第９回
ロボティクスシンポジア

Robotics Symposia

2004.3.8-9

RSJ JSME SICE

covers
Sensing Instrument Control Engineering
Systems Information Computer Ergonomics

【共同主催】
日本ロボット学会
日本機械学会ロボティクスメカトロニクス部門
計測自動制御学会システムインテグレーション部門

【後　　援】
琉球大学工学部情報工学科
沖縄観光コンベンションビューロー
日本旅行沖縄
琉球新報社

RT 要素のモジュール化および RT コンポーネントの実装

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

Modularization of RT Element, and RT Component Implementation

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

Abstract—

Key Words: RT(Robot Technology), software component, middleware, robot system

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 年計画で行われている [2, 3, 4]。

本プロジェクトの目的は、ロボットシステムを構成するロボット技術要素 (Robot Technology, RT[1]) をソフトウェア的にモジュール化し、これらの自由な組み合わせにより、新しい機能を実現する RT システムを容易に構築可能にする基礎技術を確立することにある。これを実現するため、筆者らはロボット要素の部品化および再利用化を促進する RT ミドルウエアを提案している。RT ミドルウエアは、既存のロボット技術を部品化し再利用を促進するための

- コンポーネントフレームワーク、
- 標準的に再利用されるソフトウエア部品群、
- ライブラリ群、
- 標準サービス群

などから構成される (図 1)。

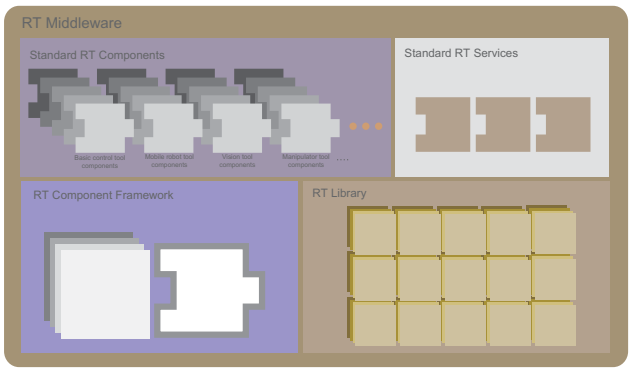

Fig.1 RT ミドルウエア

[1]RT (Robotic Technology) とは、「ロボット技術を活用した、実世界に働きかける機能を持つ知能化システム」に関する技術の総称である。移動ロボット、マニピュレータなどロボット単体のみならず、知能化空間など一見ロボットには見えないシステムも、RT の集合体とみなす [1]。

本稿では、最も基本的な要素であるコンポーネントフレームワーク (=RT コンポーネント) に関する議論を行いその基本機能を紹介する。さらに、この考えに基づいて実装されたコンポーネントフレームワークおよび、コンポーネント実装例を紹介する。

2. RT 要素のモジュール化に関する検討

RT 要素のモジュール化に対する要請に関しては文献 [2, 3, 4] において検討、提案が行なわた。これは、以下のようにまとめられる。

アクティブなモジュールを実現する機能 オブジェクト指向におけるオブジェクトは通常、メッセージに対して応答を返すパッシブな動作を基本としている。これに対してロボット要素はサーボなど、それ自身のタスクを持つだけでなく、必要なデータを自ら収集したり、イベントの発生を通知したりする必要がある。

安全性を確保するための機能 モータなどのアクチュエータを動作させる場合、動作途中に通信障害が発生した場合にも安全に動作させる工夫が必要となる。

周期動作やモジュール間の時刻同期等の時間管理機能 サーボ制御では安定した周期動作が必要である。また複数のモジュールを協調させるためには時刻同期が必要となる。センサデータの収集においても複数データ間の同時性、時間関係の把握は極めて重要なことである。

モジュール間の高速で密な連携を実現する機能 RT システムではサーボなど、家電ネットワークやプロセス制御などと比べるとモジュール間の高速で密な連携が要求される。

これら RT 要素特有の機能の実現を考慮しながら、RT 分野のアプリケーション全体に広く共通的に使われる機能を抽出して、それを整理、整備することにより RT 要素のモジュール化のフレームワークを確立する必要がある。

3. 分散オブジェクトによるモジュール化に関する検討

3·1 分散オブジェクトによるサーボの記述例

具体例として、Fig.2 のようなセンサ、モータ、サーボからなる制御システムを考える。

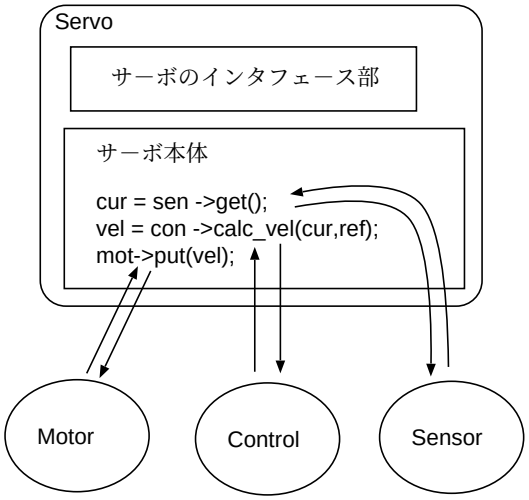

Fig.2 サーボのプログラム

これを分散のオブジェクト技術 CORBA により実装する場合を考慮すると以下の示唆が得られる。

サーボ本体の働きは IDL では記述できない

これは当り前のようだが、Sensor や Motor などの機能がほぼ IDL で記述され尽くされているように見えることとは対照的である。さらに、このサーボ本体の働きは、サーボのモジュールにとって最も重要な機能である。この例ではこの機能がプログラムの形でしか表れていないが、モジュール化された「サーボ」の利用者にとっては、最低限そこで行われる他モジュールとの入出力は IDL のようなモジュールの仕様記述として表現されていることが必要となる。

サーボ本体は、裏で常に活動し続けている

IDL は分散オブジェクトとしての静的なインタフェースを表現しているが、その裏ではサーボ本体が常に活動して内部の状態を変更している。したがって他のモジュールとデータやコマンドなどの情報をやり取りするときに、その時間的な整合性を保証することが重要となる。

センサデータはセンサへの要求の返り値として取得されている

この例では、センサデータを取得するためには、まず、サーボからセンサにデータ要求メッセージを送る。そして、その戻り値としてセンサデータを得る。これはセンサデータを得るのにネットワーク上の往復の時間が必要であることを意味する。これは高速な制御を行う場合にネックになる。また、このような常に働いているループの場合には 2 倍の通信量になるだけだが、イベントを発生するセンサの場合、ネットワーク越しに、このようなポーリングを行うのは無駄が多い。

モータへの出力要求も戻りを待つ

同様に、分散オブジェクトの呼び出しでは一般に oneway call は推奨されていないため、モータへの出力も呼び出しからの戻りを待つ必要がある。

3·2 RT 機能のモジュール化の課題

前項の考察から RT 機能のモジュール化に関して以下のような課題が抽出される。

モジュール本体の働きの定式化

ロボットの機能要素のモジュール化に関してその本体の働きの定式化は重要である。その実装の問題を除いたとしても、インタフェース部以外で発生する外部との入出力の定式化および仕様記述は行わなくてはならない。

モジュールの構造の定式化

ロボットの要素機能のモジュールの構造に関しては、最低限インタフェース部とアクティビティ部という構造があり、そのようなモジュールを簡単に作れるようなデザインパターン、支援ライブラリを作成する必要がある。

ネットワークを意識した通信の効率化

分散オブジェクトの枠組の中では、各オブジェクトはどこにあっても (同じ 1 つのプログラム場合も含めて) 扱いに差がないのが理想である。しかし、ロボットの場合、ハードウェアに直結し場所を動かせないモジュールの存在や、ネットワークの通信時間の影響が重要な場合がある。また、 リアルタイム性が要求される場合にはいろいろな手段で通信を効率化する必要がある。

4. RT ソフトウェアコンポーネント

前節で説明したように分散オブジェクトとは IDL などで記述された明示的なインタフェース (interface) を通してのみクライアントにサービスを行うサーバオブジェクトである。これはコマンドとして送られて来たメッセージに対して応答を返すというパッシブな動作の側面のみをサポートするものであり、そのオブジェクト内での処理については基本的には立ち入らない。

これに対してロボットの要素の多くはサーボなど、それ自身のタスクを持つだけでなく、必要なデータを自ら収集したり、またイベントの発生などの通知を行ったりなどの固有の処理 (アクティビティ) を持っている。その固有の処理の中では、自身のインタフェースで要求されるデータを随時書き換えたり、他の RT モジュールへクライアントとして要求を出したり、などが行われることになる。

ロボットの要素を部品として再利用できる形でモジュール化するためには、上記の機能をひとまとまりとした単位で考える必要がある。これをオブジェクトと区別して RT ソフトウェアコンポーネントと呼びロボットの要素のモジュール化の単位とする。

4·1 RT コンポーネントの動作パターンによる分類

RT コンポーネントの機能の面から分類すると、典型的には前述のセンサ、モータ、コントロール、サーボなどの分類が考えられる。しかし、ここでは、この分類の詳細には踏み込まず、より一般的な RT コンポーネントの動作パターンから大きく Fig.3 の 3 つの分類を検討した。

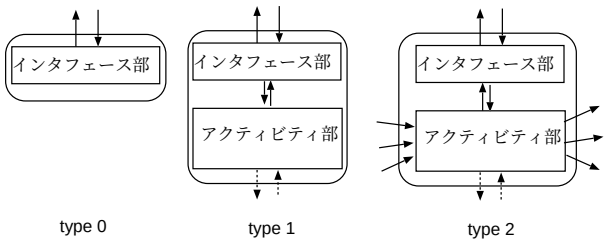

Fig.3 RT コンポーネントの典型的動作

type 0 並列動作のアクティビティ部なし。単純な分散オブジェクトとして扱える。

type 1 インタフェース部からのみ受動的に外部とのやりとりをする。固有ハードウェアとのインタフェースはあるとしても、アクティビティ部でのモジュール間通信はなくひたすら内部処理を行う。外部的には、普通の分散オブジェクトとして扱える。

type 2 アクティビティ部から能動的に外部にリクエストを出す。アクティビティ部は分散オブジェクトとしてはクライアントになる。

4·2 RT コンポーネント間の通信についての検討

これと同時に RT コンポーネント間の通信についても検討を行った。RT コンポーネント間の通信はインタフェース部を通しての通信とアクティビティ部で行う通信に大きく分けられる。インタフェース部を通した通信では、主に、インタフェース部への通常のメッセージやアクティビティ部へのコマンドなどがやり取りされ、アクティビティ部で行う通信では、主に、アクティビティ部からのデータ取得やアクティビティ部へのデータ書き込みなどアクティビティ部で必要とされるデータがやり取りされると考えられる。しかし、これについては完全にそのように切り分けられるのか、またはそのように切り分けたほうが良いのかということなど、引き続き検討が必要な項目が数多くあることが分かった。

たとえば、センサなどは type0 のコンポーネントとして扱うことができるが、その場合センサデータのやり取りはインタフェース部を通して行うことになる。しかし、もしセンサが push 型のデータ発信を行う必要がある場合には、アクティビティ部を通さなくてはならなくなる。

またこれらの通信に関しては以下のような問題点、課題が抽出された。

- アクティビティ部がデータを書き換えるため、データのコンシステンシーを保つ必要がある。
- データ取得を通常の分散オブジェクトの呼び出しとして同期的に行う場合、通信遅延が大きくなる。
- データ送信を通常の分散オブジェクトの呼び出しとして同期的に行う場合、不要な返答を待つ必要が生じる。
- 多数のデータ入出力要求を逐次的に行うと各々の遅延が加算される。

5. 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。

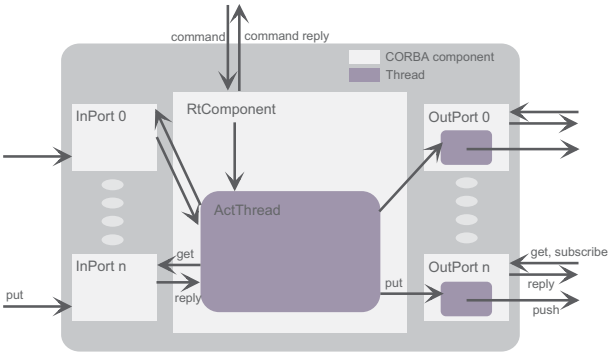

Fig.4 RT コンポーネント

RtComponent オブジェクト 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();
};
```

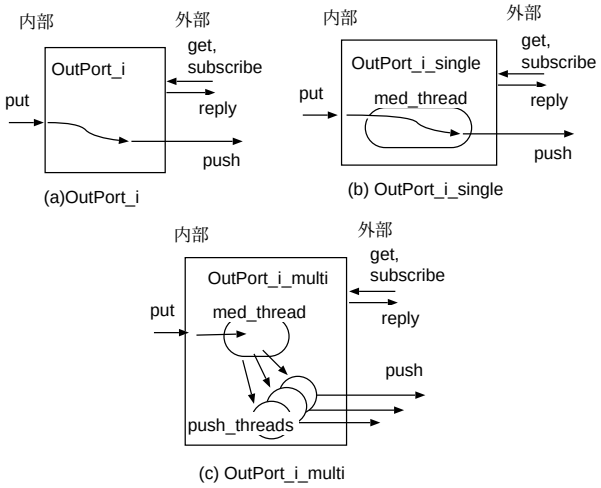

Fig.6 OutPort の実装

6. CORBA オブジェクトの実装概要

6·1 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。

6·2 InPort

InPort の実装例として以下の 2 種類のものを作成した。

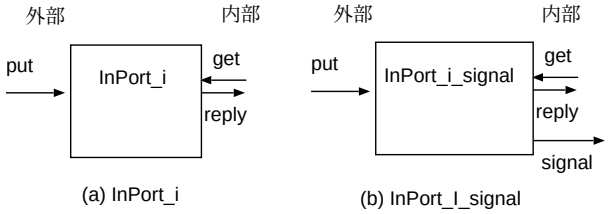

Fig.5 InPort の実装

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。

6·3 OutPort

OutPort の実装例として以下の 3 種類のものを作成した。

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。

7. RT コンポーネントブラウザ

上記の方針により実装した RT コンポーネントの状態を監視したり、相互の接続を構成する操作を行う RT コンポーネントブラウザを作成した。これを図 7 に示す。

各コンポーネントはそれぞれ、長方形で表されその状態を色によって示している。マウスで、クリックすることによりコンポーネントのアクティブ・非アクティブ状態を切り替えることが出来る。

また、各コンポーネントには各機能に応じて入力ポートまたは出力ポートがある。長方形の左側にある窪んだ部分が入力ポート、右側の尖った部分が出力ポートを表している。これらのポートをマウスでドラッグアンドド

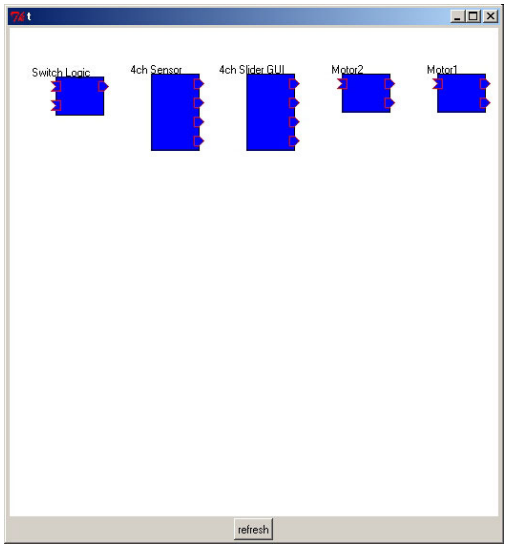

Fig.7 RT コンポーネントブラウザ

ロップすることにより、コンポーネント間の入出力ポートの接続を行うことが出来る。

このGUIを用いることで、コンポーネント状態の監視および、コンポーネント間の接続をビジュアル的に行うことができる。このコンポーネントブラウザはPythonで実装され、WindowsおよびLinux上で動作する。

8. RT コンポーネントデモシステム

上記のRTコンポーネントの有用性を示すために、アクチュエータ、センサ、その他から成るRTコンポーネントデモシステムを製作した。

コンポーネントは、ハードウエアに関連したモータコンポーネント、センサコンポーネント、およびハードウエアの実体を持たないロジックコンポーネント、GUIスライダーコンポーネントをそれぞれ実装した。RTコンポーネントC++およびPythonにより実装されている。特に、Pythonで実装されたコンポーネントはほぼ変更なしで、WindowsおよびLinux上で動作した。

ハードウエアはモータ2台および4チャネルA/D変換器に接続されたセンサ4個(3種類)から構成される。これらは、すべてUSB接続可能なデバイスであり、PCに接続した際にはシリアルデバイスとして認識される。モータコントロールボックスにはバッテリを内蔵、センサボックスはUSBを電源として利用可能な構成になっている。

8·1 モータコンポーネント

モータはロボットを構成するもっとも基本的なコンポーネントである。本デモシステムでは、Megarobotics社のホビー用サーボモータであるAIモータを使用した(図8)。AIモータはシリアルポートを持ち、コマンドにより簡単な位置制御、速度制御、位置計測、電流計測などが行えるモータである。このモータを位置制御モータコンポーネントとして実装した。

入力ポート(InPort)に位置指令値を入力すると、指定された位置まで動く。また、出力ポートは上から、位置出力、電流出力となっており、フィードバックをかけることも可能である。なお、入出力ポートのデータ型はFloat型のスカラー値となっている。

Fig.8 モータコンポーネント(左)とモータ(右)

8·2 センサコンポーネント

センサーユニットは4チャネルの8bit A/D変換器に4個3種類のセンサが接続されている(図9)。センサは、図9の写真の左側から、圧力センサ2個、温度センサ、光センサ(CdS素子)である。

このセンサユニットを4チャネルの出力ポートを有するセンサコンポーネントとして実装した。圧力センサ、温度センサ、光センサはそれぞれ圧力、温度、明るさに比例した値が出力される。出力データ型は各チャネルともにFloat型のスカラー値となっている。

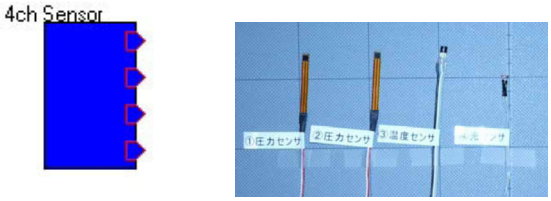

Fig.9 センサコンポーネント(左)とセンサ(右)

8·3 ロジックコンポーネント

ハードウエアの実体を持たないコンポーネントである(図10)。ここで実装したものは、2つの入力(チャネル1、チャネル2)を受け取り、チャネル2の値が閾値より大きければ、チャネル1の値を出力ポートに出力するという簡単なものである。

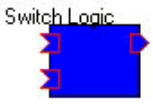

Fig.10 ロジックコンポーネント

8·4 GUI スライダーコンポーネント

GUIもまたコンポーネント化することが可能である。図11は、上記のセンサコンポーネントの代替となる4チャネル出力を有する、GUIスライダーコンポーネントである。このコンポーネントは、スライダーの位置に応じた値が4つの出力ポートよりそれぞれ出力されるコンポーネントとなっている。

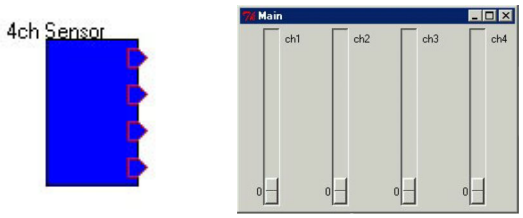

Fig.11 GUI コンポーネント (左) とスライダー GUI(右)

9. RT コンポーネントの接続実験

9·1 力センサ+光センサ+ロジック+モータ

力センサ、光センサ、ロジックおよびモータ 1 個を用いて、図 12 に示す簡単なシステムを構成した。この例では、力センサ、光センサの出力はロジックコンポーネントのチャネル 1(上) およびチャネル 2(下) に入力され、ロジックコンポーネントの出力はモータの入力に接続されている。

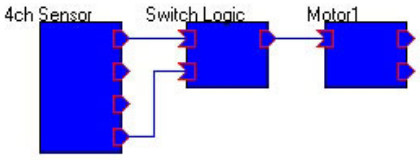

Fig.12

このシステムでは、光の強さがある閾値以上ならば、力センサに加わる力に比例した位置へモータが回転することになる。システムは、実際に上記の通り、明るい時には力センサを指で押すとモータが力に比例して回転し、暗くなると力センサに力を加えてもモータは反応しなくなった。

9·2 モータ 1+モータ 2

モータ 2 個を用いて、図 13 に示す簡単なシステムを構成した。この例では、モータ 1 の位置出力をモータ 2 の位置指令入力に接続し、モータ 2 の位置出力をモータ 1 の位置指令入力に接続する、いわゆる対称型バイラテラル系を構成した。

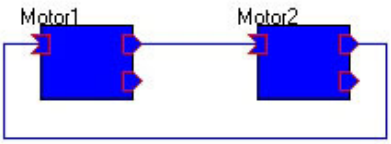

Fig.13

このシステムでは、モータ 1 とモータ 2 の位置の差が発生すると、これを打ち消すようにそれぞれのモータが動く。したがって、モータ 1 の軸を手で動かすと、モータ 2 の軸も同様に追従する。逆に、モータ 2 の軸を手で動かすと、モータ 1 の軸も同様に追従し、双方向に動きおよび力を伝えることが出来る。

実際に上記のシステムを構成し、それぞれのモータを手で動かす実験を行った。片方のモータを手で動かすと、もう片方のモータも追従して動いた。また、両方のモータを同時に手で動かすと、感度は良くないが力フィードバックも行うことができた。

10. 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。

また、いくつかのモータ、センサ、GUI 等のコンポーネント化を行い、およびそれらを接続する GUI システムについても紹介した。実際に、コンポーネントの接続を様々に変え簡単なシステムを構成し、そのいくつかを紹介した。予め必要な機能単位をコンポーネントとして実装しておけば、それらの組み合わせにより多種多様な構成を容易に実現できることが確認できた。現在は、上述したデモシステムのみならず、PA10 等のマニピュレータおよび、これを制御する種々のコンポーネントを実装し、マニピュレータの力制御等の実験においても実用的に使用可能な段階に達しつつある。

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。

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) 「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.
2) 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.
3) 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”, ロボティクスメカトロニクス講演会, 2003.
4) 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.