OpenRTM-aist を用いたリアルタイムシステムの構築手法

○金広文男　安藤慶昭　中岡慎一郎　原功 (産総研)

1. はじめに

本稿では OpenRTM-aist[1] をミドルウェアとして用いたリアルタイムシステムの構築手法について述べる．OpenRTM-aist は分散オブジェクト技術の一つである CORBA を用いて実装されており，ハードリアルタイムシステムの構築には不向きであるとみなされている場合もあるが，OpenRTM-aist の持つ拡張性，システム構築の柔軟性をうまく利用することで，数 [ms] 程度の周期で実行されるリアルタイム部分と非リアルタイム部分とが混在したシステムを一つのミドルウェア上に構築することが可能である．実際にヒューマノイドロボット HRP-2[2] や HRP-4[3] のリアルタイムシステムは OpenRTM-aist を用いて構築されている．

2. RT コンポーネントのリアルタイムタスク化

RT コンポーネントの開発を行う場合，ユーザが主に実装を行うのは `onExecute()` 等のコールバック関数である．これらのコールバック関数は RT コンポーネントが持つ状態遷移マシンの状態に応じて呼び出される仕組みとなっている．この状態遷移マシンを駆動しているのが実行コンテキストと呼ばれる部分であり，通常の RT コンポーネントでは非リアルタイムの周期実行を行う実行コンテキストが使用されている．これが周期的に RT コンポーネントの状態をチェックし，対応するコールバック関数の呼び出しを行っている．従って RT コンポーネントの動作をリアルタイム化するには，この実行コンテキストをリアルタイム化する必要がある．

リアルタイム OS である ART-Linux[4] 及び Preemptive Linux[5] 用のリアルタイム実行コンテキストは既に OpenRTM-aist に含まれており，設定ファイルである rtc.conf を用いてこれらの使用を指定するだけで RT コンポーネントのリアルタイム実行が可能となる環境が整っている．例えば，ART-Linux を用いてリアルタイム実行する場合には，rtc.conf を以下のように設定するだけで，ユーザが実装したコールバック関数の変更は不要である（無論コールバック関数がリアルタイム実行を妨げるような内容になっていないことが前提である）．

```
manager.modules.load_path: /usr/local/lib
manager.modules.preload: ArtExecutionContext.so
exec_cxt.periodic.type: ArtExecutionContext
```

ここでは ART-Linux 用のリアルタイム実行コンテキストが実装されているシェアードオブジェクト，ArtExecutionContext.so をダイナミックロードし，周期実行コンテキストを生成する場合には ArtExecutionContext を生成するように設定を行っている．

3. 連携する複数の RT コンポーネントからなるリアルタイムシステムの構築方法

前節で述べた方法により，全く独立にリアルタイム動作する複数の RT コンポーネントを実行することはできるが，複数の RT コンポーネントを連携させながらリアルタイム実行する為には，それに適した設定を行う必要がある．

本節では A と B という 2 つの RT コンポーネントがあり，各実行周期においてまず A が実行されて計算結果をデータポートから出力し，次に B が実行されてデータポートから受け取った A の計算結果を用いて計算を行う，という場合を例題として考える．

3·1 複数の RT コンポーネントの同期実行

RT コンポーネントが生成される際にはその RT コンポーネントに紐付けされた実行コンテキストも同時に生成される．複数の RT コンポーネントを実行した場合それらは非同期に周期実行を行うこととなる．各実行周期において複数のコンポーネントを順に実行するには，RT コンポーネントを個々の実行コンテキストで駆動するのではなく，1 つの実行コンテキストによって複数の RT コンポーネントを順に駆動するように設定する必要がある．この設定を行うには，実行コンテキストの以下のインタフェースを用いる．

```
// RTC.idl
ReturnCode_t stop();
ReturnCode_t
      add_component(in LightweightRTObject comp);
```

本節で想定するシステムでは 1 つの実行コンテキストで 2 つの RT コンポーネントを駆動するため，2 つの内 1 つの実行コンテキストは不要である．そこでまず B の実行コンテキストに対して `stop()` を呼び出して実行コンテキストを停止する．次に A の実行コンテキストに対して `add_component()` を呼び出して，B も駆動対象として登録する．

以上の手続きにより複数の RT コンポーネントの同期実行が可能となる．

3·2 rtcd の利用

ここまでの設定では，同期実行は出来るもののリアルタイム実行はできない．これは，別プロセスとして起動されている RT コンポーネント間の通信がリアルタイム実行を阻害する為である．これを防ぐためには，2 つのコンポーネントを同一プロセス上に生成することが必要となる．これは本来はネットワーク通信によって行われる CORBA インタフェースの呼び出しが，呼び出し元と呼び出し先が同一プロセス上にある場合には，単なる関数呼び出しに置き換えられるという機能を利用するためである（ただしこの関数呼び出しへの

置き換えは CORBA 仕様によって規定されているものではなく，CORBA の実装依存であるため注意が必要である）．

複数の RT コンポーネントを同一プロセス上に生成するには OpenRTM-aist に含まれるユーティリティである rtcd を用いる．rtcd を用いて複数の RT コンポーネントを同一プロセス上に生成するには，rtc.conf を用いる方法とマネージャの CORBA インタフェースを用いる方法がある．rtc.conf を用いる場合には以下のように記述する．

```
manager.modules.load_path: /usr/local/lib
manager.modules.preload: compA.so, compB.so
manager.components.precreate: compA, compB
```

ここでは RT コンポーネントを実装したシェアードオブジェクト 2 つをダイナミックロードし，それぞれについてインスタンスを 1 つ生成する事を指示している．この方法では予め rtc.conf に記述した RT コンポーネントしか生成することができないため，より動的に RT コンポーネントを生成するには，以下のマネージャの CORBA インタフェースを利用する．

```
// Manager.idl
RTC::ReturnCode_t load_module(in string pathname,
                              in string initfunc);
RTC::RTObject create_component(in string compname);
```

これらのインタフェースを外部の CORBA クライアントから呼び出すことで，必要なシェアードオブジェクトを動的にロードし，インスタンスを生成することができる．

この rtcd を用いて同一プロセス内に複数の RT コンポーネントを生成し，同一の実行コンテキストで同期的に駆動する手法はリアルタイムシステムのみならず，非リアルタイムシステムにおいても有用である．例えば，画像をキャプチャし，様々なフィルタ処理を連続して適用する画像処理アプリケーションにおいては，キャプチャコンポーネントとフィルタコンポーネントを同一プロセス上に生成し，逐次的に実行することで，プロセス間通信を行わずにすみ，実行周期のずれによる処理の遅れを無くすことが可能となる．

3·3 サブスクリプションタイプの設定

データポート間で Push 型通信を行う場合には，データの送信方法に応じて異なるサブスクリプションタイプ [6] を選択することができる．サブスクリプションタイプには New, Periodic, Flush があり，New と Periodic では実行コンテキストに呼び出されたコールバック関数はデータポートの送信バッファへの書き込みだけを行い，実際の送信処理は別のスレッドが実行する．Flush では実行コンテキストが実際の送信処理までを行う．New や Periodic は，スレッドが介在することになるため，送信したデータがいつ実際に受信側に到着したかを知ることはできない．本節で考えているシステムでは B が実行される際には A が送信したデーが到着している必要があるため，サブスクリプションタイプとして Flush を指定することが必要となる．

逆にリアルタイム実行されている RT コンポーネントから，別プロセスとして動作している RT コンポーネントへデータを送信する場合には，通信処理によってリアルタイム性が失われることを防ぐため，Flush は指定してはならないことに注意が必要である．

なおリアルタイムシステム側が受信側となる場合は，受信処理は受信スレッドが行うために，リアルタイム実行が妨げられることはない．

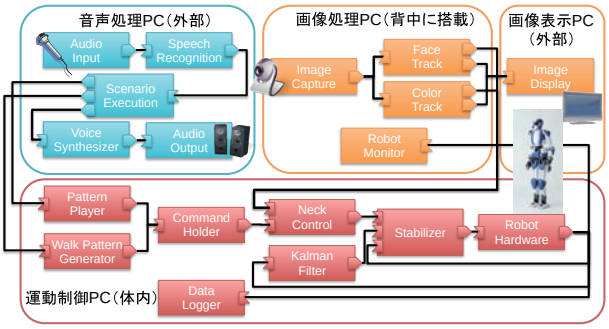

図 1 HRP-4 のシステム構成

4. システム構築例：HRP-4

2010 年 9 月に行った HRP-4 のデモンストレーション [7] では，音声認識による動作制御及び画像処理による動作制御を行った．この時のシステム構成を図 1 に示す．全体で 18 個の RT コンポーネントからなるシステムであり，それらは体内に搭載した 2 台のコンピュータ及び外部の 2 台のコンピュータに分散して実行される．システムは大きく分けて，運動制御，音声処理，画像処理の 3 つの部分から成っている．運動制御部分は周期 5[ms] のリアルタイムタスクとして実行されており，8 個の RT コンポーネントが 1 つの実行コンテキストによって逐次実行される．音声処理部分は OpenHRI[8] の RT コンポーネントを組み合わせて体外の計算機で実行される．画像処理部分は OpenCV を用いて実装されており，非実時間実行ではあるが，画像キャプチャと画像処理とを同一プロセス上で同期実行している．

OpenRTM-aist をミドルウェアとして用いる場合，RT コンポーネント間のデータのコピー及びマーシャリング／アンマーシャリングがオーバーヘッドとして処理時間に加算されることとなる．データを参照渡しで交換していた HRP-2 の旧システムと比較してオーバーヘッドは 1 周期当たり 100[us] から 200[us] 程度であった．実行周期が 5[ms] である HRP-4 の場合にはこれは許容可能な範囲である．

5. まとめ

本稿では OpenRTM-aist を用いたリアルタイムシステムの構築手法について述べた．(1) リアルタイム実行コンテキストの使用，(2)rtcd を用いた同一プロセス上への複数の RT コンポーネントの生成，(3) 適切なサブスクリプションタイプの設定等を行うことで OpenRTM-aist を用いたリアルタイムシステムの構築が可能である．

謝辞

本研究の一部は経済産業省・NEDO の「次世代ロボット知能化技術開発プロジェクト」の支援を受け行われた．また HRP-4 のデモンストレーション作成には松坂

要佐氏，森澤光晴氏，辻徳生氏にご協力頂いた．ここに謝意を表す．

参考文献

[1] 安藤慶昭． OpenRTM-aist-1.0 の新機能． 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), pp. 2L1–3, 2008.
[2] 五十棲隆勝, 赤地一彦, 平田勝, 金子健二, 梶田秀司, 比留川博久. ヒューマノイドロボット HRP-2 の開発. 日本ロボット学会誌, Vol. 22, No. 8, pp. 1004–1012, 2004.
[3] 赤地一彦, 宮森剛, 林篤史, 金平徳之, 金子健二, 金広文男, 森澤光晴. 働く人間型ロボット研究開発用プラットフォーム「HRP-4」の開発. 第 11 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, 2010.
[4] ART-Linux. http://www.dh.aist.go.jp/jp/research/humanoid/ART-Linux/.
[5] RTwiki. https://rt.wiki.kernel.org/index.php/Main_Page.
[6] 安藤慶昭, 清水昌幸, 神徳徹雄. 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, pp. 1P1–E08, 2008.
[7] 働く人間型ロボット研究開発用プラットフォーム HRP-4 を開発． http://www.aist.go.jp/aist_j/press_release/pr2010/pr20100915/pr20100915.html.
[8] OpenHRI. http://openhri.net/.