第10回
ロボティクスシンポジア
Robotics Symposia

2005.3.14-15 in HAKONE

RSJ JSME SICE

【共同主催】
日本ロボット学会
日本機械学会ロボティクスメカトロニクス部門
計測自動制御学会システムインテグレーション部門

リアルタイム制御のための複合 RT コンポーネントフレームワークの実装 –RT ミドルウェアの基本機能に関する研究開発 (その 11)–

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

Development of RT Composite Component Framework for Real-Time Control – R & D of RT Middleware Fundamental Functions (Part 11) –

*Noriaki ANDO (AIST), Takashi SUEHIRO (AIST), Kosei KITAGAKI (AIST),
Tetsuo KOTOKU (AIST) and Woo-Keun YOON (AIST)

Abstract— We have developed a framework of RT-component which promotes application of Robot Technology (RT) in various field. In this paper, we will discuss robotic system development methodology and our RT-middleware concepts. The system development methodology using RTComponent, and new framework to make composite component for RT-component will be shown. A evaluation of real-time composite component will be derived. Finally conclusion and future work will be described.

Key Words: RT(Robot Technology), software component, middleware, robot system, composite component

1. はじめに

近年、ロボット工学は個々の機能の研究とともに、これまでロボット工学が積み上げてきた様々な機能をいかに組み合わせシステムを構築するかといった、インテグレーション手法もまた重要となりつつある。また、社会的要請からロボット機能要素 (RT: RobotTechnology[1]) をロボットのみならず、実生活空間の知能化やユビキタス化に適用し、本格的応用を模索する動きも活発化してきた。こういった背景から、図 1 に示すように、属人性を排除したシステムインテグレーションを体系的に実践する知識と、同時にこれをサポートするプラットフォームの必要性が高まってきた。

著者らは、RT のソフトウエアモジュール化を実現し、システムインテグレーションを効率的に行うためのソフトウエアプラットフォーム「RT ミドルウエア」の基本機能に関する研究開発を行ってきた。

これまで RT ミドルウエアのソフトウエア部品:RT コンポーネントのアーキテクチャを提案し [2, 3, 4, 6, 7]、RT コンポーネントを用いて実際にいくつかのシステムを構築しその有用性を示した [5]。

本稿では、RT ミドルウエアに新たに加わった、RT コンポーネントを密に連携させる機能「複合コンポーネント」について述べる。はじめにロボットシステムを体系的に構成するための方法論について議論する。さらに、こうしたシステム構築を実装レベルでサポートするための、プラットフォーム「RT ミドルウエア」およびそのソフトウエア部品である「RT コンポーネント」の必要性について述べる。また、RT コンポーネントのアーキテクチャ・実際の作成方法、作成された複数の RT コンポーネントを組み合わせ、実際のシステムを構築する際に必要な機能について議論を行う。最後に、実装された複合コンポーネントフレームワークの評価実験を行いその有用性を示す。

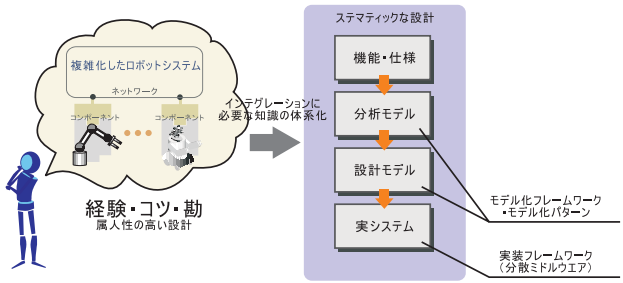

Fig.1 勘に頼る設計からシステマティックな設計へ

2. コンポーネント指向開発

ロボット研究はこれまで、機械機能の解析 (アナリシス) および比較的単純な機能の実現が重点的に行われてきた。その一方で、解析の結果やそこから得られた機能を実世界に適用するには、これらを再統合 (シンセシス) する必要がある。現在のロボット工学にはこのシンセシスの知識が比較的少なく、その知識も体系化されているとはいい難い。

ロボットシステムの開発フローは、ソフトウエアシステム開発のそれを参考にするならば図 2 のように考えることができる。

研究者や開発者が経験と勘によって行ってきた、ロボットシステムを構築するために必要な知識を大別すれば、システムを分析するための知識、設計に関する知識、実装に関する知識に分けることができる。

さらに、分析はシステムが要求する仕様を実際の技術レベルに即した形で分析を行うための知識 (分析パターン) と枠組み (分析モデルフレームワーク) に分けられる。また、設計や実装に関しても設計をより現実的なレベルに落とし込む知識 (設計パターン) および枠組み (設計フレームワーク) から成る。

図 2 に示すように、実装に近いものほど抽象度が低く、一般に寿命が短い。したがって、実装レベルに近いフレームワークだけでは技術が陳腐化すればすぐに使えなくなる可能性がある。逆に、より一般化されたパ

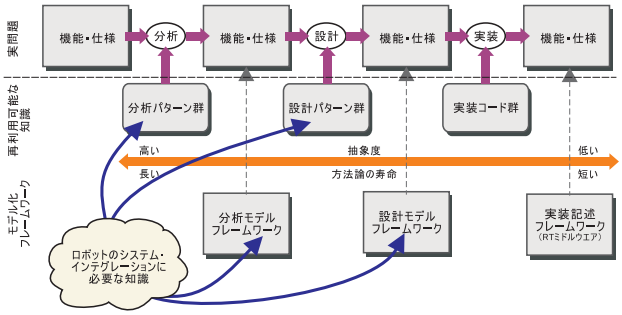

Fig.2 RT システム設計

ターンやモデル化フレームワークを獲得することができれば、技術的進歩により実装技術が変化しても、すぐに新しい技術を用いたロボットシステム開発を行うことができる。

我々は、RT ミドルウエアが提供するフレームワークを用いることで、効率的により抽象度の高いパターンやモデル化フレームワークを獲得することを目指している。次節以降では、RT コンポーネントおよびコンポーネントを用いたシステム開発について述べる。

3. RT コンポーネント

3·1 RTComponent アーキテクチャ

図 3 に RT コンポーネントのアーキテクチャ・ブロック図を示す。RT コンポーネントはネットワーク上に分散されたコンポーネントへの透過的アクセスを実現するために、分散オブジェクト技術を用いて実装される。現在開発中の RT ミドルウエアは言語、OS 非依存なプラットフォームを目指しており、CORBA を用いて実装が進められている。

通常の分散オブジェクトに対する RT コンポーネントの特長は以下のとおりである。

- コンポーネント自体が常に動作し続ける「アクティビティ (Activity) 」を持つ。
- 入出力の相互接続互換性を保証する共通インターフェース (InPort/OutPort) を持つ。
- アクティビティはコンポーネント間の互換性を保証する共通の状態遷移を持つ。

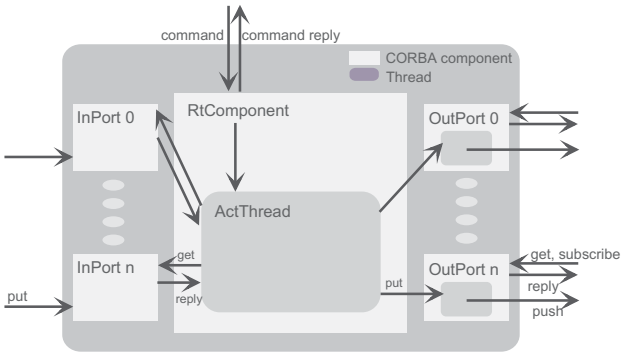

Fig.3 RT コンポーネント

通常の分散オブジェクトでは、オペレーションに対して何らかの処理を行い、その結果を返すといった動作が一般的である。RT コンポーネントはコンポーネント自体が常に動作し続けるアクティビティを持ち、これがロボット等デバイス制御の主体となる。

一方、ロボットの制御においては、機能ブロックをデータを「受け取り」、「処理」し、結果を「出力する」機能単位とするとシステム構成が行いやすい場合が多い。RT コンポーネントはデータの受け渡しを行う共通インターフェースとして InPort/OutPort とよばれる入出力ポートを持つ。

また、様々な機能を持つコンポーネントでも、それらが共通のソフトウエア部品として扱うことが出来るように、コンポーネントとしての状態遷移を規定した。その規定に従ってコンポーネントを構成することにより、様々な粒度のコンポーネントでも同様に扱うことが出来るようになっている。また、この状態遷移を規定することにより、コンポーネントを入れ子にした複合コンポーネントを実現することが可能となる。

3·2 状態遷移

さまざまなコンポーネントデベロッパによって作成された多数のコンポーネントを統一的に扱うには、コンポーネントのライフサイクルにおいて共通の状態遷移を持つ必要がある。共通の状態遷移を持たせ、その意味を予め規定しておくことにより、多数のコンポーネントの挙動を統一的に制御することが可能となる。

RT コンポーネントのアクティビティは、`BORN`, `INITIALIZE`, `READY`, `STARTING`, `ACTIVE`, `STOPPING`, `ABORTING`, `ERROR`, `EXITING`, `FATALERROR`, `UNKNOWN` の 10 の状態を持つ。図 4 にアクティビティ部の状態遷移図 (UML ステートチャート) を示す。各ステートにはその状態で行うべき処理のメソッド名が記述されている。UML ステートチャート記述法に従い、

- entry: 状態に入るときに実行される処理、
- do: 状態にいる間に実行される処理、
- exit: 状態から出るときに実行される処理、

として記述されている。entry 処理のみ持つ状態はすぐに次の状態に遷移する過渡状態を表しており、do 処理を持つ状態はその状態に留まることができる定常状態を表している。

表 1 に各状態の意味を示す。

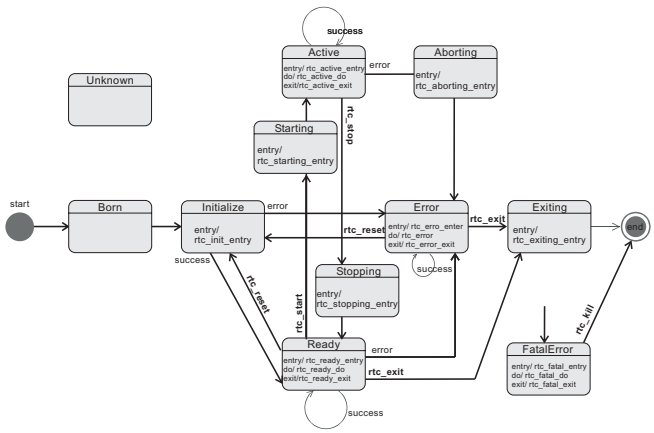

Fig.4 アクティビティの状態遷移図

コンポーネント開発者は自分が実装したい対象についてこれらの状態を当てはめ、それぞれの状態での処理をコンポーネントフレームワークに記述すればよい。アクティビティは、自らまたは外部からのオペレーションにより状態遷移を行う。

全てのコンポーネントに同じ状態遷移を持たせることにより、後述する複数のコンポーネントを結合させ

Table 1 アクティビティの状態

BORN	コンポーネント生成状態。コンポーネントのインスタンスのを生成している状態。外部からオペレーションを実行することはできない状態。
INITIALIZE	初期化状態。ここでコンポーネントの各種初期化を行う。たとえば、デバイスコンポーネントであれば初期化は INITIALIZE 状態で行う。
READY	待機状態。この状態ではすぐにアクティブ状態に移行可能であるが実際には処理を行わない。アクチュエータ等をコンポーネント化する例を考えた場合、サーボ ON の状態かつ安全に停止している状態を READY 状態に割り当てることもできる。
STARTING	ACTIVE 状態に入る前の過渡状態。ACTIVE 状態になるために必要な処理をここに割り当てることができる。
ACTIVE	活性状態。主たる処理を行う状態であり、コンポーネントは他のコンポーネントからのデータを受け取り、処理しあるいは出力することができる。
STOPPING	ACTIVE 状態から READY 状態へと遷移する時の過渡状態。
ABORTING	ACTIVE 状態で何らかのエラーが発生した場合に遷移する過渡状態。この状態を経た後 ERROR 状態へと遷移することが保証されている。
ERROR	エラー状態。何らかのエラーを検出するとこの状態に遷移する。ACTIVE 状態以外の定常状態では、エラーが発生した際に直接この状態へと遷移する。他のコンポーネントとの通信は停止される。再初期化を行うと、INITIALIZE 状態を経て再び READY 状態に戻ることができる。
EXITING	終了状態。適切な終了処理が行われた後にコンポーネントは停止する。ハードウエアを扱うコンポーネント等ではデバイスを安全に停止させリソースを開放等の処理を割り当てることができる。
FATALERROR	何らかの致命的エラーが発生した際に遷移する状態。FATALERROR に遷移したコンポーネントは再初期化を行うことはできない。
UNKNOWN	上記以外の状態。通常決してこの状態になることはない。

同期的に動作させる「複合コンポーネント」(Composit Component) を構成するのが容易になる。すなわち、複数の密接に関係し同期的に動作するコンポーネントの塊をひとつのコンポーネントとして捉えることができるようになる。この複合コンポーネントは、多層に階層化された複雑なシステムや、複数のコンポーネントをリアルタイム動作させる際に有効である。

4. RT コンポーネント開発

RT ミドルウエアは、コンポーネントの開発者 (=コンポーネントデベロッパ) が持つ既存のソフトウエア資産、あるいは新たに作成したソフトエアを容易にコンポーネント化し、分散システムを構築するためのフレームワークおよびコンポーネント運用するためのサービス群を提供するミドルウエアである [4, 6, 7]。

図 5 に RT コンポーネント作成の流れを示す。

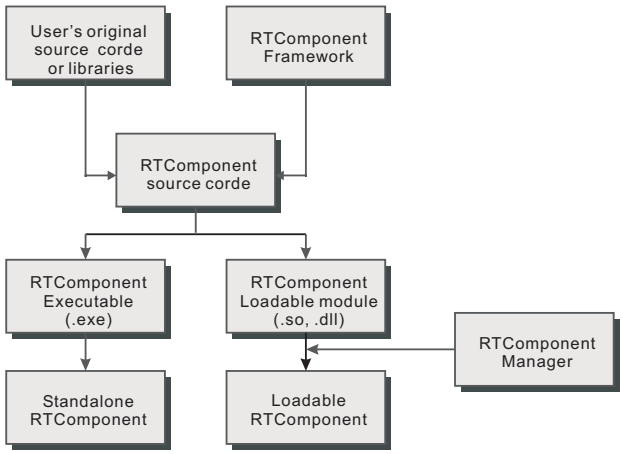

Fig.5 RT コンポーネントの開発フロー

コンポーネントデベロッパは、既存のソフトウエア資産をコンポーネントフレームワークに埋め込むことで、ソフトウエア部品として再利用可能な RT コンポーネントを容易に作成することができる。

一旦コンポーネント・インターフェース等の仕様が確定し、デバッグされ安定に動作するコンポーネントが完成すれば、そのコンポーネントを利用するユーザは、そのコンポーネントのソースコードに手を入れることなしに、他のコンポーネントと組み合わせ、ロボットシステムを構築することができる。デベロッパはソースコード、ローダブルモジュール形式、実行ファイル形式の任意の形式で配布することができ、さまざまな形で再利用することができる。

4·1 RT コンポーネント同士の連携

ソフトウエア部品として作成された RT コンポーネント同士を連携させる方法には、アプリケーションの階層の高低に応じて、また用途によって様々なものが考えられる。

コンポーネントの利用者は、複数のコンポーネントを部品として組み合わせることによりシステムを構築する。RT ミドルウエアにおいてはコンポーネントを連携させる方法は、図 7 に示すように、

- GUI ツール
- XML により記述されたアセンブル情報 (実装中)
- スクリプト言語
- 一般的なプログラム
- 他の RT コンポーネント

等がある。なお、XML によりアセンブル情報の記述については実装の途中段階である。

4·1.1 GUI によるアセンブル

アセンブリ GUI は、制御ブロック線図のように、コンポーネントの InPort/OutPort の接続や、コンポーネントの活性化/非活性化を GUI 画面上から操作することができるインターフェースであり、ユーザは視覚的にコンポーネントを組み合わせることができる。この接続情報は、XML 形式で保存され、この XML ファイルを用いることで以前の接続情報を再現することができる。

4·1.2 スクリプト言語によるアセンブル

コンポーネントを記述可能なスクリプト言語としては Python が利用可能である。Python のようなスクリ

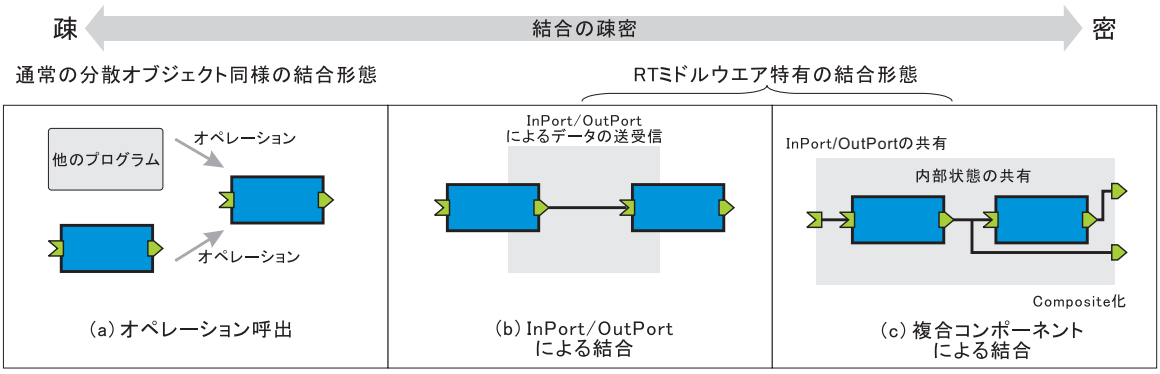

Fig.6 RT コンポーネントの連携方法

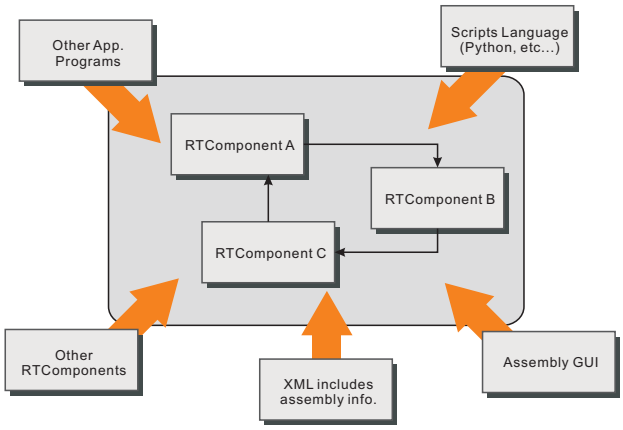

Fig.7 コンポーネントのアセンブル

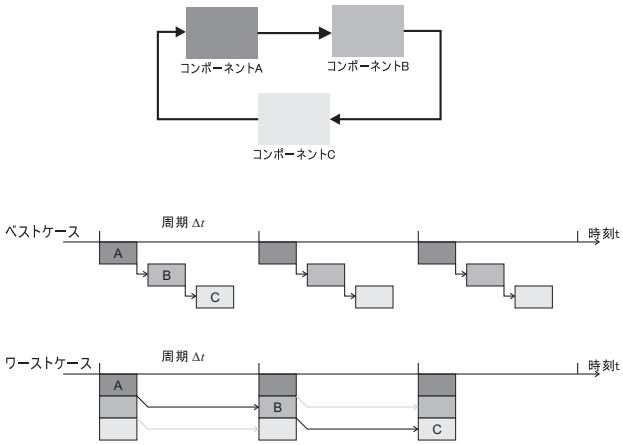

Fig.8 コンポーネントを個別にリアルタイム化した場合

プト言語を用いることにより、コンパイルすることなくラピッドプロトタイピングが行える。また、高速な実行速度を必要としない上位のアプリケーションレベルのシステムを構築する際には、スクリプト言語のような高級言語がより適している。

4·1.3 CORBA オブジェクトとして利用

RT ミドルウエアは分散オブジェクトミドルウエア：CORBA を利用しており、通常の CORBA オブジェクトとして用いれば、アプリケーションプログラムからコンポーネントを制御することができる。

また、コンポーネントの接続や活性化/非活性化は、他のコンポーネントからも操作することができるため、動的な接続の変更などをプログラムから行うことができる。

5. 複数のコンポーネントの協調

これまで、上記のコンポーネントフレームワークに基づいて、我々はいくつかのコンポーネントを実際に作成し、実際のロボットを動作させるなどして検証を行ってきた [5]。ロボットを制御するプログラムにおいては、各種処理をリアルタイム動作させる必要が様々な場面で発生する。我々は、いくつかのコンポーネントのアクティビティを ART-Linux を用いてリアルタイム化し動作させ、単一コンポーネントレベルでのリアルタイム化も容易に可能であることを確認した [5]。しかしながら、コンポーネントの独立性を高めると、逆にコンポーネント間の密な連携を阻害する可能性がある。

図 8 に示す 3 つのコンポーネント (A, B, C) が互いの出力データに依存して動作している場合を考える。コンポーネント A, B, C はそれぞれリアルタイム化され、周期 Δt を守って動作している。図 8 のベストケースの場合、A, B, C からなるシステムは全体として周期 Δt を守って動作することが出来るが、ワーストケースの場合では A がデータを受け取って C が出力するまでに $2\Delta t + t_C$ の時間がかかることになる。なお、t_C はコンポーネント C が入力を受け取って出力するまでの処理時間とする。

n 個のコンポーネントが直列に接続されたシステムを考えると最初のコンポーネントがデータを受け取ってから、最後のコンポーネントがデータを出力するまでに最大で、

$$(n-1)\Delta t + t_n, \tag{1}$$

の時間がかかる。ここで、t_n は n 番目のコンポーネントの処理時間である。

上記の例で、コンポーネント A, B, C がそれぞれ、マニピュレータ、コントローラ、力センサとして、マニピュレータの力制御系を実現したいとする。これらをそれぞれ 1 ms のリアルタイム周期で動作させた場合、力センサで力を計測してからマニピュレータに指令値を出力すると、最悪のケースでは $2 + t_{manipulator}$ ms 遅れで指令値が出力される。また、システムを構成するコンポーネント数が多くなった場合にはさらに遅れが増加し、そうした構成では安定な制御系の設計は望めない。

RT ミドルウエアは、まったく個別に作成されたコンポーネント同士を組み合わせて RT システムを構築することを目指しており、上記のようなシステムを構成したい場合、複数のコンポーネントを同期させてリア

ルタイムで動作させることが出来なければならない。

6. 複合コンポーネント

上述の例のような場合を解決するために、RT ミドルウエアに複合コンポーネントのための新たなフレームワークを導入した。

複合コンポーネントには、大きく分けて「非同期複合コンポーネント」「同期複合コンポーネント」がある。図 9 に示すように、非同期複合コンポーネントは、

- 個々の InPort/OutPort を複合コンポーネントの InPort/OutPort として扱う。
- 個々のアクティビティはそれぞれスレッドを持ち並列に動作する。
- 個々のアクティビティはそれぞれ状態を持ち、すべては必ずしも一致しない。

といった複数のコンポーネントを単にグルーピングするための機能である。一方、同期複合コンポーネントは、

- 個々の InPort/OutPort を複合コンポーネントの InPort/OutPort として扱う。
- 個々のアクティビティは一つのスレッドにより直列に実行される。
- 複合コンポーネントとしての状態を持ち、すべてのコンポーネントの状態が一致する。

といった前節での問題を解決するための枠組みである。

非同期複合コンポーネントは、外部からはひとつのコンポーネントとして扱うことができるが、図 9 に示すよう、各コンポーネントは並列に動作し、これらが複数の CPU 上で動作しているときには CPU 時間を有効に使用することができる。ただし、同期的に複数のコンポーネントをリアルタイム動作させたい場合には前節で指摘したような問題が発生する。

一方、同期複合コンポーネントは、内包した各コンポーネントを予め設定された順序で実行し、各コンポーネントのアクティビティは直列かつ同期的に動作する。同期複合コンポーネントを同一 CPU 上かつリアルタイムプロセス上で構成すると、別々に作成された複数のコンポーネントが、あたかも一つのリアルタイムプロセスのように実行することができる。

7. 評価実験

7·1 呼出オーバーヘッドの計測

ここで、同一プロセス内での同期複合コンポーネントの呼び出しのオーバーヘッド (図 9) を評価する実験を行った。いくつかのローダブルモジュールコンポーネントを複合コンポーネントとして構成した際の、呼び出しにかかるオーバーヘッドを計測した。計算機には Pentium4 2.8GHz, OS にはリアルタイム OS の ART-Linux を用いた。リアルタイム周期実行を行う場合を想定し、1 ms のリアルタイムループ内で各コンポーネントを実行させた。

実験結果を表 2 に示す。コンポーネントは全ての場合において、直列に実行されていることを確認した。表 2 の平均実行周期および標準偏差から、全ての場合において実行周期が守られていることがわかる。さらに、コンポーネントの一周期あたりの総呼出時間から推定された、1 コンポーネントあたりの呼出時間は約 2.6 ～ 4.8 μs であり、その標準偏差からもリアルタイム周期実行を妨げるものではないことが確認できた。

これは、実験に使用した CORBA が同一プロセス内のオペレーションを単なる関数呼び出しとして実行しているためであると思われる。

Table 2 同期複合コンポーネント呼出しオーバーヘッドの評価：内部は何もしない空のコンポーネント

コンポーネント数	1	2	10
平均実行周期 [ms]	1.00	1.00	1.00
標準偏差 [ns]	86.6	66.1	47.3
総呼出時間 [μs]	4.84	7.85	30.4
1 周期当り			
呼出時間 [μs]	4.84	2.62	3.04
標準偏差 [ns]	86.6	121	92.1

この実験から、RT コンポーネントの仕様に従い実装されたコンポーネントは、全く個別に作成されたものであっても、それらを複数組み合わせ、かつリアルタイム周期実行可能であることが確認された。

7·2 マニピュレータ力制御システム

同期複合コンポーネントを実システムのリアルタイム制御に適用した。手先力・トルクセンサ、マニピュレータ、ジョイスティック (力センサ)、コントローラの 4 つのコンポーネントからなるシステムを同期複合コンポーネントにより構成した。複合コンポーネントの内部構成は、図 11 に示すように、コンポーネントの実行順序が、

1. 手先力・トルクセンサ
2. ジョイスティック (力センサ)
3. コントローラ
4. マニピュレータ

となるように構成した。

制御用計算機は Pentium4 2.8GHz, OS はリアルタイム OS である ART-Linux を使用した。リアルタイムループの実行周期はマニピュレータの運動制御ボードの制御周期から 2.00 ms とした。

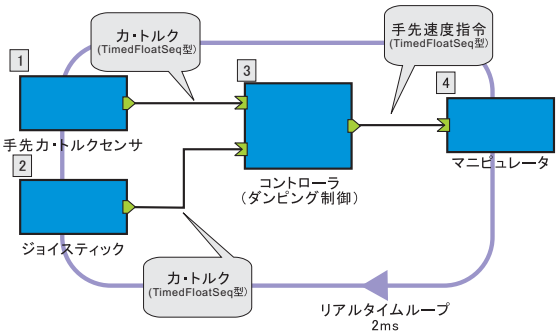

Fig.10 同期複合コンポーネントによるマニピュレータ力制御システムの構成: ブロックの左肩の数字が実行順序をあらわしている。

表 3 に実験時の実行周期の計測結果を示す。実験において、手先力センサに力を加えると力の方向に手先位置が移動する制御が実現されていることが確かめられた。また、ジョイスティックに力を加えても同様にマニピュレータの手先位置が動くことが確かめられた。いずれの場合においても、力制御が安定に動作していることが確認された。

ここで重要な点は、これら 3 つのデバイスと 1 つの制御プログラムが 1 枚岩 (モノリシック) なプログラム

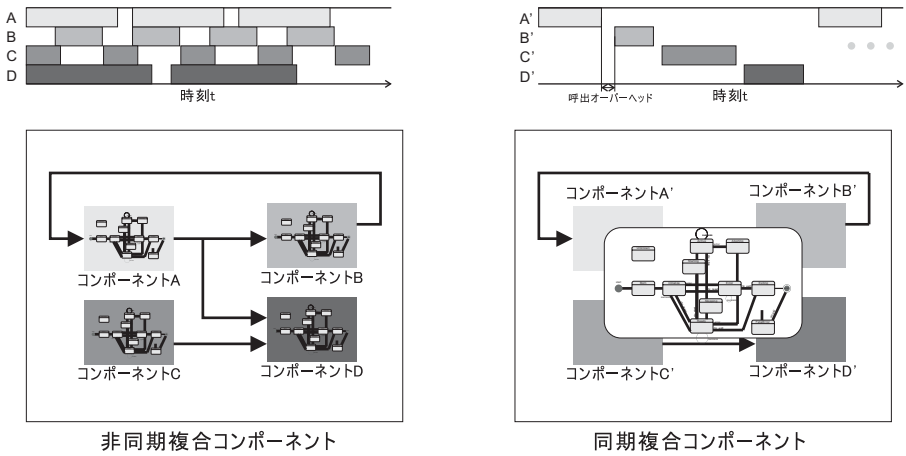

Fig.9 非同期複合コンポーネント/同期複合コンポーネント

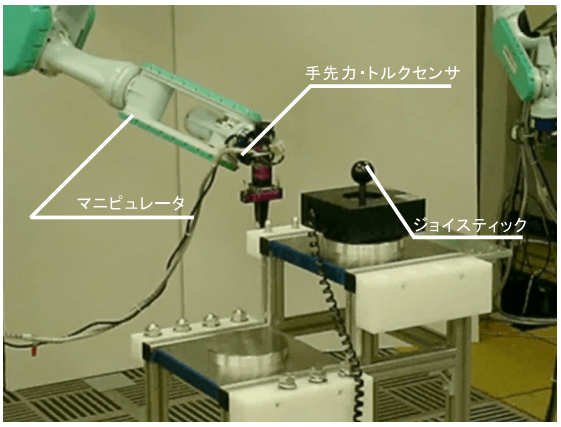

Fig.11 手先力・トルクセンサ、マニピュレータ、ジョイスティック

Table 3 マニピュレータ力制御システムの実行周期

設定周期	2.00 ms
最大周期	2.01 ms
最小周期	1.99 ms
平均周期	2.00 ms
標準偏差	4.41 μs

としてではなく、まったく別々に作成されたプログラムであると言うことである。さらに、これらがリアルタイムに同期して実行された点は重要である。

このように、RT コンポーネントを用いるとソフトウエアの再利用性を向上させるとともに、コンポーネントをブラックボックス化して組み合わせることが可能となるため、複雑なシステムの構築を柔軟に行うことができる。

8. おわりに

本稿では、はじめにロボットシステムを構成するための体系的な方法論の必要性について議論した。また、こうしたシステム構築を実装レベルでサポートするための、プラットフォーム「RT コンポーネント」の必要性について述べた。複数の RT コンポーネントを用いて、システムを構築するための手法、および複合コンポーネントによるシステムの構成手法について検討した。最後に評価実験において、新たに実装した複合コンポーネントがコンポーネントの独立性を保ちつつ、ロボット制御において不可欠な、リアルタイム実行においても十分なパフォーマンスが得られることが示された。

参考文献

[1] 「21 世紀におけるロボット社会創造のための技術戦略調査報告書」, (社) 日本機械 工業連合会, (社) 日本ロボット工業会, 2001.

[2] 末廣 尚士, 北垣 高成, 神徳 徹雄, 尹 祐根, 安藤 慶昭, ”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

[3] 末廣 尚士, 北垣 高成, 神徳 徹雄, 尹 祐根, 安藤 慶昭, ”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 2)”, 第 21 回 日本ロボット学会学術講演会予稿集, p.1F28, 2003.09

[4] 安藤 慶昭, 末廣 尚士, 北垣 高成, 神徳 徹雄, 尹 祐根, ”RT 要素のモジュール化および RT コンポーネントの実装”, 第 9 回 ロボティクスシンポジア, pp.288-293, 2004.03

[5] 北垣 高成, 末廣 尚士, 神徳 徹雄, 尹 祐根, 安藤 慶昭, ”RT コンポーネントによるマニピュレータ制御システム構築— RT ミドルウエアの基本機能に関する研究開発（その 5）—”, 日本機械学会 ロボティクス・メカトロニクス講演会 2004, p.1A1-L1-6, 2004.06

[6] 安藤 慶昭, 末廣 尚士, 北垣 高成, 神徳 徹雄, 尹 祐根, ”RT 複合コンポーネントおよびリアルタイムコンポーネントの実装-RT ミドルウエアの基本機能に関する研究開発 (その 7)-”, 日本機械学会 ロボティクス・メカトロニクス講演会 2004, p.1A1-L1-5, 2004.06

[7] 安藤 慶昭, 末廣 尚士, 北垣 高成, 神徳 徹雄, 尹 祐根, ”RT 複合コンポーネントおよびコンポーネントマネージャの実装— RT ミドルウエアの基本機能に関する研究開発（その 8）—”, 日本機械学会 ロボティクス・メカトロニクス講演会 2004, p.1C26, 2004.09