

OpaqueRPCを用いたRTMとTECS間通信の実現

八幡 尚文[†] 安積 卓也[†] 西尾 信彦[†]

近年, RTミドルウェア (RTM) がロボットのプラットフォームとして注目されている. しかし, RTMはCORBAを用いているためリアルタイム性の確保ができない. そこで, RTMとTECSの通信を提案する. TECSは, 組み込み機器で用いるコンポーネントシステムであり, リアルタイム性が要求されるシステムに向いている. RTMからリアルタイム性が必要な部分を分離し, TECSで実装することによりRTMのシステムの一部をリアルタイム性の厳しいシステムとして扱うことができる. 加えて, プラグインを利用することによりRTMからTECSへ通信するためのコンポーネントを生成する. RTMとTECSの通信にはTECSのRPC機構であるOpaqueRPCを用いた. 評価では, RTMからRTMへの通信とRTMからTECSへの通信の処理時間を比較した. さらに, プラグインを用いた場合と通信コンポーネントを作成する場合のコード記述量を比較した.

Realization of Communication for RTM and TECS by OpaqueRPC

NAOFUMI YAWATA,[†] TAKUYA AZUMI[†] and NOBUHIKO NISHIO[†]

Recently, RTM (Robot Technology Middleware) is attracting attention as a component oriented platform for a robot development. However, RTM is unable to ensure the real-time processing in CORBA because CORBA manages packets by FIFO. In this paper, we propose communications for RTM and TECS in an effort to real-time processing. TECS is a component system for embedded systems. TECS is suitable for real-time systems. RTM is seceded apart of real-time processing. We seceded the parts necessary for real-time processing from RTM. We meet the demand of real-time processing by TECS. In addition, we can be possible to generate components to communicate from RTM to TECS by using a plug-in. Thus, labor will be reduced. OpaqueRPC is RPC mechanism by TECS to communicate for RTM and TECS. In the evaluation, we compared the processing time of the communication from RTM to RTM and RTM to TECS. And the amount of codes make a comparison between a communication component and used the plug-in.

1. はじめに

近年, 人間の代わりに掃除や補助を行う知能ロボットへの期待が高まってきている¹⁾. 知能ロボットは人と同一の空間で作業することが多く, 移動時に人と接触するなどの事故が発生する可能性がある. そのため, 停止などの動作を決められた時間までに終了する必要があり, リアルタイム性の確保が求められる. ネットワークを介して複数の知能ロボットや組み込み機器を分散制御し, 生活支援を行うための知的空間の開発が進められている²⁾. 一方で, 従来のロボット開発は異なるアーキテクチャ上で行われていたため, ロボット機能の再利用性に乏しかった. これにより, ソフトウェアを機能ごとに部品として分割し, 必要に応じて組み合わせることができるコンポーネント指向開発が

注目を集めている.

ロボットのコンポーネント指向開発が行えるソフトウェア規格として, RTM (Robot Technology Middleware)³⁾が提案されている. その実装としてLinuxやWindows上で動作するOpenRTM-aist⁴⁾やOpenRTM.NET⁵⁾などが存在し, OpenRTM-aistで作成された再利用可能なコンポーネントをRTC (RTComponent)と呼ぶ. RTMを用いることにより, ロボットのコンポーネント指向開発が可能である. しかし, RTMはCORBA (Common Object Request Broker Architecture)⁶⁾を用いて分散オブジェクト機能を実現しているため, 知能ロボットで必要となるリアルタイム性を確保できない. CORBAで用いられる抽象プロトコルであるGIOP (General Inter-ORB Protocol)⁷⁾は, 管理するパケットをFIFO (First In, First Out)で処理している. さらに, CORBAは通信のオーバーヘッドが大きい. これらの理由により, 優先度が高いコンポーネントがデッドラインミスをしてしまう可能

[†] 立命館大学
Ritsumeikan University

性がある。一方で、RTM を組込み機器上で動作させることができれば、RTC 同士の連携が容易に行える。しかし、RTM は汎用 OS 上で用いる CORBA を利用しているため多くの組込み機器では利用できない。そのため、RTM と組込み機器の連携を可能にするためには、組込み機器と通信できるコンポーネントを作成する必要がある。本研究は、RTM システム (以下 RTMS) の一部をリアルタイム性の厳しいシステムとして扱え、組込み機器と通信するコンポーネントを作成できることを目的とする。

組込み機器のコンポーネント指向開発が行えるシステムとして、TECS (TOPPERS Embedded Component System)⁸⁾ が挙げられる。TECS は組込み機器に多く用いられている μ ITRON4.0 仕様⁹⁾ に準拠したリアルタイム OS である TOPPERS/ASP カーネルなどを用いている。センサ処理やアクチュエータ制御などハードリアルタイムシステムの部分に TECS を用いて、画像処理などソフトリアルタイムシステムの部分に RTM を用いることで、RTM システムの一部をリアルタイム性の厳しいシステムとして扱える。本論文は、全 7 章で構成される。2 章では関連技術について述べる。3 章では関連研究を紹介する。4 章では RTM と TECS 間通信手法を説明し。5 章で実装について述べる。6 章では評価について述べ、最後に 7 章でまとめる。

2. 関連技術

本章では、本研究で用いる RTC と TECS について説明する。

2.1 RTC

RTC は RTM におけるコンポーネントである。RTC は、様々な粒度でモジュール化が可能であり、多様なプログラミング言語や OS 上で動作できる分散コンポーネント型のフレームワークを提供している。図 1 に RTC のアーキテクチャを示す。RTC は、連続的なデータの送受信を行うデータポート、CORBA の IDL (Interface Definition Language) で定義したインタフェースを用いて通信を行うサービスポート、コンポーネントの状態を示すアクティビティ、制御パラメータを変化させるコンフィギュレーションがある。

2.2 TECS

組込みコンポーネントシステムである TECS は、TOPPERS (Toyohashi OPen Platform for Embedded Real-time Systems) プロジェクトのワーキンググループにおいて仕様の策定が進められている。TECS は既存の多くのコンポーネントシステムとは異なり、

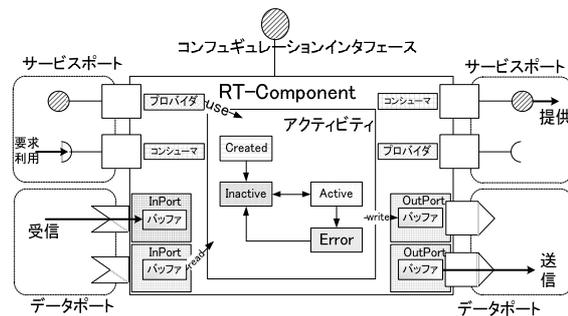


図 1 RTC アーキテクチャ
Fig. 1 RTC Architecture

コンポーネントの静的なインスタンス化・結合や、コンポーネント呼出しコードの最適化と相性が良い C 言語が用いられている。

2.2.1 コンポーネントモデル

TECS のコンポーネントモデルで記述されるセル、呼び口・受け口、シグニチャについて説明する。

- セル

セルとは、インスタンス化されたコンポーネントである。セルは、自身の機能を提供するインタフェースである呼び口、セルの定数を表す属性、セルの内部状態を表す変数で構成される。

- 呼び口・受け口

呼び口とは、他のセルの機能を利用するための関数呼出しの集合を示し、接頭辞に 'c' を用いる。受け口とは、セルの機能を提供する関数の集合を示し、接頭辞に 'e' を用いる。1 つのセルは、複数の呼び口や受け口を持つことができる。

- シグニチャ

シグニチャとは、受け口と呼び口の型を示した関数の集合である。

セルの呼び口は、同一シグニチャを持つほかのセルの受け口と結合できる。これにより、前者のセルから後者のセル関数群を呼び出すことが可能になる。図 2 にクライアントがサーバを利用しているコンポーネントモデルを示す。図 2 は、セル Server の呼び口 cServer とセル Server の受け口 eServer を結合したことを表しており、クライアントセルはサーバセルの関数を呼び出せる。呼び口名の頭には 'c' を記述し、受け口名の頭には 'e' を記述する。sServer はコンポーネント間のシグニチャ名を示す。

3. 関連研究

本章では、RTM を拡張しリアルタイム性を向上さ

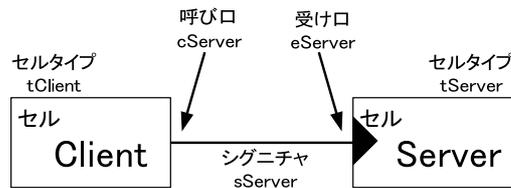


図 2 コンポーネントモデル
Fig. 2 Basic component model

せた研究や組込み機器との連携を行った研究について紹介し、問題点を挙げる。

3.1 RTM のリアルタイム性の向上

千代ら¹⁰⁾は、RTM のコンポーネント間通信で用いる GIOP パケットに周期情報を加えた拡張コンポーネントアーキテクチャを提案した。このアーキテクチャは、コンポーネント間通信はグローバル変数を用いることでリアルタイム性の向上を行った。このアーキテクチャを利用することで、オーバーヘッドは増加したが、高確率でデッドラインを満たすことが確認できた。

この研究は、RTM にリアルタイム性の向上はできているが、RTC を拡張しているため基本的なコンポーネントの作成方法が異なる。さらに GIOP の拡張を行うと GIOP の処理部分を書き換えるために専門的な知識が必要である。加えて、グローバル変数を用いるため分散環境では利用できない。

3.2 RTM と連携可能な組込み機器

安藤ら¹¹⁾は小型デバイスと通信するブリッジコンポーネント (RTC-Lite) を作成することで、小型デバイスに RTC を搭載した。これにより、RTC と小型デバイスが通信できる。

池添ら¹²⁾はリアルタイム OS である VxWorks 上で RTM を動作させる研究を行った。現状の CORBA は組込み機器に適さないため、軽量の CORBA を開発し、VxWorks で利用できるライブラリを作成した。その結果 VxWorks を用いた組込み機器上で RTM の動作を可能にした。

これらの研究では、RTC が組込み機器で動作させているが、リアルタイム性の向上については言及されていない。

3.3 組込み機器上で動作する RTM

池添ら¹³⁾は RTM を組込み機器上で動作させ、リアルタイム性を向上するための研究を行った。組込み機器の OS には VxWorks を利用し、組込み機器で RTM を利用するためのアーキテクチャの提案を行った。

この研究は、RTM を組込み機器上で動作させているが、設計までしか記述されておらずリアルタイム性に関する評価の記述がない。

4. RTM と TECS 間通信手法

RTM の問題点を以下にまとめる。

- (1) RTM はリアルタイム性の確保が難しい。
- (2) RTM を利用するには CORBA が必要である。そのため、組込み機器では利用できない。

本章では、上記の問題を解決するための要件を述べ、要件を満たすためのアプローチについて述べる。

4.1 要件

3章で挙げた問題点より、本研究の要件を記述する。

- (1) RTMS のリアルタイム性向上

知能ロボットは人と同一の空間で作業することが多く、移動時に人と接触するなどの事故が発生する可能性がある。そのため、停止などの動作を決められた時間内に終了する必要がある。

- (2) 組込み機器と連携する RTC の自動生成

現状、組込み機器と連携を行うためには、組込み機器と通信を行える RTC を作成する必要がある。多くの研究では、利用するために CORBA を変更している場合が多い。そのため、組込み機器を利用する RTC を自動作成できることが望ましい。

4.2 アプローチ

本研究では、RTMS にリアルタイム性を取り入れるために、組込みシステムのコンポーネントシステムである TECS を利用する。TECS はリアルタイム性が要求されるシステムに適しているため要件 1 を満たす。さらに、TECS のコンポーネントを作成する際に用いる TECS ジェネレータにプラグインを追加することで、RTM - TECS 間通信を行うコンポーネントを作成するコードを生成可能である。これにより要件 2 を満たす。

4.3 TECS の RPC 機構

TECS では、RPC を行うコンポーネント群として TransparentRPC¹⁴⁾ と OpaqueRPC¹⁵⁾ の 2 つが存在する。TransparentRPC はマルチプロセッサなどメモリを共有している環境で用い、OpaqueRPC は分散環境で用いる。本研究では、RTM と TECS の通信は分散環境を想定しており、OpaqueRPC を RTM と TECS の通信に用いる。

4.3.1 OpaqueRPC

本項では、はじめに OpaqueRPC の構造について述べ、OpaqueRPC を利用した際の RPC の流れを説明する。RTM には OpaqueRPC のクライアント側のコンポーネントと同様の動作を行うコンポーネントを作成する。OpaqueRPC の構成を図 3 に示し、各コン

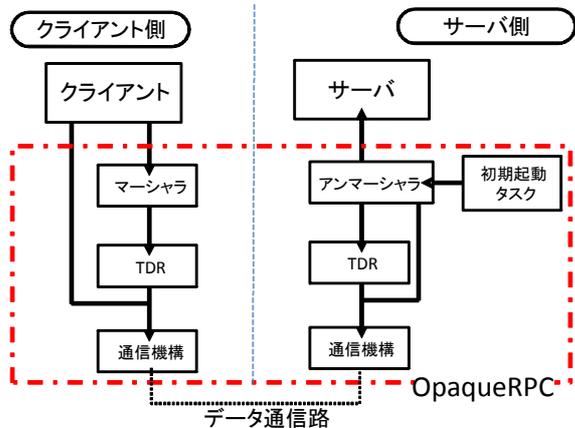


図 3 OpaqueRPC の構成
Fig. 3 Structure of OpaqueRPC

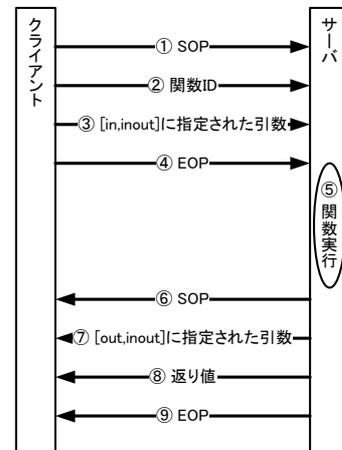


図 4 RPC の手順
Fig. 4 Communication steps of RPC

ポーネントの説明を以下に示す。

(1) サーバ

サーバは、クライアントより呼びだされた関数を実行する。引数があればそれを用いて実行し、戻り値があればマーシャラを通じてクライアントに送信する。

(2) クライアント

クライアントから、サーバ内に存在する関数を呼び出すことで指定した関数の実行が可能になる。この際に引数があれば引数を指定し、戻り値がある場合は関数の戻り値の型に合わせて記述する必要がある。

(3) マーシャラ

マーシャラは、クライアントより受け取ったデータを通信機構で通信を行えるデータ形式にシリアライズする。さらに、サーバから送られてきたデータをクライアントが処理できる形式にデシリアライズしクライアントへ伝える。

(4) アンマーシャラ

アンマーシャラは、通信機構より送られてきたデータをサーバプログラムで処理できる形式にデシリアライズし、関数 ID によりサーバの ID に対応する関数を呼び出す。その後、実行結果をシリアライズし、通信機構に送る。

(5) TDR

TDR (TECS Data Representation layer) は、RPC メッセージの判定やバイトオーダ、型変換を行う。

(6) 通信機構

通信機構は、クライアント - サーバ間のデー

タ転送に関わる処理を行う。通信路のオープンとクローズ、データの送信と受信の機能を持つ。現状、通信機構はシリアル通信を用いる方法と TCP/IP 通信を用いる方法の 2 種類が存在する。

(7) 初期起動タスク

初期起動タスクは、クライアントと接続が可能な状態の間のみアンマーシャラを起動させるタスクである。

4.3.2 RPC の手順

OpaqueRPC を利用した RPC について、図 4 に手順を示し説明する。

- 1 クライアントが SOP (Start Of Packet) を送信し、RPC のメッセージの開始を伝える。
- 2 クライアントがサーバで実行する関数の ID を送信する。
- 3 クライアントが方向指定子 [in, inout] に指定された引数を送信する。
- 4 クライアントが EOP (End Of Packet) を送信し、RPC のメッセージの終了を伝える。
- 5 サーバが関数を実行する。
- 6 サーバが SOP を送信する。
- 7 サーバが方向指定子 [out, inout] に指定された引数を送信する。
- 8 サーバに戻り値があれば送信する。
- 9 サーバが EOP を送信する。

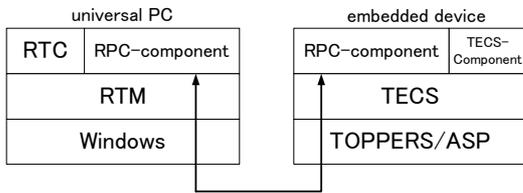


図 5 システム構成
Fig. 5 Communication system between RTM and TECS

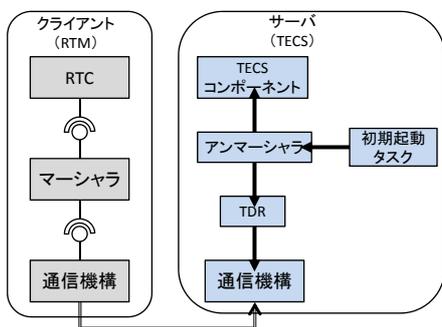


図 6 本システムのコンポーネント
Fig. 6 Proposed component system

5. 実装

本研究のシステム構成を図 5 に示す。本システムは RTM を PC 上で動作させ、TECS を組込み機器上で動作させる。RTM には Windows を用いて、TECS には μ ITRON4.0 仕様ベースのリアルタイム OS である TOPPERS/ASP を用いる。現状、クライアント-サーバ間の通信は 1 対 1 のみを対象としている。今回、RTM-TECS 間の RPC には 4.3.1 項で説明した OpaqueRPC のクライアント側にあるコンポーネントと同じ動作をする RTC を作成することによって TECS と通信する。図 5 中の RPC component は RTM と TECS で用いる RPC のコンポーネント群を示す。

図 6 に RTM と TECS のコンポーネント図を示す。各 RTC 間はサービスポートを用いて通信を行う。図 6 の RTM 側に TDR はないが、通信機構で TDR の機能を実現している。RTM と TECS では、変数の型名が違うので TECS の型と RTM の型を対応付ける必要がある。TECS ジェネレータから RTC 用のコードに変換する際は表 1 を用いて変換を行っている。

5.1 RPC プラグイン

RPC プラグインは TECS ジェネレータで利用する

表 1 RTM と TECS の型対応

Table 1 Corresponding data types between RTM and TECS

型	TECS	RTM
文字型	char_t	char
	uchar_t	char
整数型	int8_t	octet
	uint8_t	octet
	int16_t	short
	uint16_t	unsigned short
	int32_t	long
	uint32_t	unsigned long
	int64_t	long long
uint64_t	unsigned long long	
浮動小数点型	float32_t	float
	double64_t	double
論理型	bool_t	boolean
文字列型	string	string
配列型	size_t	any

ことにより、RTC コードを生成するプラグインである。RPC プラグインが生成する一覧を図 7 に示す。RPC プラグインが生成するものはマーシャラコード、通信機構コード、bat ファイル、IDL ファイル、XML ファイルの 5 種類である。以下に生成される 5 種類について説明する。

5.1.1 マーシャラコード

マーシャラは 4.3.2 項で示した順番で RPC メッセージの送受信を行う必要がある。生成されたマーシャラコンポーネントは 4.3.2 項で示した順番で行うことが可能であり、OpaqueRPC と同じ動作を行う。

5.1.2 通信機構コード

通信機構は、OpaqueRPC の TDR と通信機構と同じ動作を行う。さらに、通信機構は RTC より提供されているコンフィギュレーションを用いており、一括して変数の値を管理できる機能である。これを用いることで、実行ファイル生成後に利用する通信方法や値の変更が可能である。1 行目は、'mode1' または、'mode2' を指定する。'mode1' ではシリアル通信、'mode2' では TCP/IP 通信が利用できる。通信機構コンポーネントが持つコンフィギュレーション変数は type, ipaddr, port, baudrate, tap の 5 つがあり以下に説明を示す。

- type
 - シリアル通信: シリアル通信であることを示す 'SERIA' が記述されており、変更する必要はない。
 - TCP/IP 通信: TCP/IP 通信であることを示す 'TCP' が記述されており、変更する必要はない。
- ipaddr

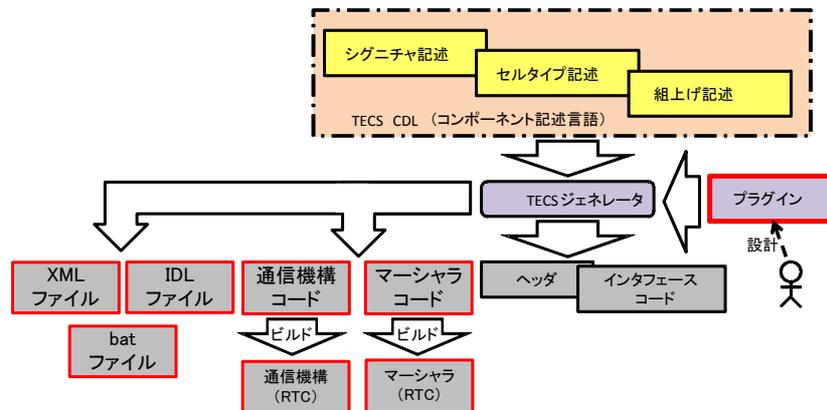


図 7 RPC プラグインによる生成
Fig. 7 The RPC plug-in generated components by RPC plug-in

- シリアル通信：利用しない。
- TCP/IP 通信：接続先の IP アドレスを指定する。
- port
 - シリアル通信：接続する COM ポートを指定する。
 - TCP/IP 通信：利用するポート番号を指定する。
- baudrate
 - シリアル通信：利用する機器に合わせたボーレート指定する。
 - TCP/IP 通信：利用しない。
- tap
 - シリアル通信：利用しない。
 - TCP/IP 通信：利用する NIC や TAP を指定する。

通信機構コンポーネントもマーシャラコンポーネントと同じく、RTC 利用者がビルドする必要があるが、一度コンポーネントを作成するとコンフィギュレーション変数の値を変更することで、汎用的に利用できる。

5.1.3 bat ファイル

bat ファイルは、マーシャラコードと通信機構コードをビルドするためのファイルである。

5.1.4 XML ファイル・IDL ファイル

クライアントコンポーネントの作成には、RTC のテンプレートを生成する開発ツールである RTCBuilder¹⁶⁾ で用いる。この際にプロファイルを参照することができ、参照ファイルに生成された XML ファイルを用いることでプロファイルのインポートができる。これにより、サーバポートが 1 つある RTC が生成できるの

```

シグニチャ
signature sServer{
    ER func1([in] int32_t indata);
    ER func2([out] char_t *outdata);
    ER func3([in] int64_t indata,
             [inout] uint32_t *iodata);
};

インタフェース
interface Server{
    long func1(in long indata);
    long func2(out char outdata);
    long func3(in long long indata,
              inout unsigned long iodata);
}
    
```

図 8 シグニチャとインタフェース
Fig. 8 Sample of signature and interface

で、サービスポートの参照ファイルに生成された IDL ファイルを用いることでマーシャラのサービスポートと通信可能なサービスポートを持った RTC が生成できる。

IDL ファイルはサーバのシグニチャを表 1 に合わせて変換してある。図 8 に変換例を示す。変換するためには、シグニチャ名、関数の戻り値の型、関数名、引数の方向指定子、引数の型、引数名をインタフェースで用いられる記述に変換する。

6. 評価

本章では、RTM-RTM 間と RTM-TECS 間のクロック・サイクル数 (以下サイクル数) を比較し両者のリアルタイム性を評価する。続いて、TECS ジェネレー

表 2 評価環境
Table 2 Evaluation environment

	RTM(Server)	RTM(Client)	TECS
CPU	2.26GHz	2.53GHz	16.8MHz
OS	Windows Vista	Windows 7	TOPPERS/ASP

タから RTC 用のコードを出力させるために必要な記述量と自動生成を利用せず作成した RTC 側のコードの記述量を比較する。

6.1 リアルタイム性

評価環境を表 2 に示し、評価構成を図 9 に示す。図 9-(a) は RTM-TECS 間通信を示し、図 9-(b) は RTM-RTM 間通信を示す。今回、TECS の評価はシミュレータ上で行うが、実機上の 1 秒とシミュレータ上の 1 秒は同じではない。シミュレータ上ではサイクル数を観測可能である。そこで、実機で得られた実行時間をサイクル数に変換することで、シミュレータと実機の違いを考慮せず比較できる。評価対象は図 9 の赤線内との青線内に入ったときから出るまで経過したサイクル数とする。実行関数は処理を行わないとし、関数を 100 回呼び出した際の平均サイクル数を比較する。本来ならばどちらも同じ環境で、図 9 の緑線内を計測する必要がある。しかし、現状同じ機器で動作させることができない。そのため今回の評価は図 9 の赤線内と青線内のサイクル数を比較する。

図 10 より、RTM のサイクル数はほとんどにおいて TECS より多かった。さらに、RTM は TECS と異なりサイクル数にばらつきがあった。RTM のサイクル数平均は 24,104,075 回であり、TECS のサイクル数平均は 4,509,746 回であった。これにより、TECS を用いることで RTM のサイクル数の約 1/5 ほどで動作可能だったことを示す。さらに、TECS は RTM と比べてサイクル数のばらつきが少なかった。ばらつきが少ないことは最悪実行時間の予想が容易であり、リアルタイム性が向上したと言える。

6.2 記述量

本研究では、TECS ジェネレータを通して必要な RTC を作成するためのコードはサーバの関数に合わせて柔軟に生成される。その際に bat ファイルが生成されるので、これを実行するだけで TECS コンポーネントとの連携を行える RTC を生成できる。本来ならば、マーシャラコンポーネントと通信機構コンポーネントを作成するために、OpaqueRPC のクライアント側のコンポーネントについて熟知する必要がある。

今回 TECS と通信するマーシャラコンポーネントと通信機構コンポーネントを作成した際に必要な記述量と TECS ジェネレータにより生成するために必要な

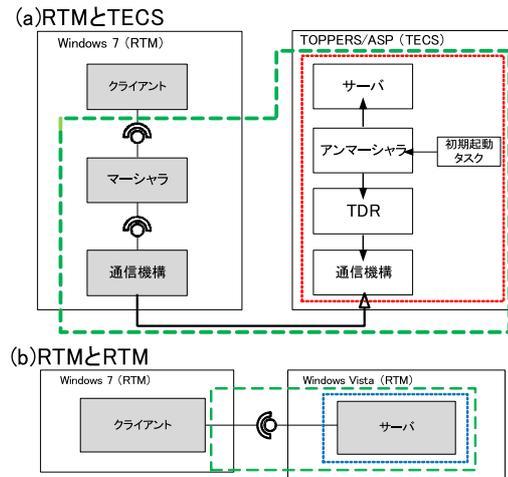


図 9 評価構成
Fig. 9 Structure of evaluation

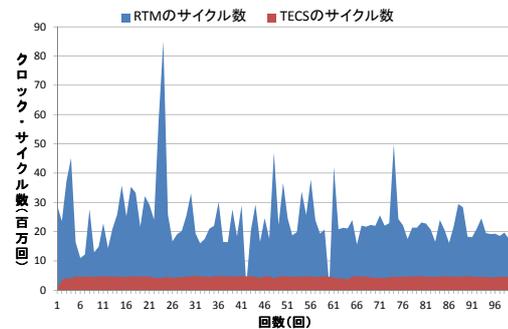


図 10 クロック・サイクル数比較
Fig. 10 Comparison of clock cycle count

記述量の比較を表 3 に示す。今回 RTCBuilder によって出力されるコード、コメントや空行は比較対象外とする。表 3 に示す通り本研究では、TECS のサーバコンポーネントの CDL ファイルに 1 行加えるだけで、893 行以上のコードを記述する必要がない。TECS コンポーネントと通信可能なコンポーネントコードを出力可能になった。 α と β は関数や引数が増えるに従って記述量が増えるため、マーシャラコードと IDL は関数が 1 つで引数がない状態の記述量のみを記載してある。結果、TECS と通信するコンポーネントを作成した際は 893 行以上も記述する必要がある。しかし、TECS ジェネレータで生成するためには「[through(RTCPlug-in,"")]」と 1 行記述を追加すればよい。

7. ま と め

本研究では、今後発展すると考えられる知能ロボットのリアルタイム性を向上するために、組み込み向けコンポーネントシステムである TECS と RTM の連携

表 3 記述量の比較

Table 3 Comparison of the amount of lines in description

	従来 (行)	本手法 (行)
マージャラコード	$191+\alpha$	0
通信機構コード	670	0
IDL	$32+\beta$	0
CDL	0	1
合計	$893+\alpha+\beta$	1

を提案した。はじめに、RTM の現状と問題点について述べた。そして、RTM の問題点を解決するための要件と、要件を満たすために RTM と TECS との連携を提案した。TECS と連携するための RPC 機構を挙げ、それを元に TECS ジェネレータが RTC コードを生成可能にするために新たにプラグインを作成した。最後に、RTC 用コードを生成するために必要な記述量と自動生成を利用しない際の記述量を比較した。

RTM-RTM 間通信と RTM-TECS 間通信のサイクル数を比較した結果より、TECS と連携すると平均サイクル数が約 1/5 回になったことを確認できた。TECS ではサイクル数のばらつきが無かったため、最悪実行時間を推測しやすいことが分かった。RTM のサイクル数にばらつきがある原因として、バックグラウンド等で動作しているアプリケーションが考えられる。RTM は組込み機器と違い他のタスクが多く動作している、これが原因であり、サイクル数にばらつきがあると考えられる。しかし、今回実機とシミュレータだったため、同じ環境で評価を取り直す必要がある。さらに、組込み機器と連携を行うために RTC 利用者が TECS と通信するためのコンポーネントを作成する必要が無く、1 行のみの記述で組込み機器と通信できる RTC を作成できた。これにより、本研究ではリアルタイム性を向上でき、RTM と TECS が容易に連携可能になった。

参 考 文 献

- 1) 産業技術総合研究所: ROBOSSA, <http://robossa.org/>.
- 2) 総務省: ユビキタス・プラットフォーム技術の研究開発, 技術報告 (2011).
- 3) 独立行政法人産業技術総合研究所知能システム研究部門: RT システム仕様記述方式 (2008).
- 4) Noriaki ANDO, Takashi SUEHIRO, Kosei KITAGAKI, Tetsuo KOTOKU, and Woo-Keun Yoon: RT-Middleware: Distributed Component Middleware for RT (Robot Technology), *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2005)*, pp. pp.3555–3560 (2005.08). Ed-

monton, Canada.

- 5) 池添明宏, 中本啓之, 長瀬雅之: .NET Framework を利用した RT ミドルウェア:OpenRTM.NET, 第 7 回計測自動制御学会 (SICE) システムインテグレーション部門講演, pp. 3B2–3 (2006).
- 6) OMG Formal Specifications: CORBA, <http://www.omg.org/spec/CORBA/>.
- 7) Kamel, M. and Leue, S.: Validation of the General Inter-ORB Protocol (GIOP) Using the Spin Model-Checker, *In Software Tools for Technology Transfer*, Springer-Verlag, pp.394–409 (1998).
- 8) 安積卓也, 山本将也, 小南靖雄, 高木信尚, 鶴飼敬幸, 大山博司, 高田広章: 組込みシステムに適したコンポーネントシステムの実現と評価, *コンピュータソフトウェア*, Vol. 26, No. 4, pp. 39–55 (2009).
- 9) Takada, H., Sakamura, K.: μ ITRON for smallscale embedded systems, *IEEE Micro*, Vol.15, No. 6, pp. 46–54 (1995).
- 10) 千代浩之, 山崎信行: RT ミドルウェアのリアルタイム拡張, *ロボティクス・メカトロニクス講演会講演概要集*, Vol. 2010, pp. 2P1–A19(1)–2P1–A19(4) (2010).
- 11) Noriaki Ando, Kenichi Ohara, Takashi Suzuki, and Kohtaro Ohba: RTC-Lite: Lightweight RT-Component for Distributed Embedded Systems, *SICE Journal of Control, Measurement, and System Integration*, Vol. 2, No. 6, pp. 328–333 (2009).
- 12) 池添明宏, 中本啓之, 長瀬雅之: OpenRT Platform/RT ミドルウェアの VxWorks 対応, *ロボティクス・メカトロニクス講演会講演概要集*, pp. 2A1–F19(1)–2A1–F19(3) (2010).
- 13) 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 (RT ミドルウェアとオープンシステム), *ロボティクス・メカトロニクス講演会講演概要集*, Vol.2008, pp. 1P1–E05(1)–1P1–E05(4) (2008-06-06).
- 14) Takuya Azumi, Hiroshi Oyama, and Hiroaki Takada: A memory allocator for efficient task communications by using rpc channels in an embedded component system, *In Proceedings of the Ninth IASTED International Conference on Software Engineering and Applications, Orlando, Florida, USA*, pp. 204–209 (2008).
- 15) 山口英之, 安積卓也, 石川拓也, 小南靖雄, 鶴飼敬幸, 高木信尚, 大山博司, 高田広章: 組込みコンポーネントシステム TECS と TINET を用いた RPC 機構, 第 12 回組込みシステム技術に関するサマワーショップ, pp. 81–82 (2010).
- 16) 独立行政法人産業技術総合研究所: RTCBuilder, <http://openrtm.org/openrtm/ja/node/1176>.