2007 JSME Conference on Robotics and Mechatronics

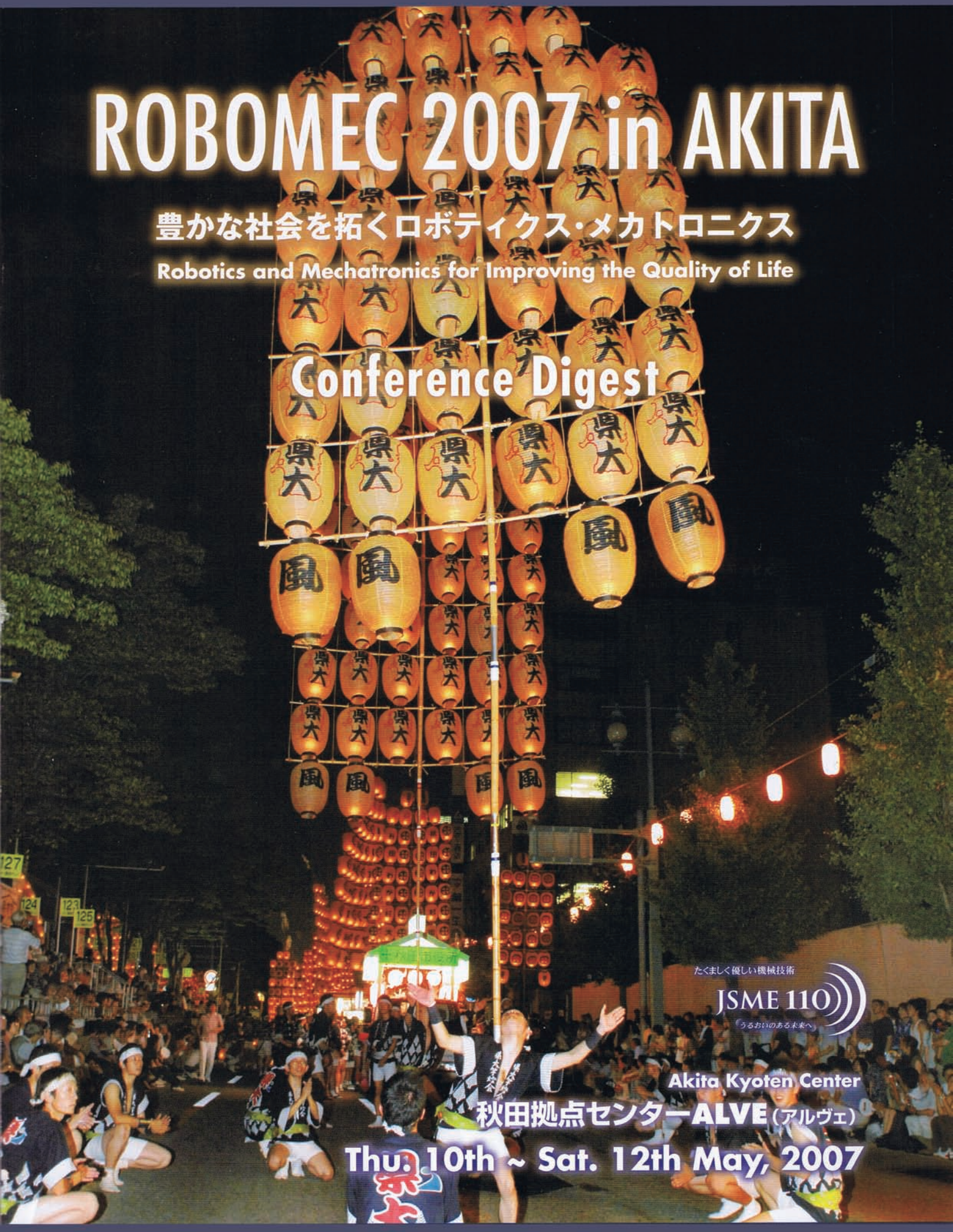
主催　社団法人 日本機械学会 ロボティクス・メカトロニクス部門

ROBOMEC 2007 in AKITA

豊かな社会を拓くロボティクス・メカトロニクス

Robotics and Mechatronics for Improving the Quality of Life

Conference Digest



たくましく優しい機械技術

JSME 110

「うるおいのある未来へ」

Akita Kyoten Center

秋田拠点センターALVE（アルヴェ）

Thu. 10th ~ Sat. 12th May, 2007

社団法人 日本機械学会

〒160-0016 東京都新宿区信濃町35番地 信濃町煉瓦館5階 TEL. (03) 5360-3500　FAX (03) 5360-3508

OMG RTC 標準仕様に準拠したRTミドルウエアの実装

OpenRTM-aist-0.4.0 新機能の紹介

The OMG RTC compliant RT-Middleware implementation
An introduction of OpenRTM-aist-0.4.0 new features

○正 安藤慶昭 (産総研) 正 神徳徹雄 (産総研)
正 末廣尚士 (産総研) 正 北垣高成 (産総研)

Noriaki ANDO, Tetsuo KOTOKU, Takashi SUEHIRO, Kosei KITAGAKI
National Institute of Advanced Industrial Science and Technology
{n-ando, t.kotoku, t.suehiro, k.kitagaki}@aist.go.jp

We have standardized RT-Component model in OMG that is international software standardization organization. The RT-Component specification was adopted in OMG on September 2006 as "OMG Robotics Technology Component Specification (OMG RTC Specification)". In this paper, new version of OpenRTM-aist that is based on OMG RTC Specification is implemented, and its new features are presented.

Key Words: RT, Middleware, Standardization

1. はじめに

著者らは, ロボットの機能要素をRTコンポーネントと呼ぶ構成要素としてモジュール化し, その組み合わせによりロボットシステムを容易に構築するためのプラットフォームとしてRTミドルウエアを提案してきた [1, 2]. また, 分散オブジェクトミドルウエアであるCORBAに基づいたRTミドルウエア・コンポーネントフレームワークとしてOpenRTM-aist-0.2.0と呼ばれる実装を公開してきた [3].

RTミドルウエアのようなプラットフォームは, ユーザに広く利用され, 多くのコンポーネントを共有することで, 初めて効率的なロボット開発環境が実現される. そのため, プラットフォーム自体は特定の実装に依存せず, 持続的・自律的発展可能なオープンアーキテクチャであることが望ましい. そこで, 我々はRTミドルウエアの実装を公開すると共に, そのインターフェース仕様を公開してきた [3]. 結果として, Java版やMicrosoft.NET版のRTミドルウエアが有志の手により開発され, 多くの言語で書かれたRTコンポーネントの相互運用が可能になった [4].

さらに, RTミドルウエア仕様のオープン化を確立するため, RTコンポーネントのアーキテクチャを, ソフトウエア標準化団体であるOMG (Object Management Group) に提案し, 標準化を進めてきた. RTコンポーネントの標準仕様が2006年9月にOMGにおいて承認され, その後約1年間の最終文書化プロセスを経た後, 2007年末には "OMG Robotic Technology Component Specification" として一般に公開される見込みである.

本稿では, OMGで承認されたRTコンポーネントの標準仕様の概要, およびこの仕様に準拠したRTミドルウエアの新バージョンOpenRTM-aist-0.4.0の新機能について述べる.

2. OMGにおける標準化

OMG(Object Management Group) は1989年に設立されたソフトウエア標準化団体である. OMGが主導して策定した代表的標準仕様としては, 分散オブジェクトミドルウエアCORBA (Common Object Request Broker Architecture), ソフトウェア分析・設計のためのモデリング言語UML (Unified Modeling Language) 等がある. 特定のソフトウェア企業に依存しない中立の非営利団体として, オープンなプロセスによって各種標準を策定している.

2.1 MDA (Model Driven Architecture)

OMGにおける標準化の特徴として, MDA (Model Driven Architecture) に基づくモデルベースの標準仕様が挙げられる.

MDAはOMGが提唱する, モデリング主導のシステム開発およびライフサイクル管理を実現するための参照アーキテクチャである. 中心となるモデルは, プラットフォームに非依存なPIM (Platform Independent Model) と プラットフォーム依存モデルPSM (Platform Specific Model) の2階層から構成される.

PIM (Platform Independent Model) PIMはUMLおよび文章により記述されるプラットフォームに依存しないシステムのモデルである. UMLのメタモデル図やプロファイル図によりクラスやインターフェースの関係等を記述し, シーケンス図やステートマシン図等を利用し振舞いや状態遷移を記述することで, 標準化対象をモデル化し仕様記述を行う.

PSM (Platform Specific Model) PSMはPIMから導出される各プラットフォームに特化したモデルであり, 実際の開発者はこのモデルを元に実装を行う. C++やJavaといった言語や, CORBAやRMIといった分散オブジェクトミドルウエアなどの特定のプラットフォームに対して, インターフェースやクラス定義を中心とした仕様となる.

MDAは, 実装・実行する言語やミドルウエアが技術的進歩により変化したとしても, 標準化されるシステムの基本的なモデルの変化は緩やかであり, たとえ実装技術が変化してもモデルは再利用可能であるという考え方に基づいている.

2.2 RTコンポーネントモデルの標準化

RTコンポーネントモデルを標準化するため, 著者らは2004年からOMGにおける標準化活動を行ってきた [5]. 2006年9月にOMGにおいて, 著者らが提案した "Robotics Technology Component Specification (RTC Specification)" の草案がOMGの標準化作業部会であるAB (Architecture Board) により承認された. このOMG RTC Specificationは 以降FTF (Finalization Task Force) において最終文書化が行われるとともに, 提案者はこの標準仕様に基づいた実装を作成し, 仕様の整合性・完全性を検証することが求められる.

3. OMG RTC Specification

本節では, この OMG RTC Specification の PIM を中心に, 標準仕様の概要を述べる.

3.1 RT コンポーネント PIM

OMG RTC Specification は大きく分けて図 1 に示す 3 つのパッケージおよび, 既存の OMG 標準仕様である SDO (Super Distributed Object) から構成される [6]. パッケージとは, モデルの論理的集合を表し, 図中の矢印はパッケージ間の依存性を示している. すなわち, LighweightRTC パッケージは他に対して独立であり, ExecutionSemantics パッケージは LighweightRTC パッケージに, Introspection パッケージは LightweightRTC, ExecutionSemantics 各パッケージに依存していることを示している. 以下, 各パッケージごとに仕様の概要を示す.

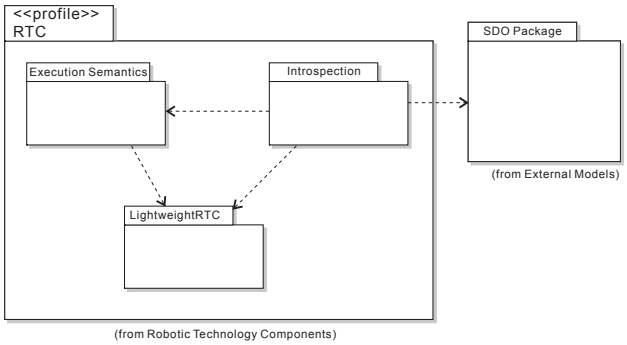


Fig.1 OMG RTC Package diagram.

3.2 LightweightRTC

LightweightRTC は RT コンポーネントの最低限備えるべき機能を規定したモデルである. LightweightRTC は,

- 外部との相互作用を行う Port
- 内部状態
- 実行主体である Execution Context

のみを持つ, 原則として静的に構成されるシステムのためのコンポーネントモデルである.

なお, OpenRTM-aist-0.2.0 で規定されていた Activity は本標準仕様では Component Action と呼ばれ, 状態遷移についても若干の変更がなされた. さらに, 以前はコンポーネントと一体であった実行主体 (実装においてはスレッドに相当する) が Execution Context と呼ばれるエンティティとして, 明示的にコンポーネントから分離された.

図 2 に LightweightRTC の状態遷移を示す. 実行主体の論理表現である Execution Context は, スレッド停止状態に相当する Stopped 状態と, スレッド動作状態に相当する Running 状態を持つ. OpenRTM-aist-0.2.0 における Ready, Active, Error 状態は, Execution Context に関連付けられ, それぞれ Inactive, Active, Error 状態に割り当てられた. また, 以前定義されていた過渡状態 (Initializing, Starting, Stopping, Aborting, Exiting) は縮退し, 各状態のアクションはそれぞれ図中の on_initialize, on_activate, on_deactivate, on_aborting, on_finalize アクションに割り当てられた. なお, 以前の FatalError 状態及び付随するアクションは削除された.

3.3 Execution Semantics

Execution Semantics パッケージは RT コンポーネントの内部ロジックの実行形態モデルである. Execution Semantics には大きく分けて以下の 3 種類のコンポーネント型が規定されている.

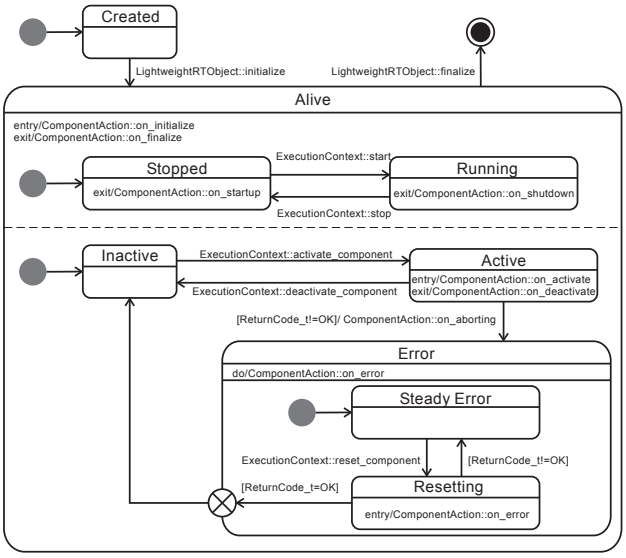


Fig.2 RTC's state machine diagram.

Data Flow Component データ取得・データ処理・データ出力からなるロジックを周期的に実行するタイプのコンポーネント. 単体コンポーネントである DataFlowParticipant 型のコンポーネントと, 複合型である DataFlowComposite 型のコンポーネントがある.

FSM Component 外部からの呼び出しやイベントなどに応じて受動的にロジックが駆動される FSM(Finite State Machine) タイプのコンポーネント. 単体コンポーネントである FsmParticipant 型のコンポーネントと, 複合型である FsmComposite 型のコンポーネントがある.

Multi Mode 幾つかのモードを持ち, モード毎に複数のロジックを切り替えて使用するタイプのコンポーネント.

上記の 3 種類の基本型, 2 種類の複合型はそれぞれ非排他的で, 一つのコンポーネントが複数の型を継承することができる.

3.4 Introspection

Introspection とは, 一般に「内省」と訳されるが, ここでは自分自身のメタ情報を取得することができる, Java 言語における reflection に似た機能を指す. RT コンポーネントは Introspection の機能により, 自身が持つメタ情報へのアクセスを可能にする. OMG RTC Specification においては, RT コンポーネント特有のメタ情報を取得するためのインターフェースを RTObject として定義するとともに, OMG の超分散オブジェクト (SDO: Super Distributed Object) 標準仕様 [6] の機能を継承することにより RT コンポーネントに Introspection 機能を持たせている. SDO は内部パラメータの設定・取得を行うための Configuration インターフェースを持ち, RT コンポーネントでは内部パラメータへのアクセスにこのインターフェースを利用することができる.

4. OpenRTM-aist-0.4.0

OMG RTC Specification では, 上述の PIM 及び, CORBA PSM, CCM (CORBA Component Model) PSM, Local PSM が定義されている [7]. これらの PSM のうち, CORBA PSM に準拠した仕様に基づき種々の機能追加した RT ミドルウエア OpenRTM-aist-0.4.0 を実装した. 本節では, OpenRTM-aist-0.4.0 の新機能について述べる.

4.1 RT コンポーネント

OpenRTM-aist-0.4.0 では, RT コンポーネントは上述の Introspection パッケージを含む全仕様を網羅したコンポーネ

ントとして実装した.

3.2 節でも述べたように状態遷移が図 2 のように変更され, さらに 3.3 節で示したように, コンポーネント型が複数あるため, コンポーネント開発者は, 作成するコンポーネントにより適切な型を選択する必要がある.

4.2 データポート

OMG RTC Specification における Port はコンポーネント間の相互作用の起点として最低限のインターフェースのみ定義されており, 実装者が具体的な相互作用の方法を定義し実装する必要がある.

データポートは, コンポーネント間のデータ指向通信をサポートするポートであり, データの出力を行う OutPort およびデータの入力を行う InPort に分けられる. コンポーネントで生成されたデータを OutPort から出力し, 他のコンポーネントに送信したり, 他のコンポーネントから受信したデータを InPort から読み込むことができる.

データポートは接続時に ConnectorProfile と呼ばれる接続に関するパラメータを与えることで, データの送受信方法やタイミングなどを動的に変更することができる.

ConnectorProfile には,

- Interface Type
- Data Flow Type
- Subscription Type

と呼ばれる, 3 種類のデータ通信方式のパラメータが存在する.

4.2.1 Interface Type

Interface Type は InPort および OutPort のインターフェースのタイプを指定するパラメータである. OpenRTM-aist-0.2.0 では, CORBA で実装された InPort/OutPort が提供され, CORBA を用いず TCP Socket を直接用いた Raw TCP InPort/OutPort も提案されている [8].

このように, 様々な方法を用いたデータ通信に柔軟に対応させるため, 接続時のパラメータにより通信方式を動的に切り替えられる仕組みが用意されている.

4.2.2 Data Flow Type

Data Flow Type はデータの送受信タイミングを InPort 側で制御するか, OutPort 側で制御するかを選択するためのパラメータである. 従来の InPort/OutPort は基本的には publisher/subscriber モデルに基づく push 型データ通信方式を採用していたが, このパラメータにより, 接続時に push 型通信か pull 型通信かを選択できる.

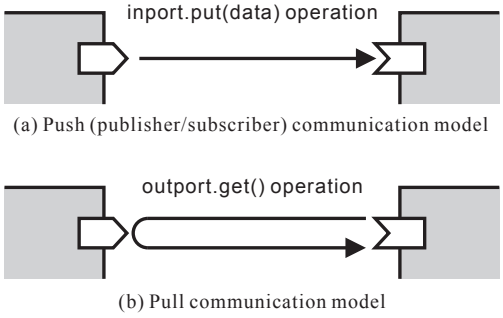


Fig.3 DataPort's data flow model in OpenRTM-aist.

4.2.3 Subscription Type

Subscription Type は OutPort から InPort にデータを送信するタイミングを制御するパラメータである. 従って, これは push 型通信を選択したときのみ有効なパラメータである.

図 4 に示すように, "new", "periodic", "flush" の 3 種類の Subscription Type から選択できる.

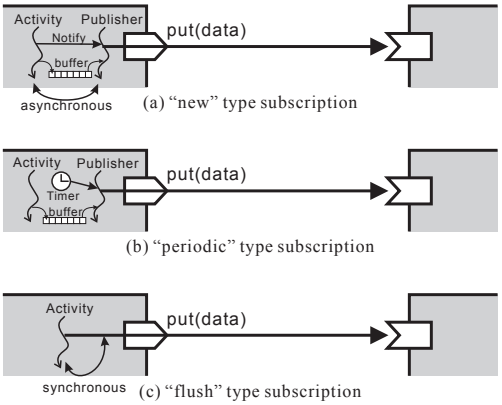


Fig.4 DataPort's subscription model in OpenRTM-aist.

図 4(a) の "new" 型は Activity とは別に Pubslisher がスレッドを持ち, Activity がデータを生成しバッファに書き込んだことを Publisher スレッドに通知し, Publisher スレッドができるだけ早く InPort に対して push する方式である. データの生成とデータの送信は非同期的であり, データの生成回数と送信回数が必ずしも同じにはならないが, Activity の動作を妨げることなく, データの送信を行うことができる.

図 4(b) の "periodic" 型は Pubslisher スレッドがある特定の周期でバッファからデータを読み出し InPort に push する方法である. 送信周期は接続時にパラメータとして指定される.

図 4(c) の "flush" 型においては Publisher はスレッドを持たず, Activity のスレッドが同期的に InPort に対してデータを送信する方式である. この方式は, Activity でのデータが生成と送信が同期的に行われるため, 全てのデータが必ず InPort に送信される反面, 送信処理により Activity がブロックされるため, リアルタイム処理を行う Activity を持つ場合には注意が必要となる.

4.3 サービスポート

サービスポート [9] とは, コンポーネント内部の詳細機能にアクセスするためのインターフェースを外部に公開するための枠組みである. サービスポートはサービスプロバイダポート (Service Provider Port) とサービスコンシューマポート (Service Consumer Port) に分類される. 機能を外部に公開するためのインターフェースをサービスプロバイダ, 外部の機能を要求するインターフェースをサービスコンシューマと呼ぶ.

例えば, コンポーネント内のロジックのパラメータ変更やモード変更, データポートで行うことが困難な複雑なデータのやり取り等は, コンポーネントにそのモジュール独自のインターフェースをサービスポートに付加することで, 内部の詳細機能にアクセス手段を提供する.

ポート間の接続が行われると, 自動的に対応するプロバイダとコンシューマが接続され, コンシューマ側でサービスを利用することができるようになる.

以下に, サービスポートの実装および使用例を示す.

4.3.1 サービスポートの実装

サービスを公開する場合, そのインターフェースは CORBA IDL で記述される. 例えば, 以下に簡単な制御インターフェース例を示す.

```
interface MyRobot
{ // ゲインをセットする
  void setPosCtrlGain(in int axis,
                      in double gain);
  // ゲインを取得する
  double getPosCtrlGain(in int axis);
};
```

上記のようにインターフェースを定義し, ソースの雛形生成ツール rtc-template にこの IDL を与えて, プロバイダ側, コンシューマ側のソースコードを生成する.

プロバイダ側では, このサービスの実装テンプレートファイルも同時に生成される. コンポーネント開発者は, 以下の例のように実装ファイルに機能を実装する.

```
class MyRobot_impl
{ /* この例では m_robo はロボットを実際に制御する
   * クラスのインスタンスであると仮定する.
   */
  void setPosCtrlGain(const int axis,
                      const double gain)
  { // 位置制御ゲインを設定
    m_robo.set_pos_ctrl_gain(axis,gain);
  }
  /* 中略 */
};
```

4.3.2 サービスプロバイダを持つコンポーネント

サービスプロバイダを持つコンポーネントの実装は, 以下のように行う. サービスを公開するポートおよびサービス自身のインスタンスを宣言し, コンストラクタで, サービスをポートに登録し, そのポートをコンポーネントに登録する.

```
class MyRoboComponent
{
 private:
  // MyRobot サービスのポートを宣言
  RTC::CorbaPort m_port;
  // MyRobot サービスのインスタンスを宣言
  MyRobot_impl m_robot;
 public:
  ManipulatorComponent(RTC::Manager manager)
  { // ポートにサービスを登録
    m_port.registerProvider("Robo0","MyRobot", m_robot);
    // ポートをコンポーネントに登録
    registerPort(m_port);
  }
```

4.3.3 サービスコンシューマを持つコンポーネント

サービスを要求するポートおよびサービスコンシューマ自身のインスタンスを宣言し, コンストラクタでサービスコンシューマをポートに登録し, そのポートをコンポーネントに登録する.

```
class MyRobotUser
{
 private:
  // マニピュレータサービスのポートを宣言
  RTC::CorbaPort m_port;
  // サービスコンシューマのインスタンスを宣言
  RTC::CorbaConsumer<MyRobot> m_robot;
 public:
  any_functions()
  { // サービスの利用例
    // ゲインをセット
    m_robot->set_pos_ctrl_gain(0,1.0);
    // ゲインを表示
    std::cout << m_robot->get_pos_ctrl_gain(i) << std::endl;
    }
  }
```

以上により, 図 5 に示す Robot サービスのプロバイダおよびコンシューマを持つコンポーネントを作成することができる.

これらのポートに対して接続操作を行うと, プロバイダとコンシューマが結合され, リモート呼び出しにより, コンシューマはプロバイダを利用することができる.

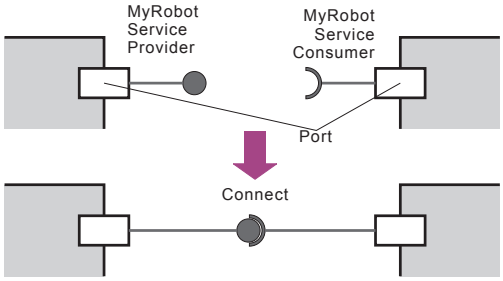


Fig.5 RT Service Provider Port and Service Consumer Port

4.4 ミドルウエア, ツール

この他, 実装や開発をサポートするミドルウエア部分や周辺ツールも, 上述の機能拡張に対応するため変更されたが, 本稿では標準化された RT コンポーネントモデルに関連する部分の拡張についてのみ述べるに留める.

5. 終わりに

本稿では, 国際的なソフトウエア標準化団体 OMG において標準化された RT コンポーネント仕様 (OMG RTC Specification) の PIM (Platform Independent Model) を中心に概説した. また, この OMG RTC Specification に準拠した新しいミドルウエア OpenRTM-aist-0.4.0 の新しい機能を紹介した. 本稿で示した OMG RTC Specification は最終文書化の途中であるため, 本実装で見つかった仕様上の不具合などにより今後変更される可能性がある. 仕様の最終文書化後 OMG において採択され, 標準仕様が公式リリースされた後に OMG 公式標準仕様に準拠した OpenRTM-aist-1.0.0 を実装, リリースする予定である.

参考文献

[1] 末廣 他, "RT コンポーネントの実装例.RT ミドルウエアの基本機能に関する研究開発 (その 1)", 第 21 回 日本ロボット学会学術講演会予稿集, p.1F27, 2003.09

[2] Noriaki ANDO et al., "RT-Middleware: Distributed Component Middleware for RT (Robot Technology)", 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2005), pp.3555-3560, 2005

[3] OpenRTM-aist-0.2.0 Tutorial Web page, http://www.is.aist.go.jp/rt/OpenRTM-aist-Tutorial/

[4] 池添 他, ".NET Framework を利用した RT ミドルウェア : OpenRTM.NET", 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, p.3B1-2, 2006.12

[5] 神徳 他, "RT ミドルウェア標準化活動への誘い", 日本機械学会 ロボティクス・メカトロニクス講演会 2005, p.2P1-N-066, 2005.6

[6] OMG final adopted specification, "Platform Independent Model(PIM) and Platform Specific Model(PSM) for Super Distributed Object(SDO) Specification", http://www.omg.org/cgi-bin/doc?formal/2004-11-01

[7] OMG draft adopted specification, "Robotic Technology Component (RTC) final adopted specification", http://www.omg.org/cgi-bin/doc?ptc/2006-11-07

[8] 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 -", 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

[9] 安藤 他, "RT コンポーネント・サービスフレームワーク-RT ミドルウェアの基本機能に関する研究開発 (その 15)- ", 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 2005 (SI2005), p.3C1-2, 2005.12