にある情報として，task_id はタスクの ID，msz はパケットのサイズ，period はタスクの周期，exec_time はタスクの最悪実行時間，comp_id は拡張コンポーネントの ID を示す．ExTaskInfo により，拡張 RT コンポーネントは周期に関する情報を用いてパケットを優先度によりスケジューリング可能になる．

```
1.  recv_packet(buffer);
2.  repeat
3.  task_info = get_next_task_info(buffer + offset);
4.  msz = task_info.msz;
5.  offset += 20; /* size of ExTaskInfo */
6.  packet = get_next_packet(buffer + offset, msz);
7.  enqueue_request(packet, task_info);
8.  offset = offset + msz;
9.  until end of buffer
10. req = dequeue_request();
11. execute(req);
```

Fig. 5 Pseudo code of unmarshaling extended GIOP packet

図 4 で示した拡張 GIOP パケットのアンマーシャリングを行うために，従来の GIOP パケットをアンマーシャリングするアルゴリズムを改良した．図 5 に拡張 GIOP パケットのアンマーシャリングの擬似コードを示す．コネクションスレッドが拡張 GIOP パケット群を buffer に受信した場合(1 行目)，コネクションスレッドは buffer に格納されたパケットを拡張 GIOP パケットにアンマーシャリングし(3～6 行目)，優先度キューに格納する(7 行目)．拡張 GIOP パケットが全てアンマーシャリングされた場合(9 行目)，コネクションスレッドは優先度キューからリクエストを取り出し (10 行目)，リクエストを実行する(11 行目)．この処理を追加することで，拡張 RT コンポーネントは従来の GIOP パケットと拡張 GIOP パケットの両方をアンマーシャリング可能になる．

3.3 実装

拡張 RT コンポーネントは，同じまたは異なる周期タスクを管理し，ローカルスレッドはこれらのタスクを実行する．図 6 に拡張 RT コンポーネントにおける周期タスクの実行を示す．svc 関数の引数は，ExTask クラスの vector の reference であ

```
void svc(vector <ExTask>& task) {
   int i, j;
   for (i = 0; i < LCM; I += GCD) {
      for (j = 0; j < task.size(); j++) {
         if ((i % task.at(j).period) == 0) {
            execute(task.at(j).task_id);
         }
      }
      wait();
   }
}
```

Fig. 6 Execution of periodic tasks in extended RT-Component

る．ExTask クラスはタスクの周期やデッドライン等を管理するクラスである．LCM は及び GCD は，それぞれ拡張 RT コンポーネント内で管理する全タスクの周期の最大公約数及び最小公倍数を示す．svc 関数では，リリース時刻になったタスクがあるか否かをチェックし，リリース時刻の場合，execute 関数を実行する．execute 関数の引数に task_id を渡すことで，task_id の示すタスクを実行する．ローカルスレッドが，あるリリース時刻で実行すべきタスクを全て完了した場合，次のリリース時刻まで，拡張 RT コンポーネントを wait 関数によりスリープさせる．

```
void create(Exmanager* ex_mgr) {
   vector<ExTask> task;
   task.push_back(ExTask(500000, 60000, DETECT_OBJECT));
   task.push_back(ExTask(20000, 500, SELECT_ROUTE));
   task.push_back(ExTask(1000, 30, CONTROL_MOTOR));
   ex_mgr->createComponent("ExRTC", task);
}
```

Fig. 7 Example of creating extended RT-Component for wheel robot

次に，車輪ロボットを例とした拡張 RT コンポーネントの使用例を示す．拡張 RT コンポーネントは障害物検知，経路選択，モータ制御の 3 つのタスクを持つ．

図 7 に拡張 RT コンポーネントの生成例を示す．create 関数の引数は，ExManager クラスのポインタである．ExManager クラスは従来の RT ミドルウェアにある Manager クラスを継承したクラスである．ExTask クラスの引数は第 1 引数から順に周期，最悪実行時間及びタスクの種類を示す enum 変数である．周期及び最悪実行時間の単位はμs とする．Manager クラスで RT コンポーネントを生成する手順とほぼ同様の手順で，ExManager クラスの createComponent メソッドを呼び出すことで，3 つの周期タスクを持つ拡張 RT コンポーネントを生成する．

```
void execute(int task_id) {
   switch (task_id) {
   case DETECT_OBJECT:
      detect_object();
      break;
   case SELECT_ROUTE:
      select_route();
      break;
   case CONTROL_MOTOR:
      control_motor();
      break;
   }
}
```

Fig. 8 Example of executing periodic tasks for wheel robot

図 8 に車輪ロボット用の周期タスクの実行例を示す．execute 関数はローカルスレッドにより呼び出される．execute 関数の引数は，タスクの番号を示す task_id である．障害物検知，経路選択，モータ制御タスクはそれぞれの処理を行う関数である detect_object，select_route，control_motor 関数を呼ぶ．従来の RT ミドルウェアでは，3 つの RT コンポーネントを生成しなければならなかったが，リアルタイム拡張した RT ミドルウェアを用いることで，1 つの拡張 RT コンポーネントで複数の周期タスクを管理できる．

4. 評価

評価には Pentium 4 2.53 GHz，2.26 GHz の二つのマシンを用いる．メモリは両方とも 512MB である．OS は ART-Linux version 2.6.22 [9]を用いる．ART-Linux を用いることで，リアルタイム性能の高い周期タスク実行が可能になる．RT ミドルウェアは OpenRTM-aist version 0.4.1 [10]を用いる．OpenRTM-aist で利用するミドルウェアは，ACE version 5.6，omniORB version 0.4.7 で構成した．評価に用いるタスクセット数は 100 とする．タスクセット内のタスクの周期は 100ms から 1000ms まで 50ms おき，実行時間は 10ms から 30ms まで 5ms おきに一様分布で生成する．評価結果には，提案手法のリアルタイム拡張した RT ミドルウェアは proposed，従来手法の RT ミドルウェアを conventional と表記する．

評価指標はオーバーヘッドとリクエストタスクの成功率である．オーバーヘッドは，図 5 で示した拡張 GIOP パケットのアンマーシャリング処理を評価する．この拡張 GIOP パケットのアンマーシャリング処理を含めた，リクエストをワーカースレッドに割り当てる処理にかかるオーバーヘッドを測定する．ReaD-Time Stamp Counter 命令[11]を用いて，実行時間を計測する．

スケジュール成功率とは，図 9 に示すように，クライアントがリクエストを送信し，そのコールバックがデッドラインまでに返ってくるか否かを判定する．タスクセット内の全タ

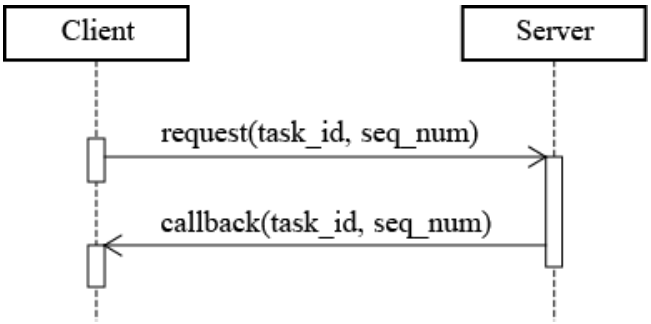

Fig. 9 Sequence diagram of client and server

スクがデッドラインまでにリクエストのコールバックが返ってきた場合，スケジュール成功と判定する．スケジュール成功率を以下に定義する．

$$Success\ Ratio = \frac{\#\ of\ successfully\ scheduled\ \ task\ sets}{\#\ of\ \ scheduled\ \ task\ sets}$$

タスクのデッドラインは周期と等しいとする．クライアントは，Pentium 4 2.53GHz，サーバは Pentium 4 2.26GHz を搭載したマシンとする．request，callback 関数の引数は図 4 で示した task_id と seq_num である．各々リクエスト τi の最悪実行時間は Ci，周期は Ti とする．システム全体の CPU 利用率 U=ΣCi/Ti と定義する．計測時間はタスクセット内の全タスクの周期の最小公倍数であるハイパーピリオドまでとする．また，全てのメッセージは oneway 属性を付加する．すなわち，メッセージは全て非同期で送信する．

4.1 オーバーヘッドの評価

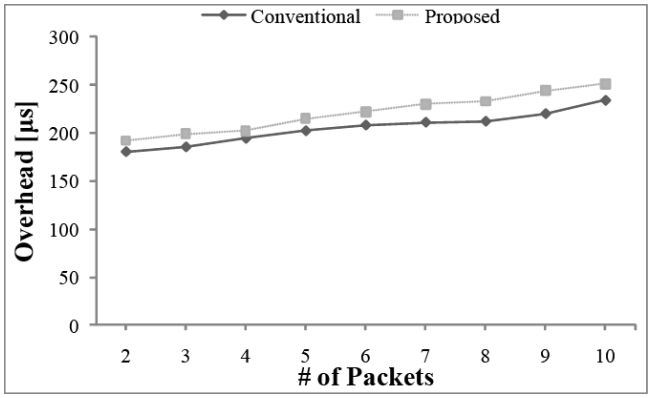

Fig. 10 Overhead of assigning requests

図 10 にリクエストを割り当てる処理のオーバーヘッドの評価結果を示す．オーバーヘッドの評価は，アンマーシャリングするパケット数毎にプロットした．リアルタイム拡張した RT ミドルウェアのオーバーヘッドは従来の RT ミドルウェアのオーバーヘッドと比較して，図 5 で示した拡張 GIOP パケットをアンマーシャリングする処理により約 5%高くなる．

4.2 スケジュール成功率の評価

図 11 に Success Ratio の評価結果を示す．Success Ratio の評価は CPU 利用率を 0.3 から 0.8 まで 0.05 おきでプロットした．CPU 利用率が約 0.4 以上になると Success Ratio が低下する理由は，パケットの送受信処理に発生するオーバーヘッドが原因であると考えられる．リアルタイム拡張した RT ミドルウェアの Success Ratio は従来の RT ミドルウェアの Success

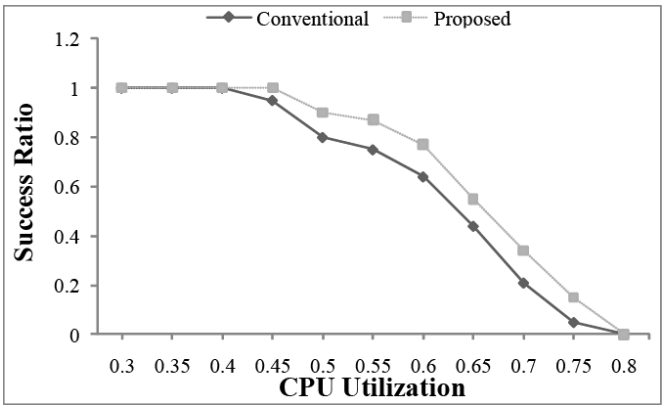

Fig. 11 Success Ratio

Ratio と比較して最大 10%高くなった．図 10 で示したリアルタイム拡張した RT ミドルウェアのオーバーヘッドが従来の RT ミドルウェアより高くなったにも関わらず，Success Ratio が高くなったので，リアルタイム性能を向上できたと言える．

5. 結論

本論文では，RT ミドルウェアをリアルタイム拡張した RT コンポーネントの性能評価を行った．評価結果により，拡張 RT コンポーネントは従来の RT コンポーネントと比較して，オーバーヘッドが約 5%高くなってしまったが，スケジュール成功率を向上させることができた．拡張 RT コンポーネントにより，RT コンポーネントを利用した開発コストの削減とリアルタイム性の向上を両立できた．

今後の課題として，拡張 RT コンポーネントで，ロボットで利用するタスクである，モータ制御，画像処理，逆運動学タスク等を実行する予定である．

謝　辞

本研究の一部は科学技術振興機構 CREST 及び文部科学省グローバル COE プログラム「環境共生・安全システムデザインの先導拠点」に依るものであることを記し，謝意を表す．

文　献

[1] F. R. T. Center. http://www.furo.org/.
[2] H. S. Ahn, Y. M. Beak, I-K. Sa, W. S. Kang, J. H. Na, and J. Y. Choi. Design of Reconfigurable Heterogeneous Modular Architecture for Service Robots. In Proceedings of the 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 1313-1318, Sep. 2008.
[3] F. Kanehiro, H. Hirukawa, and S. Kajita. OpenHRP: Open Architecture Humanoid Robotics Platform. The International Journal of Robotics Research, 23(2):155–165, 2004.
[4] N. Ando, T. Suehiro, K. Kitagaki, T. Kotoku, and W. K. Yoon. RT-Middleware: Distributed Component Middleware for RT (Robot Technology). In Proceedings of the 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 3555–3560, Aug. 2005.
[5] O. M. Group. Common Object Request Broker Architecture (CORBA) Specification, formal/08-01-04 edition, Jan. 2008.
[6] D. C. Schmidt. The ADAPTIVE Communication Environment: An Object-Oriented Network Programming Toolkit for Developing Communication Software. In Proceedings of the 12th Annual Sun Users Group Conference, Dec. 1993.
[7] omniORB. http://omniorb.sourceforge.net/.
[8] C. Liu and J. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. Journal of the ACM, Vol. 20, pp. 46–61, 1973.
[9] 石綿陽一，松井俊浩，國吉康．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.
[10] OpenRTM-aist. http://www.is.aist.go.jp/rt/OpenRTM-aist/.
[11] I. Corporation. Intel 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Nov. 2007.