CONFERENCE DIGEST

ロボティクス・メカトロニクス講演会2010

2010 JSME Conference on Robotics and Mechatronics

ROBOMEC2010 in ASAHIKAWA

ロボティクス・メカトロニクス・フロンティア・ビッグバン

Robotics・Mechatornics・Frontier・Big-Bang

June 13 Sun. - 16 Wed., 2010

Asahikawa TAISETSU Arena

主催　社団法人 日本機械学会 ロボティクス・メカトロニクス部門

The Japan Society of Mechanical Engineers, Robotics and Mechatronics Division

OpenRTM-aist-1.0 における新しいデータポートの実装

Implementation of new Data Ports in the OpenRTM-aist-1.0

○ 栗原 眞二 (産総研)　片見 剛人 (富士ソフト)
白田 浩昭 (テクノプロ・エンジニアリング)　宮本 晴美 (テクノプロ・エンジニアリング)
坂本 武志 (テクノロジックアート)　正 安藤 慶昭 (産総研)

Shinji KURIHARA, National Institute of Advanced Industrial Science and Technology, shinji.kurihara@aist.go.jp
Tsuyoto KATAMI, FUJISOFT
Hiroaki HAKUTA, TechnoPro Engineering
Hiromi MIYAMOTO, TechnoPro Engineering
Takeshi SAKAMOTO, TECHNOLOGIC ARTS
Noriaki ANDO, National Institute of Advanced Industrial Science and Technology

RT-Component framework in OpenRTM-aist provides DataPorts (a.k.a. InPort and OutPort) as data-centric communication interface. DataPorts have provided control methods for communication characteristics that vary according to the relation between RT-Components. Some additional functionalities for data exchange are introduced in the newly released OpenRTM-aist-1.0. In this paper, design and implementation of these features are shown and discussed.

Key Words: RT-Component, RT-Middleware, Modularization

1. はじめに

OpenRTM-aist[1] の RT コンポーネントは，コンポーネント間でデータ指向の通信を行うための，データポートという機能を持っており，他のコンポーネントからデータを受け取るポートを InPort，他のコンポーネントにデータを送るポートを OutPort と呼ぶ．

データポートにおいては，用途に応じて様々な通信方法が考えられるため，これを実現するための必要な機能について，これまで [2, 3, 4] にて議論し，実装を行ってきた．

本稿では，[4] で実装された機能に加えて，OpenRTM-aist-1.0 で導入された，よりきめ細かなフロー制御を実現するバッファリングポリシー，および異言語間で共有メモリなど様々な通信媒体を介してデータ交換を行うためのデータのシリアライズ (直列化) 方法について設計方針と実装を示す．

2. 設計方針

本節では，これらの OpenRTM-aist-1.0 における，データポートの新機能について設計方針を述べる．

2.1 多様な言語・通信方式への対応

以前のデータポートでは，CORBA による通信 (CORBA Any 型) とソケットによる直接通信 (Raw TCP 型) の通信方法を提供していた [2]．CORBA Any 型ではデータを Any 型 (任意の型を格納可能な型) に格納することで暗黙的にシリアライズを行い，Raw TCP 型では，CORBA で用いられる CDR(Common Data Representation) 方式によるシリアライズをそれぞれ個別に行っていた．しかしながら，共有メモリ型通信等新たな通信インターフェースによるデータポートの拡張を行う際には，個別にシリアライズ機構を実装する必要がある．そこで，通信インターフェースレベルでは，データ型に依存せず単なるバイト列の受け渡しになるよう，OutPort に書き込まれた直後に CDR 形式でシリアライズする方式を採用する．この方式を採用することにより，通信方式が新たに追加された場合でも，個別にシリアライズ機構を実装する必要がなくなると共に，多様な言語や通信方式に対応することが可能となる．

これに伴い，CORBA による通信インターフェースも CORBA Any 型から octet(ネットワーク転送中に変更されないことが保障された 8 ビットデータ型) の sequence 型に変更し，不要な変換を排除するよう変更を行う．

以下に，新たな CORBA CDR 型インターフェース定義を示す．

```
// file: DataPort.idl

module OpenRTM
  {
    enum PortStatus
     {
         PORT_OK,
         PORT_ERROR,
         BUFFER_FULL,
         BUFFER_EMPTY,
         BUFFER_TIMEOUT,
         UNKNOWN_ERROR
     };

    typedef sequence<octet> CdrData;

    interface InPortCdr
     {
         PortStatus put(in CdrData data);
     };

    interface OutPortCdr
     {
         PortStatus get(out CdrData data);
     };
};
```

2.2 多様なデータフロー制御への対応

ある種のコンポーネント化された制御対象に対して，目標値を送る場合，通常最新の値のみが取得できれば問題ない．しかしながら，データのロギングや，データ生成・消費速度が異なるモジュール同士の接続などが求められるケースも存在する．こうした多様な利用方法に対応させるために，データを出力する OutPort のバッファリングとデータ送信にポリシーを導入する．ポリシーの設定は，接続される両端の RT コンポーネントの特性や，アプリケーションの性質により変わるため，接続時でも行えるようにする必要がある．

2.3 データポート間の接続数の制限

OpenRTM-aist-1.0 のデータポートでは，図 1 のように，一つのデータポートに対して複数のポート間接続が可能となっている．[5]．一方，複数のポート間接続を想定していないデータポートに対しても，複数のポート間接続を行う事ができるため，これにより何らかの問題が発生するといった懸念もある．

こうした問題に対応するため，データポートにおけるポート間接続の上限値を設定できるようにするとともに，上限値以上の接続は行わないようにポート接続を制限するための枠組みを導入する．ポート間接続の上限値の設定は，RT コンポーネントの使用目的によって変わる可能性があるため，コンポーネント用の設定ファイルにて設定できるようにする必

要がある．

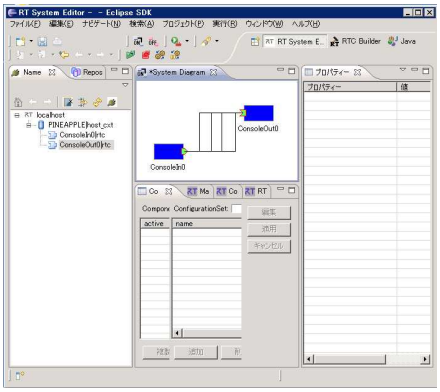

Fig.1 Multi Connection Diagram of RTSystemEditor

3. 機能

本節では，OpenRTM-aist-1.0 のデータポートにて新たに導入された機能のうち，“シリアライズ機構” と “バッファリングポリシーによるバッファ制御” について述べる．さらに，シリアライズ処理にはエンディアンを指定する必要があるため，ポート間のエンディアン調停についても述べる．

3.1 シリアライズ機構

CORBA では，IDL (Interface Definition Language) で定義されたすべてのデータ型について，自動的に CDR との変換を行うためのオペレータが定義される．これを利用し，OutPort に書き込まれたデータを，即座に CDR 型に変換する．Publisher がバッファを持つ場合は，CDR 型を格納可能なバッファを持つ．データは最終的には CDR 形式のバイト列として InPort に引き渡される．InPort 側では受け取ったバイト列を CDR 型としてバッファに格納する．このデータは，読みだされる直前に CDR 型から元の型に変換される．

CDR 型は言語に非依存なシリアライズ方式のため，異なる CPU アーキテクチャや異なる言語間の InPort と OutPort であっても，型安全性を保持したままデータの受け渡しが可能である．

3.2 ポート間のエンディアン調停

上述したように，新たなデータポートではシリアライズ機構が実装された．シリアライズ処理ではエンディアンを指定する必要があるため，ポート間でのエンディアン調停を行う必要がある．ここでは，データポート間のエンディアン調停の方法について述べる．

OpenRTM-aist で対応しているエンディアンの方式 (以下，エンディアン) は “little” と “big” であり，データポートの接続時に ConnectorProfile の properties 属性にエンディアンを指定する事ができる．また，エンディアンは，“little,big” のように複数指定する事も可能である．エンディアンの調停は図 2 のように，データポート間の接続処理過程において行われる．エンディアンが ConnectorProfile にて指定され，かつ，対象のデータポートが指定されたエンディアンに対応していない場合，ConnectorProfile のエンディアン情報からそのエンディアンタイプを削除する．指定されたエンディアンに対応している場合は，ConnectorProfile のエンディアン情報に関しては特に何も行わない．このようにして，最終的に ConnectorProfile のエンディアン情報に残ったエンディアンタイプが，対象のデータポート間で採用される．なお，ConnectorProfile にてエンディアンが指定されていない場合は，リトルエンディアンが適用される．

3.3 バッファリングポリシーによるバッファ制御

送信ポリシーについては [4] において導入されいる．新たなデータポートにおいては，バッファリングに関する以下のポリシーを新たに導入する．

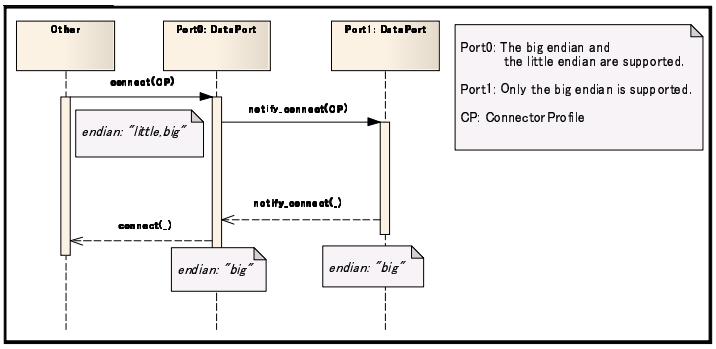

Fig.2 The flow diagram of endian mediation

- バッファがエンプティ状態でのデータ読み出しに関するポリシー
- バッファがフル状態でのデータ書き込みに関するポリシー

また，これらのポリシーによるバッファ制御を行うために必要な情報として，

- バッファサイズ
- バッファへの書き込み・読み出しにおけるタイムアウト時間

を設定可能としている．

なお，これらのポリシーは，設定ファイルによるポート全体に対する指定と，RTSystemEditor 等のツールによる，接続ごとの指定が可能である．

設定ファイルによるバッファのポリシーの指定書式は以下に示すとおりである．

```
# file: component.conf
# InPort name is "in".   OutPort name is "out".

# The length of inport's buffer. default length is 8.
port.inport.in.buffer.length: 8

# The policy about reading in a buffer empty state.
# policy: readback(default), do_nothing, block
port.inport.in.buffer.read.empty_policy: block

# Timeout time for data read.
# Default timeout time is 1.0 second.
# It does not time out, when a value is 0.
port.inport.in.buffer.read.timeout: 1.0

# The length of outport's buffer. default length is 8.
port.outport.out.buffer.length: 8

# The policy about writing in a buffer full state.
# policy: overwrite(default), do_nothing, block
port.outport.out.buffer.write.full_policy: block

# Timeout time for data write.
# Default timeout time is 1.0 second.
port.outport.out.buffer.write.timeout: 1.0
```

接続時におけるポリシーの指定に対応するため，RT コンポーネント操作ツールである RTSystemEditor の接続設定ダイアログに対して，図 3 のような設定項目を追加した．これにより，個々の接続に対して個別にポリシーを設定することが可能である．

4. 実装

上述したように，新たなデータポートではシリアライズ機構が実装された．本節では，OpenRTM-aist が対応しているそれぞれのプログラミング言語 (C++，Python，Java) におけるシリアライズ機構の実装方法について述べる．

4.1 C++言語でのシリアライズ機構の実装

C++言語 (以下，C++) でのシリアライズ処理には omniORB の cdrMemoryStream クラスを使用している．cdrMemoryStream クラスを使用する事で，エンディアンを指定する事が可能であるため，ポート間の接続に応じたシリアライズ処理を行うことが可能となる．

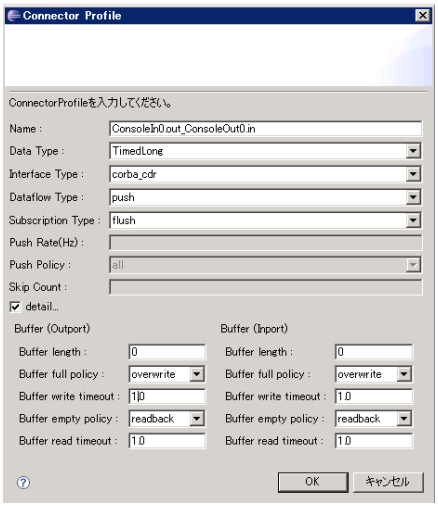

Fig.3 Connector Profile Dialog of RTSystemEditor

また，omniORB では，IDL ファイルで定義したデータ型と cdrMemoryStream との変換を行うためのオペレータが準備されているため，cdrMemoryStream クラスを使用する事で，データポートで使用するデータ型 (TimedLong，TimedLongSeq など) と CDR 型との変換が容易に行えるというメリットがある．

C++でのシリアライズ処理を以下に示す．

```
// cdrMemoryStream m_cdr;

// isLittleEndian(): return it whether endian setting.
//      Return true in the case of "little",
//      false in "big" than it.

template <class DataType>
ReturnCode write(const DataType& data)
 {
  // rewind pointer
  m_cdr.rewindPtrs();

  // set endian
  m_cdr.setByteSwapFlag(isLittleEndian());

  // serialize
  cdrMemoryStream _cdr;

 data >>= _cdr;

 ::OpenRTM::CdrData cdr_data(data.bufSize(),data.bufSize(),
                             static_cast<CORBA::Octet*>
                                    (data.bufPtr()), 0);
 }
```

4.2 Python 言語でのシリアライズ機構の実装

Python 言語 (以下，Python) でのシリアライズ処理には omniORBpy の cdrMarshal()/cdrUnmarshal() 関数を使用している．これらの関数を使用する場合，特定の CORBA データ型についての情報を格納するためのコンテナ (以下，TypeCode) とエンディアンタイプが必要である．TypeCode とエンディアンタイプが揃っていれば，これらの関数を使用するだけでデータのシリアライズが行えるため，C++や Java に比べ，少ない行数で実装することができる．

Python でのシリアライズ処理を以下に示す．

```
from omniORB import *
from omniORB import any

# self._endian: True is little endian, False is big endian.
def serialize(self, data):
  # get data typecode
  _type = any.to_any(data).typecode()

  # data -> (conversion) -> CDR stream
  cdr_data = cdrMarshal(_type, data, self._endian)

  # CDR stream -> (conversion) -> data
  _data = cdrUnmarshal(_type, cdr_data, self._endian)
```

4.3 Java 言語でのシリアライズ機構の実装

Java 言語 (以下，Java) では，IDL ファイルを IDL コンパイルすることで IDL にて定義したクラスや構造体のヘルパークラスが生成される．このヘルパークラスでは，シリアライズしたいデータを OutputSteam 型に変換するためのメソッドが準備されている．また，OutputStream クラスではバイト列を返す toByteArray() というメソッドが実装されている．これらを組み合わせて使用する事で，シリアライズ処理を容易に行う事ができる．

Java でのシリアライズ処理を以下に示す．

```
//  private com.sun.corba.se.spi.orb.ORB m_spi_orb;
//  protected boolean m_isLittleEndian;

public <DataType> ReturnCode write(final DataType data,
                                   OutputStream out)
 {
    OutputStream cdr;
      = new EncapsOutputStream(m_spi_orb,m_isLittleEndian);

    TimedLongHolder tlong = new TimedLongHolder();

    tlong.value = data;

    // Serialization processing TimedLong -> CDR
    tlong._write(cdr);

    byte[] ch = cdr.toByteArray();

    out.write_octet_array(ch,0,ch.length);
 }
```

5. おわりに

本稿では，OpenRTM-aist-1.0 にて新たに機能拡張されたデータポートの設計方針，機能，実装について述べた．今回，新たに実装された “バッファリングポリシーによるバッファ制御”により，様々なデーターフロー制御を行うことが可能となった．また，“シリアライズ機構”の実装により，様々な RT システムに応じたデータ送受信が実現できるようになった．今後は，CORBA 以外の TCP ソケットを直接利用したコネクタや，共有メモリ型を利用したコネクタなどを実装し，通信方式の拡張を行うとともに，バッファリングポリシーやデータ送信ポリシーの有効性についても検証していく予定である．

文献

[1] OpenRTM-aist Web page, http://www.openrtm.org

[2] 安藤他, “RT コンポーネントの InPort/OutPort データ転送方法の多様化-Raw TCP/IP Socket によるデータ転送-”, 第 7 回計測自動制御学会システムインテグレーション部門講演会 2006 (SI2006), p.3B2-1, 2006.12

[3] 安藤他, “RT コンポーネント間のデータ送受信方法に関する考察”, 日本機械学会ロボティクス・メカトロニクス講演会 2008,1P1-E08, 2008.06

[4] 安藤他, “コンポーネント間の多様なポリシに基づく送受信方式を実現するデータポートの実装-OpenRTM-aist-1.0 の新しいデータポート-”, 第 10 回計測自動制御学会システムインテグレーション部門講演会 2009 (SI2009), p.3D3-6, 2009.12

[5] 安藤, “OpenRTM-aist-1.0 の新機能”, 第 9 回計測自動制御学会システムインテグレーション部門講演会 2008 (SI2008), p.2L1-3, 2008.12