ユーザマニュアル
RTコンポーネント接続制御モジュール
User's manual concerning
RT component connection control module

中央大学　理工学研究科　電気電子情報通信工学専攻
人間機械協調システム研究室　國井研究室
今井清貴　小島隆史

2008年11月30日

目 次

1　はじめに

1.1　本書について

本書では、RT ミドルウェア上で動作するコンポーネント

- 共有メモリコンポーネントと自動接続ツール群
- RT コンポーネント接続制御モジュール

に関してその機能、利用方法、環境等について説明致します。

2　提供するソフトウェアの機能概要

ロボットのソフトウェアは、一般に Fig.1 に示すようにモジュール間の接続が多く、対応するポート間を手動で接続することでシステムの運用を行っていました。より効率的にシステムを運用するために、「共有メモリコンポーネントと自動接続ツール群」及び「RT コンポーネント接続制御モジュール」を開発しました。

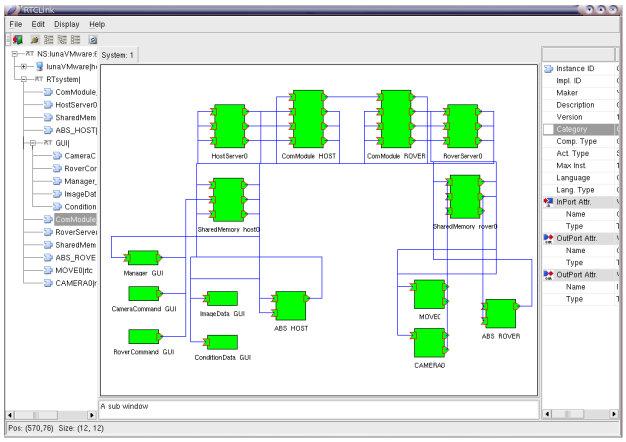

Fig. 1: RT module connection on large-scale RT system

2.1　共有メモリコンポーネントと自動接続ツール群

一つ当たりのモジュールが接続するポートの数をｎ、総モジュール数をｍとしたときに、一つ当たりのモジュールがつなぐ量はｎ×ｍの組み合わせがあり、さらに、すべてのモジュール数分を考慮すると、ｎ×ｍ×ｍとなり、数が増えれば増えるほど 2 乗のオーダーで、接続コストが大きくなります (Fig.2)。このような接続コストの問題を改善することを目的として、共有メモリコンポーネントと自動接続ツール群の研究・開発を行い、ソフトウェアを提供することにしました。

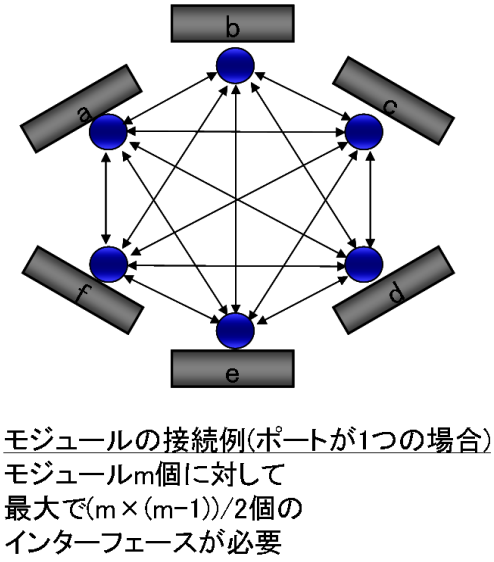

Fig. 2: Example of connecting module(When the port is one.)

2.2 RT コンポーネント接続制御モジュール

RT System Editor は、 RT コンポーネント をリアルタイムにグラフィカル操作する機能を持っているが、コンポーネント組み立て作業を手動で行う仕様になっており、システムの運用・開発過程において、モジュール接続が負担となります。RT コンポーネント接続制御モジュールでは、モジュール接続の過程を軽減するために、下記に示す 3 つの機能を RT System Editor を用いることなく実現します。

1. モジュール間の接続
2. モジュールのアクティブ化
3. モジュールのディアクティブ化

また、モジュールの接続、アクティブ化、ディアクティブ化を簡単に操作可能とするために、GUI によるモジュール操作機能を実現しました (Fig.4)。GUI を用いて IP アドレスを入力し、操作するモジュールの名前及び入出力ポートを選択し、接続の設定を行います。接続設定後、モジュールの接続及び状態制御の操作が可能となります。

モジュール間の接続は、RTCompConnect と定義した接続クラスにより行われています。モジュールのアクティブ化及びディアクティブ化は、RTCompControl と定義した接続クラスにより行われます。GUI からの操作指令は、まず、共有メモリを介して、RTCompConnect または RTCompControl に渡されます。次に、操作指令は、RTCompConnect または RTCompControl の内部で処理されます。RTCompConnect、RTCompControl のクラスは、CorbaNaming.h(CORBA naming service

helper class) と RTObject.h(RT component base class) を利用して構成されています。そのため、ORBvar 管理クラスにアクセス可能であるためモジュールの接続、アクティブ化、ディアクティブ化が可能となっています。

Fig.3 に示す概念に基づいて、RT コンポーネント接続制御モジュールを実装しました。

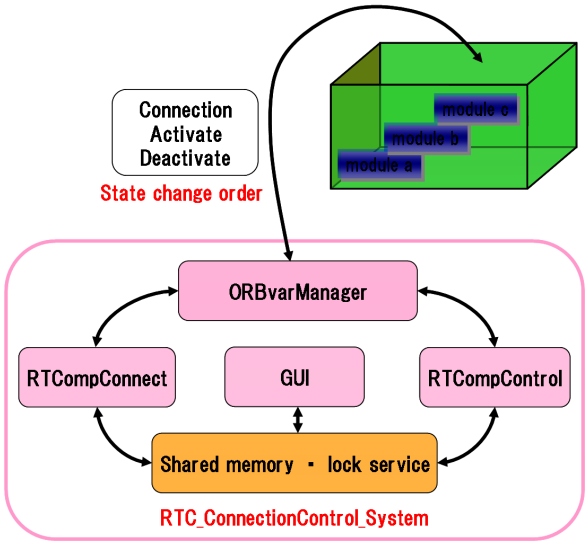

Fig. 3: Software Architecture Model on RT component connection control module

3　提供するソフトウェアの利用方法

3.1　共有メモリコンポーネントと自動接続ツール群の利用方法

共有メモリコンポーネントと自動接続ツール群の利用方法に関して説明致します。

1. Lib ディレクトリに入る。
2. make.sh を実行する。
3. Root 権限で、install.sh を実行
   (/usr/local/lib/ と /usr/local/include にコピーされる)
4. インストールしたライブラリのパーミッションを変える必要があります。
   (chmod 777 ファイル名)

3.2 SHM Automatic Connection System ディレクトリ

Sync 及び WithShmServer は、自動接続が行われるようになっているので、動作確認が簡単であると思います。(接続せずに All Activate するだけです)。現時点では、接続してから実行すると、エラーになる仕様になっています。SHM Automatic Connection System ディレクトリの中には、下記に示す 4 つのディレクトリがあります。

Lib

共有メモリコンポーネントと自動接続ツール群を利用するためのライブラリとシェルスクリプトがあります。

Sync

複数のＣＰＵを想定したサンプルです。共有メモリサーバの同期ポートを接続した場合と、接続しない場合を比較します。ポートをつなぐと同期がはじまり、切断すると同期が止まります。

WithoutShmServer

データベースを使わない接続サンプルです。そのため、通常通りＲＴポートを接続して動作を確認するということになります。このサンプルは、自動接続にすることが望ましいかもしれないことを示唆するためのサンプルなので深い意味はありません。意図としては、２つあり、作成者以外が接続したりするには、少し難しいことがありえるということ (複数の接続パターンが考えられるなど)、どこに何をつなぐかというのは数が増えれば増えるほど、管理が難しくなるということ (接続コストの増大) を示すために作成しました。そこで、これらの問題を改善するためのサンプルとして、WithShmServer を作成しました。

WithShmServer

このサンプルは All Activate 後に、自動的に接続されて動作します。ただし、一つ一つアクティベートしていかないと、エラーが生じる場合があります。原因は、インターフェースクラスにセマフォをつかった順番制御機能などを実装していないためです。今後、修正していきます。

3.3 RT コンポーネント接続制御モジュールの利用方法

RT コンポーネント接続制御モジュールの利用方法に関して説明致します (Fig.4)。

1. `GUI_RTC_ConnectionControl_System` ディレクトリに入る。
2. `./GUI_RTC_ConnectionControl_System` を実行する。
   実行時に IP アドレスが自動入力されます。
3. InPort Profile の設定
   Module Name で InPort 側のモジュール名を選択し、
   InPort Name でモジュールのポートを選択します。
4. OutPort Profile の設定
   Module Name で OutPort 側のモジュール名を選択し、
   OutPort Name でモジュールのポートを選択します。
5. Module Set(接続設定) を行った後、
   connection(モジュール間の接続) を行います。
6. connection(モジュール間の接続) を行った後、activate(アクティブ化),deactivate(ディアクティブ化) の操作が可能になります。

3.4 RTC ConnectionControl System ディレクトリ

GUI により RT コンポーネント接続の設定を行い、その設定情報を共有メモリに書き込み、RT コンポーネントの接続制御を行うモジュールに、接続設定の情報を反映させて、接続制御を行います。RTC ConnectionControl System ディレクトリの中には、下記に示す 3 つのディレクトリがあります。

CoreSystem

共有メモリとロックサービス（セマフォによる排他制御）を提供するモジュールです。

GUI RTC ConnectionControl System

RT コンポーネントの接続を操作する GUI モジュールです。下記に示す 4 つの設定ファイルを内容を変更することで、接続するモジュール名及びポート名を変更することが可能です。`./GUI_RTC_ConnectionControl_System` を実行時に、設定ファイルが読み込まれて、GUI に設定情報が反映されます。

- `InPort_Module_Items`
  InPort 側のモジュール名を設定するファイル
- `InPort_Name_Items`
  InPort 側のポート名を設定するファイル
- `OutPort_Module_Items`
  OutPort 側のモジュール名を設定するファイル
- `OutPort_Name_Items`
  OutPort 側のポート名を設定するファイル

RTCompConnect,Control-1.0.0

共有メモリから接続設定の情報を取得して、RT コンポーネントの接続を行うモジュールです。

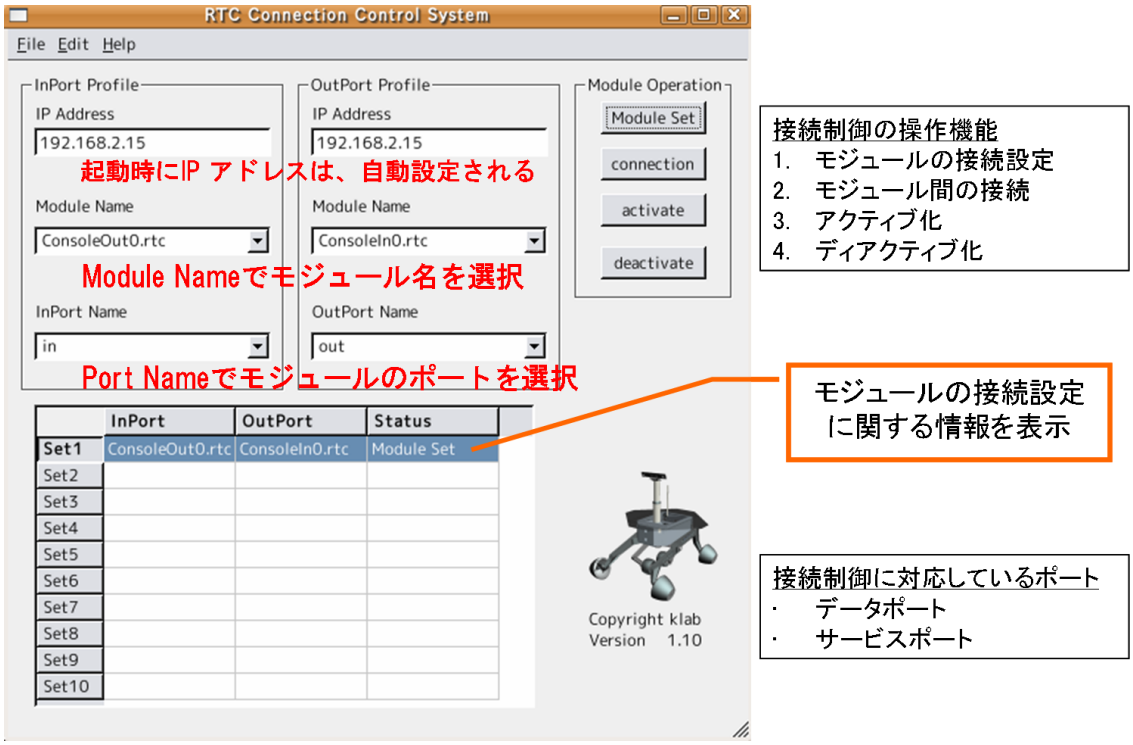

Fig. 4: Use of RT component connection control module

4　環境

4.1　開発環境

- OS：Ubuntu 8.04 LTS
- OpenRTM：OpenRTM-aist-0.4.2
- GUI toolkit：Qt3.3

4.2　動作環境

OpenRTM-aist-0.4.2 がインストールされている Linux 環境であれば基本的にコンパイルし、実行可能です。Windows は sharedMemoryManager,sharedMemoryClient クラスが対応していないため使うことができません。

5　その他

5.1　注意事項

本バージョンのサンプルプログラムでは、同期の際に生じるようなクリティカルセクションについては、サポートしておりません。ゆえに、同時に別々の SHM コンポーネントに作成された同じ共有メモリが書き換えられたときの動作については保証いたしません。

5.2　エラー対策

```
*****共有メモリ・ロックサービスの起動に失敗した場合*****
klab@klab-desktop:~/RTC_ConnectionControl_System/GUI_RTC_ConnectionControl_System
$ ./GUI_RTC_ConnectionControl_System
Make   SharedMemory[  292]  ID:1081357 ,KEY:1929504220 ,Purpose:Rover_Shared_memory
Error in clear : Semaphore information is nothing
Make   Semaphore   [    2]  ID:0 ,KEY:1929504220 ,Purpose:Rover_Semaphore
Read   SharedMemory[  292]  ID:1081357 ,KEY:1929504220 ,Purpose:Rover_Shared_memory
Read   SharedMemory[  292]  ID:1081357 ,KEY:1929504220 ,Purpose:Rover_Shared_memory
Error in inReading : Semaphore information is nothing

*****共有メモリ・ロックサービスの起動に成功した場合*****
klab@klab-desktop:~/RTC_ConnectionControl_System/GUI_RTC_ConnectionControl_System
$ ./GUI_RTC_ConnectionControl_System
Make   SharedMemory[  292]  ID:1802255 ,KEY:1929504193 ,Purpose:Rover_Shared_memory
Make   Semaphore   [    2]  ID:32769 ,KEY:1929504193 ,Purpose:Rover_Semaphore
```

```
Read   SharedMemory[  292]  ID:1802255 ,KEY:1929504193 ,Purpose:Rover_Shared_memory
Read   SharedMemory[  292]  ID:1802255 ,KEY:1929504193 ,Purpose:Rover_Shared_memory
Read   Semaphore   [    2]  ID:32769 ,KEY:1929504193 ,Purpose:Rover_Semaphore
```

共有メモリ・ロックサービスの起動に失敗した場合は、下記の操作が必要になります。

```
klab@klab-desktop:~$ cd /RTC_ConnectionControl_System/CoreSystem/SHMSEM_Manager
klab@klab-desktop:~/RTC_ConnectionControl_System/CoreSystem/SHMSEM_Manager$ ls
Makefile            clearSHMSEM.sh   main.cpp       obj
SHMSEM_Manager.exe  deleteSHMSEM.sh  makeSHMSEM.sh
klab@klab-desktop:~/RTC_ConnectionControl_System/CoreSystem/SHMSEM_Manager
$ make clean all
```

上記の操作を行っても起動に失敗する場合には、`/CoreSystem/SHMSEM_Manager/main.cpp` を開き、再度、保存した後、`make clean all` を行うことで解決することがあります。

参考文献

[1] Richard Volpe, Issa A.D.Nesnas, Tara Estlin, Darren Mutz, Ricahrd Petras, Hari Das, "CLARAty(Caupled Layer Architecture for Robotic Autonomy)",*Jet Propulsion Laoratory,California Institure of Technology,2000,12*

[2] http://www.is.aist.go.jp/rt/OpenRTM-aist/html/index.html

[3] 安藤慶昭, 末廣尚士, 北垣高成, 神徳徹雄, 尹祐根, "RT コンポーネントによるシステム構築法— RT ミドルウエアの基本機能に関する研究開発（その 14）— ", 日本機械学会ロボティクス・メカトロニクス講演会 2005, 2005.06

[4] 原, 高橋, 國井, 安藤,"RT ミドルウエアを利用したモジュールの階層化構成による遠隔移動ロボットのシステム検討及び開発", 第 *12* 回ロボティクスシンポジア予稿集, *pp.498-503,2007*