クァッドロータを制御するRTコンポーネント群

地方独立行政法人 東京都立産業技術研究センター
佐々木 智典

目次

1　はじめに

この文書は題目「クァッドロータを制御するRTコンポーネント群」としてRTミドルウェア コンテスト2012にて発表しますソフトウェアについて説明します。

1.1　ライセンス

- 本ソフトウェアは修正BSD(三条項)ライセンスの下で公開します。ライセンスの条文中にあるとおり， ソフトウェアの動作，使用による結果等に関していかなる保証をするものでもありません。
    - 修正BSDライセンス（参考日本語訳）: http://sourceforge.jp/projects/opensource/wiki/licenses%2Fnew_BSD_license

1.2　凡例

- 本ソフトウェアを収録した圧縮ファイルの名称を shmquadrtcs.tar.bz2 とします。
- この圧縮ファイルを展開した先のディレクトリを "<shmquadrtcs>" によって示します。

1.3　開発・テスト環境

- 開発環境:
    - Ubuntu 12.04 (x86)
    - ATDEv4 (x86)
- 言語: C++
- コンパイラ: GCC 4.4 以降
- テスト環境:
    - Ubuntu 12.04 (x86)
    - atmark-dist 20120727 (ARM9 (i.MX25=Armadillo-440))

1.4　ビルド方法

以下のように ソースファイルを展開したディレクトリにおいて，makeを実行します。

- ホストPC（x86）上で shmport他を利用する場合:

```
$ tar xjf shmquadrtcs.tar.bz2
$ cd shmquadrtcs
$ make x86
```

- Armadillo用のプロログラムをクロスビルドする場合:

```
$ tar xjf shmquadrtcs.tar.bz2
$ cd shmquadrtcs
$ make armel
```

2　shmport: 共有メモリによるデータ送受信

本節ではサービスポートと共有メモリによるデータ送受信を実現するテンプレート クラス ライブラリ shmportについて説明します。

2.1　使用例: PrimitiveProviderとPrimitiveConsumer

shmport の単純な使用例を示し，このライブラリによって提供する内容を紹介します。 shmport の主要なテンプレートクラスは ShmOutPort<T>，ShmInPort<T> の二つです。 これらはOpenRTMのデータポートのテンプレートクラス RTC::InPor<T>，RTC::OunPort<T> とほぼ同様に使うことができます。以下では型Tを基本データ型（あるいはプリミティブ型， すなわち int, long, float など）として，その値を出力するコンポーネント PrimitiveProvider，入力として受け取るコンポーネント PrimitiveConsumer を定義して ShmOutPort<T>，ShmInPort<T> の 使用例を示します。

2.1.1　共有メモリ出力ポート: ShmOutPort<T>

RTコンポーネントからデータの出力を行う ShmOutPort<T> について使用例を示します。 ShmOutPort<T>を使うコンポーネント PrimitiveProviderを定義するヘッダファイルは 次のように書けます。

```
// ... 省略 ...
#include <shmport/ShmOutPort.hpp>

class PrimitiveProvider  : public RTC::DataFlowComponentBase
{
  // ... 省略 ...
private:
  ShmOutPort<long> m_shmOutPort;
  // ... 省略 ...
};
```

PrimitiveProviderのコンストラクタにおいて，ShmOutPort<T>のインスタンス m_shmOutPort は次のように初期化します。

```
PrimitiveProvider::PrimitiveProvider(RTC::Manager* manager)
  : RTC::DataFlowComponentBase(manager),
    m_shmOutPort("shmOut")
{
}
```

ここでShmOutPort<T>のコンストラクタの引数"shmOut"はポートの名称です。 この名前がサービスポートの名前として RTSystemEditor等で表示されます。

m_shmOutPort をRTコンポーネントに取り付けるには次のようにします。

```
RTC::ReturnCode_t PrimitiveProvider::onInitialize()
{
  m_shmOutPort.attachTo(*this);
  // ... 省略 ...
  return RTC::RTC_OK;
}
```

m_shmOutPort によってデータを送信するには次のようにメンバ関数 write() を使います。

```
RTC::ReturnCode_t PrimitiveProvider::onExecute(RTC::UniqueId ec_id)
{
  static size_t idx = 0;

  if(idx < m_sendingData.size()){
    m_shmOutPort.write(m_sendingData[idx]);
    idx++;
  }
```

```
  return RTC::RTC_OK;
}
```

以上のように テンプレートクラス ShmOutPort<T> をデータポートと同様に使うことで データを共有メモリ経由で送信することができます。

2.1.2　共有メモリ入力ポート: ShmInPort<T>

次にRTコンポーネントへデータの入力を行う ShmInPort<T> について使用例を示します。 ShmInPort<T> を使うコンポーネント PrimitiveConsumer の定義は次のように書けます。

```
// ... 省略 ...
#include <shmport/ShmInPort.hpp>

class PrimitiveConsumer  : public RTC::DataFlowComponentBase
{
  // ... 省略 ...
private:
  ShmInPort<long> m_shmInPort;
  long m_boundVar;
};
```

ここで m_boundVar は ShmInPort<T> のインスタンス m_shmInPort に結びつける 変数で，データの読み込みとともにその値が更新されます。この変数の型は ShmInPort<T> の T に一致させます。

PrimitiveConsumer のコンストラクタにおいて，ShmInPort<T> のインスタンス m_shmInPort は次のように初期化します。

```
// ... 省略 ...
PrimitiveConsumer::PrimitiveConsumer(RTC::Manager* manager)
  : RTC::DataFlowComponentBase(manager),
    m_shmInPort("shmIn", m_boundVar),
    m_boundVar(0)
{
}
```

ここでShmInPort<T>のコンストラクタの第1引数 "shmIn" はポートの名称です。 この名前がサービスポートの名前として RTSystemEditor等で表示されます。 コンストラクタの第2引数には m_shmInPort に結びつける変数を指定します。

m_shmInPort をRTコンポーネントに取り付けるには次のようにします。

```
RTC::ReturnCode_t PrimitiveConsumer::onInitialize()
{
  m_shmInPort.attachTo(*this);
  return RTC::RTC_OK;
}
```

m_shmInPort によってデータを受信するには次のようにメンバ関数 read() を使います。

```
RTC::ReturnCode_t PrimitiveConsumer::onExecute(RTC::UniqueId ec_id)
{
  using namespace std;

  if(m_shmInPort.isNew()){
    m_shmInPort.read();
    cout << m_boundVar << endl;
  }

  return RTC::RTC_OK;
}
```

以上のように テンプレートクラス ShmInPort<T> をデータポートと同様に使うことで データを共有メモリ経由で受信することができます。

2.1.3　ビルド

shmportを利用するプログラムをビルドする際には，RTミドルウェアのライブラリの他に， <shmquadrtcs>/lib/libshmport.so および インストール先の環境の librt.so とのリンクが必要です。

2.2　ShmOutPort<T>, ShmInPort<T>のTに指定可能な型

共有メモリにデータを置く都合上，T は基本データ型，ポインタを使わない，もしくは 共有メモリに置かれることを前提にポインタを使うように設計された型である 必要があります。 現状のshmportの実装において ShmOutPort<T>，ShmInPort<T> の T に指定可能な型は次のとおりです。

Tに指定可能な型

型	対応するShmInPort，ShmOutPortの別名定義
bool	ShmBoolInPort / ShmBoolOutPort
short	ShmShortInPort / ShmShortOutPort
int	ShmIntInPort / ShmIntOutPort
long	ShmLongInPort / ShmLongOutPort
unsigned short	ShmUShortInPort / ShmUShortOutPort
unsigned int	ShmUIntInPort / ShmUIntOutPort
unsigned long	ShmULongInPort / ShmULongOutPort
float	ShmFloatInPort / ShmFloatOutPort
double	ShmDoubleInPort / ShmDoubleOutPort
ShmString	ShmStringInPort / ShmStringOutPort

ここで ShmString は 共有メモリにデータが置かれることを前提に 設計されている文字列クラスです。ShmString はおおよそSTLの std::stringと同様に使えます（実態上，ShmString は boost::interprocess::string の別名として定義しています）。

上記以外の型を T に指定するには, ShmInPort<T>, ShmOutPort<T> の 実装を明示的に記述する必要があります。これらの実装には，ポインタ メンバを含まないクラスであれば，次のように既存の実装が記述された ヘッダ ShmInPort_primitive_impl.hpp を利用できます。 型 UserData 用の入力ポート ShmInPort<UserData> の実装を記述する ソースファイルを ShmUserDataInPort.cpp とすると，その内容は 次のように書けます。

```
#include "ShmUserDataInPort.hpp"

// プリミティブ型用の実装を利用する
#include <shmport/ShmInPort_primitive_impl.hpp>

// テンプレートを実体化する
template class ShmInPort<UserData>;
```

これに対応するヘッダ ShmUserDataInPort.hpp においては次のように typedefを記述します。

```
#ifndef SHM_USER_DATA_IN_PORT_HPP
#define SHM_USER_DATA_IN_PORT_HPP

#include <shmport/ShmInPort.hpp>
#include "UserData.hpp"

// 別途 cppファイルで明示的に実体化する型
typedef ShmInPort<UserData> ShmUserDataInPort;
```

```
#endif // SHM_USER_DATA_IN_PORT_HPP
```

上記 ShmInPort<UserData> を利用するソースファイルでは ShmUserDataInPort.hpp を #include し，ビルド時に ShmUserDataInPort.cpp をコンパイルしたオブジェクトファイルと リンクします。

T に指定する新たなクラスを定義する場合は次のような事項に 注意が必要です。

- newやmallocで確保したデータはそのままでは使えないこと。
    - new や malloc で確保したデータへのポインタは，その 確保を実施したプロセスの中でのみ有効です。このような ポインタ型のメンバを持つクラスのインスタンスを shmportで送受信する場合は，データを共有メモリに確保するように クラスの設計を変更する，もしくは送受信用に変換を 行うクラスを用意する必要があります。
- 仮想関数が使えないこと。
    - 仮想関数の仕組みにはポインタ型（メンバへのポインタのテーブルvtable）が 関わっていますので，これに対する工夫なしでは送信側のコンポーネントで メンバ関数が正しく呼び出せても，受信側のコンポーネントでは正しく呼び出せず， 原因が特定しづらいセグメントフォールトを引き起こします。

3　shmquad: クァッドロータを制御するRTコンポーネント群

<執筆中です>