RSJ 2003

THE ROBOTICS SOCIETY OF JAPAN

日本ロボット学会第21回学術講演会

講演概要集

2003/9.20-22

会場｜東京工業大学

主催（社）日本ロボット学会

RT コンポーネントの実装例

– RT ミドルウェアの基本機能に関する研究開発 (その 2) –

○末廣尚士 北垣高成 神徳徹雄 尹祐根 安藤慶昭
(独) 産業技術総合研究所 知能システム研究部門

Implementation Framework of RT Component

*Takashi SUEHIRO, Kosei KITAGAKI, Tetsuo KOTOKU,
Woo-keun YOON and Noriaki ANDO
National Institute of Advanced Industrial Science and Technology(AIST)

Abstract—

A new project of the METI was begun for three years from 2002 fiscal year. The purpose is a basic research and development which enables us to construct a robotic system with a new function easily by modularizing elements as software components and combining them freely. We have proposed the RT software component as a basic framework of the modularization of of robotic functional elements. This paper describes an example of basic functions of the RT software component implemented on the omniORB(one of a CORBA implementation).

Key Words: RT(Robot Technology), Middleware, RT software component

1. はじめに

経済産業省のプロジェクト「ロボット機能発現のために必要な要素技術開発」が平成 14 年度より 3 年計画で開始された [1, 2, 3]。

本プロジェクトの目的は、ロボットシステムを構成する要素をソフトウェア的にモジュール化し、それを部品として自由に組み合わせることにより、新しい機能を持ったロボットシステムを容易に構築できるようにするための基礎技術を確立することにある。このようにして、新しい機能を持ったロボットシステムを自由に作れるようになれば、現在は対応しきれていないユーザごとの個別にニーズに答えることができ、従来は、主に製造業分野に限られていたロボット産業が非製造業分野へと拡大して行くことが期待されている [4]。

これを実現するために、我々は、ロボットの要素を部品として再利用できる形のモジュール化の単位として RT ソフトウェアコンポーネント (簡単のため、以後、本稿では RT コンポーネントまたは単にコンポーネントと呼ぶ) を提案している。本稿では、その RT コンポーネントの基本機能の実装例を CORBA の実装の 1 つである omniORB を使用して作成したので紹介する。

2. RT コンポーネントを構成する CORBA オブジェクト

ここでは、コンポーネント自身がアクティブに動作し、相互に通信しながらシステムの機能を実現するタイプのモジュール化を実現する。1 つの RT コンポーネントは、以下の 3 種類の CORBA オブジェクトから構成されることになる。

- RtComponent オブジェクト

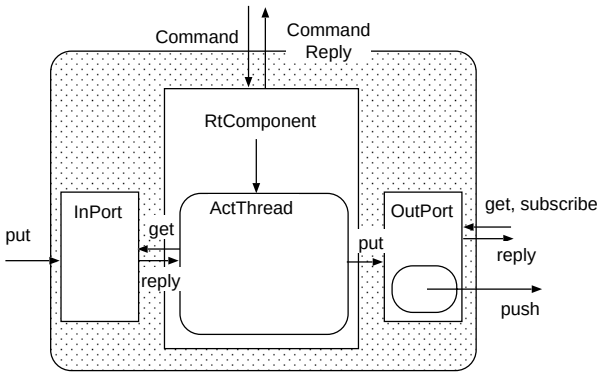

Fig.1 RT コンポーネント

RT コンポーネントの本体。コンポーネントのコマンドインタフェースはここで定義される。RtComponent では、様々なコンポーネントに共通なものを定義する。個々のカテゴリーのコンポーネントは RtComponent を継承して作られる。ただし、複雑な多重継承をさけるために実装の方は RtComponent の実装を継承せずに個別にする方が良い。実装部はアクティビティ部を別 thread として持つ。入力ポート、出力ポートは、各々InPort, OutPort の”実装”を内部で生成することで実現する。それらもまた別 thread で実行される。

- InPort オブジェクト
RT コンポーネントの入力ポート。IDL で公開された put 操作により外部からデータを受け取る。受け取った後の処理はコンポーネントごとに異な

るが数種類のパターンに別けられると考えている。これらの違いは、それぞれの”実装”を用意することで吸収する。

- OutPort オブジェクト
  RT コンポーネントの出力ポート。IDL で公開された get 操作により外部からの pull 型アクセスを受け付ける。また、InPort の object reference を引数にして subscribe することにより、指定された InPort に対してアクティビティ部でのデータ生成に応じて push 型のデータ出力を行う。

これらの IDL 定義を以下に示す。引用関係があるので本文説明とは宣言の順番が変更されている。入出力ポートは、any 型のデータを用いることで IDL 定義を各々 1 つですむようにした。通信や処理の効率を考えると any 型でない方が良いのだが、その場合、ポートのデータ型に応じて異なる IDL 定義が必要となる。

```
enum ReqType {once,repeated};
//
interface InPort {
  void put(in any value);
  readonly attribute string name;
  readonly attribute CORBA::TypeCode port_type;
};
//
interface OutPort{
  any get();
  boolean subscribe(in ReqType r_type, in InPort i_port);
  boolean unsubscribe(in ReqType r_type, in InPort i_port);
  readonly attribute string name;
  readonly attribute CORBA::TypeCode port_type;
};
//
typedef sequence<InPort> InPortList;
typedef sequence<OutPort> OutPortList;
//
interface RtComponent {
  void on();
  void off();
  InPortList in_ports();
  OutPortList out_ports();
};
```

3. CORBA オブジェクトの実装概要

3·1 RtComponent

- RtComponent_i
  いくつかのメンバ変数や関数が定義されてるが前章で書いた理由で実際には個々のカテゴリの実装が利用される。

3·2 InPort

InPort の実装例として以下の 2 種類のものを作った。

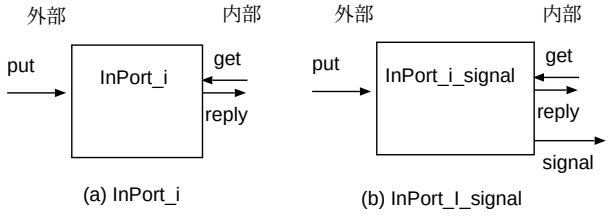

Fig.2 InPort の実装

- InPort_i
  外部からの put 操作で送られたデータを蓄えておき、必要に応じて内部からの get 関数で読み出すという単純なものである。明示的な排他制御はしていないが、単一スレッドで走らせれば問題はない。内部からは、外部からの通信とは衝突さえしなければ無関係に好きなタイミングでデータを読みにいける。外部 put、内部 get ともにデータコピーをしてすぐ抜けるだけの単純なものである。
- InPort_i_signal
  外部からの put 操作を受けたときに、条件変数を使って内部の待ちスレッドを起動できるようにしたものである。

3·3 OutPort

OutPort の実装例として以下の 3 種類のものを作った。

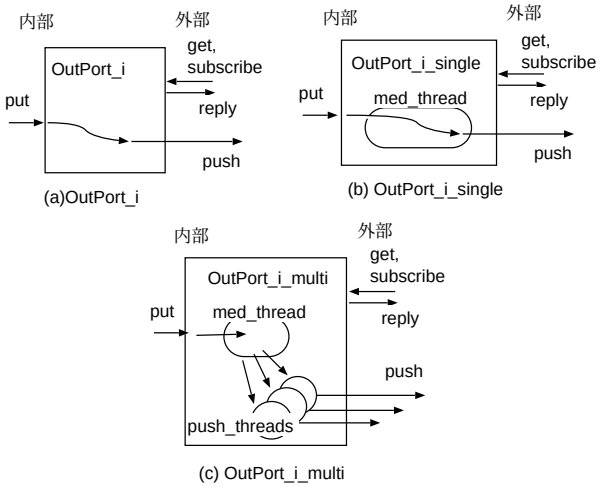

Fig.3 OutPort の実装

- OutPort_i
  内部から put 関数によりデータを受け取る。その後、あらかじめサブスクライブされている InPort へ put することにより push 型のデータ出力を行なう。また、外部からの get 操作により蓄えられているデータを読み出すことができる。単一スレッドで走らせれば排他制御の問題はない。外部 get はデータコピーのみで処理はすぐに抜ける。問題は push 型のデータ転送で、InPort への put の際に遠隔呼出しが返って来るまで待たされる。それを内部 put でやっているため、その間、内部の処理も待たされてしまう。
- OutPort_i_single
  上記の push 型転送の問題点を解決したものである。仲介の MedThread を作り push の処理をさせる。これで内部 put をデータコピーのみにできるので内部の処理を待たせなくなる。MedThread では、条件変数を使って内部 put を待ち受け、put 終了後に別スレッドで push の処理をする。別スレッドであるため遠隔呼出しで待たされても他の処理の妨げにならない。コピーされたデータへのアク

セスを mutex で排他制御しているため、分散オブジェクトとして multi thread で走らせても安全である。

- OutPort_i_multi
  複数の InPort へ push できるようにしたものである。そのため PushThread を定義して、各 InPort ごとにスレッドを割り当てている。

4. RT コンポーネントのサンプルの IDL 定義とその実装

センサ (SampleSensor) 、モータ (SampleMotor) 、コ

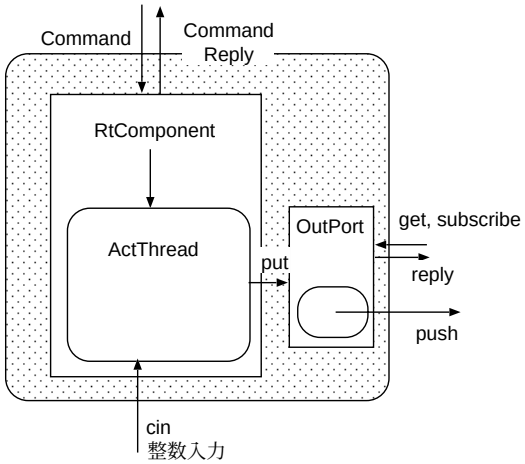

Fig.4 センサコンポーネント

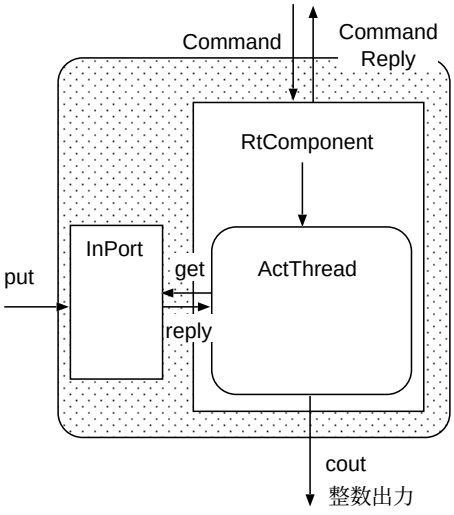

Fig.5 モータコンポーネント

ントローラ (SampleController) の 3 つの RT コンポーネントのサンプルを作った。コントローラコンポーネントは、Fig.1 の InPort を 2 つにした構造になるので図は省略する。IDL 定義は以下のようになる。RtComponent を継承するだけで新しい operation の追加はなにもしていない。

```
#include char "rt_component.idl"
interface SampleSensor : RtComponent { };
interface SampleMotor : RtComponent { };
interface SampleController : RtComponent { };
```

逆に各々の実装では RtComponent_i は継承させない。継承させると 2 つの点が問題となる。ひとつは POA_RtComponent と POA_SampleSensor などが上位で同じクラスを継承しているため、関連するメソッドの指定で混乱することがあること、もうひとつは内部で定義したスレッドのクラスを friend 宣言しても、継承元の RtComponent_i の friend とはならず、そのデータにはアクセスできないこと、である。

各々の実装ではコンポーネントの生成時に必要な入出力ポートオブジェクトも生成する。そのためコンポーネントの生成時に、サーバプログラム (後述) で起動した orb と rootPOA を引数として取る。それぞれの動作は以下のようになる。

- SampleSensor(SampleSensor_i)
  標準入力から入力された整数値を読み、OutPort_i_single を通して出力する。
- SampleMotor(SampleMotor_i)
  InPort_i に外部 put されたデータを標準出力に書き出す。
- SampleController
  2 つの InPort_i からの入力の差を OutPort_i_single を通して出力する。

各々のコンポーネントを 1 つずつ起動するためのサーバプログラムを作成した。各々の違いは、どのコンポーネントを起動するかだけで他は全く同じである。

5. RT コンポーネントを利用したサーボプログラム

ここでは、SampleController に 2 つの SampleSensor と 1 つの SampleMotor とを接続し、サーボ的な動作を実現するプログラムについて説明する。このサーボプログラムは、まず、あらかじめ起動されている RT コンポーネントに対してコマンドを出し、それぞれの入出力ポート結合する。そして各々のアクティビティを活性化することで全体としてサーボ的な機能を実現する。概略手順は以下のようになる。

- 最初に、あらかじめ (別々の window) で各々のコンポーネントを走らせておく。

```
$ ./SampleSensor sen.ref

$ ./SampleSensor ref.ref

$ ./SampleMotor mot.ref

$ ./SampleController cnt.ref
```

  ここでは各々のオブジェクトレファレンスはファイルを通して受渡すことにしている。引数は、そのファイル名である。
- さらに別の window で ServoProgram を実行する。

```
$ ./SampleServo
```

  このときサーボプログラムはすぐに抜けるが、各コンポーネントは生き続けてサーボ的な動作をする。

サーボプログラム内での処理は以下のようになっている。

1. あらかじめ起動されている RT コンポーネントの IOR ファイルを読み込み、各 RT コンポーネントへの object reference を作成する (Fig.6)。

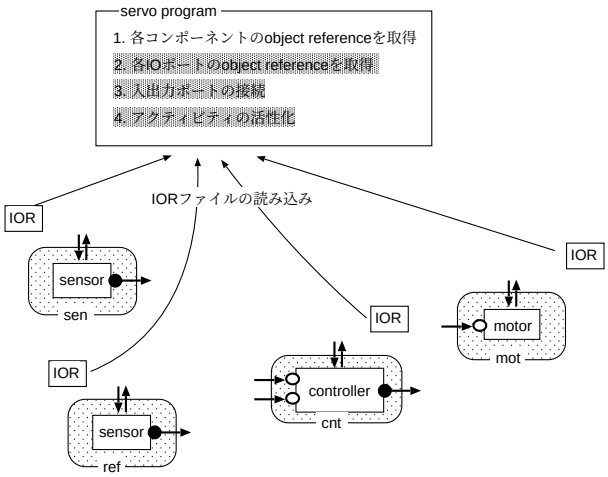

Fig.6 サーボプログラムの動作 (1)

2. 各 RT コンポーネントの入出力ポートの object reference を取得する (Fig.7)。。

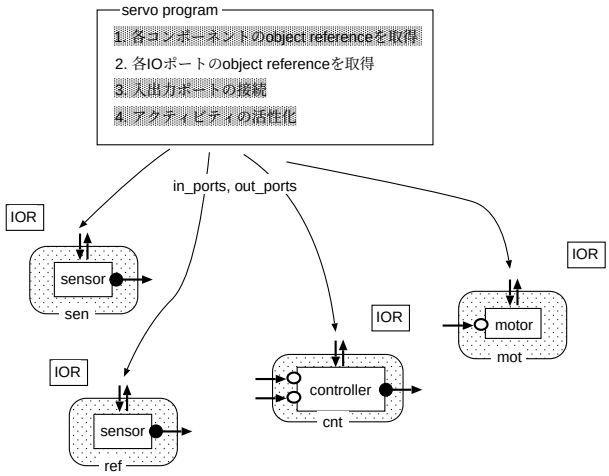

Fig.7 サーボプログラムの動作 (2)

3. OutPort への subscribe により各入出力ポートの結合を行なう (Fig.8)。
4. 各 RT コンポーネントのアクティビティをオンにする (Fig.9)。

6. おわりに

本稿で実装した RT コンポーネントは、コンポーネント自身がアクティブに動作し、相互に通信しながらシステムの機能を実現するタイプのモジュール化を実現するものである。極めて簡単な例ではあるが基本的な枠組として有用性を確認できた。一方で、RT コンポーネントの記述、オブジェクトレファレンスの引き渡し方法、RT コンポーネントの状態監視、入出力ポート以外の接続の検討などなど、数え切れないほど多くの検討課題、研究課題があることも確認できた。

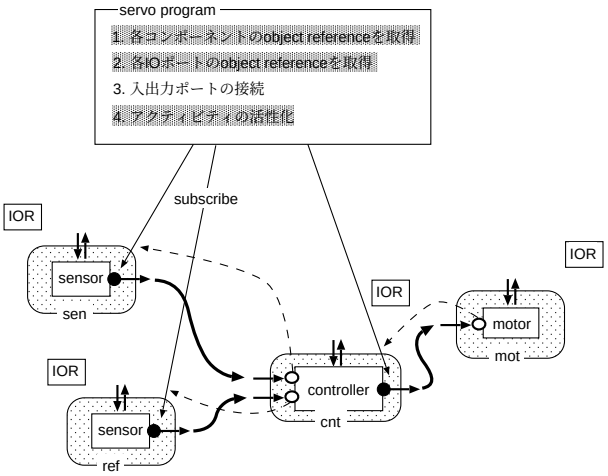

Fig.8 サーボプログラムの動作 (3)

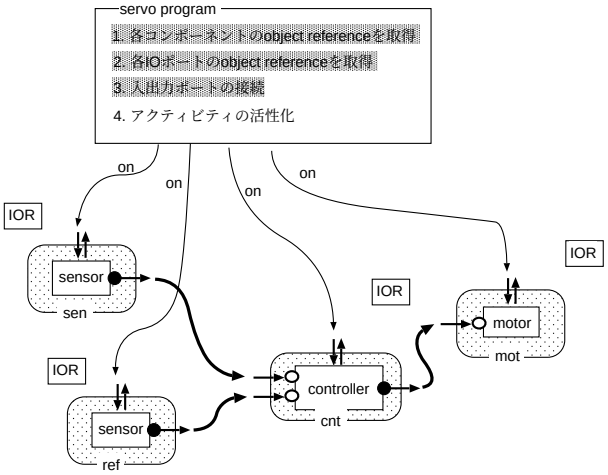

Fig.9 サーボプログラムの動作 (4)

本研究は、NEDO「ロボットの開発基盤となるソフトウェア上の基盤整備」の依託を受けて行なったものである。

参考文献

1) 北垣、末廣、神徳、平井、谷江: RT ミドルウェア技術基盤の研究開発について -ロボット機能発現のために必要な要素技術開発-、ロボティックシンポジア予稿集、pp.487-492, 2003。
2) 北垣、末廣、神徳、平井、谷江: RT ミドルウェア技術基盤の確立に向けて、ロボメカ、2003。
3) 平井、末廣、北垣、神徳、谷江: ロボットシステムのソフトウェアに基づくモジュール化に関する一考察、SICE SI 部門講演論文集、vol.2, pp.53-54, 2002。
4) 21 世紀におけるロボット社会創造のための技術戦略調査報告書、(社) 日本機械 工業連合会、(社) 日本ロボット工業会、2001。