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 RT コンポーネントマネージャ

The RT-Component Manager in the OpenRTM-aist-1.0

○正 安藤 慶昭 (産総研)　栗原 眞二 (産総研)　ビグズ ジェフ (産総研)　神徳 徹雄 (産総研)

Noriaki ANDO, National Institute of Advanced Industrial Science and Technology, n-ando@aist.go.jp
Shinji KURIHARA, National Institute of Advanced Industrial Science and Technology
Geoffrey BIGGS, National Institute of Advanced Industrial Science and Technology
Tetsuo KOTOKU, National Institute of Advanced Industrial Science and Technology

RT-Component's interface and its lifecycle have already been defined in the OMG RTC standard[3]. However, in order to manage whole lifecycle of RT-system such as lifecycle of RTCs before instantiation and after destruction, a cirtain management systems are necessary. In this paper, the design and functionality of newly introduced RT-Component manager is discussed. An implementation of the service, and tools which control the service by GUI or CUI are shown.

Key Words: RT-Middleware, RT-Component, component lifecycle

1. はじめに

多数の CPU・センサ・アクチュエータ等から構成される実用的 RT システムを構築する上で, ソフトウエアの各ノードへの配置・設定・起動 (=デプロイメント: deployment[1]) および, システムの起動・終了等のライフサイクルの管理は重要な問題である.

著者らが提案する, コンポーネント指向のソフトウエアプラットフォームである RT ミドルウエア (RTM) では, 多数のノード上に分散配置された RT コンポーネント (RTC) と呼ばれるソフトウエアモジュール群を, 互いに連携させることでシステムを構築する.

本稿では, RTC の生成・破棄を含むライフサイクルの管理操作を上位のアプリケーション等に提供するサービスとして, RTC マネージャを提案する. RTC マネージャの基本的概念については, [2] において既に提案されているが, 本稿では RT ミドルウエアを基盤とした RT システム全体のライフサイクルを考慮し, 必要とされる機能について議論する. 議論の結果定義されたインターフェースと, その実現の方法を示す.

2. RT システム

RT システムは一般に, 1 つ以上の CPU およびアクチュエータ, センサ等から構成される. アクチュエータやセンサは, CPU を持つ制御の主体 (本稿ではこれをノードと呼ぶ) により制御される. また, ノードにはアクチュエータやセンサを制御しない, ソフトウエアのみが動作するものも存在する. これらのノードは, PC のように, 操作者やユーザなどが対話的に操作可能なディスプレイやキーボード等の入出力装置は必ずしも備わっていない. RTM はこうした 1 つ以上のノードがネットワークを介して協調する RT システムのソフトウエアを, 効率よく配置・設定・起動するための機能を提供すべきである.

2.1 システムライフサイクル

多数のノードが関連するシステムにおいては, それぞれのノードのハードウエア起動後, 必要なソフトウエアを起動する必要がある. 一般的な組み込み機器等では, ハードウエア起動時に決まったプログラムが起動する. 一方, RT システムにおいては, 各ハードウエアが動作時に動的に役割が変更される可能性があり, こうした静的なプログラムの起動方法では十分に対処できない. そこで, 中央のアプリケーションプログラムなどから動的にソフトウエアの起動や終了を行う仕組みが必要となる.

2.2 RTC ライフサイクル

RTC は OMG RTC 仕様 [3] によりコンポーネントとしてのライフサイクルが定義されている. 要約すると, RTC には "Created" と "Alive" 状態が存在し, 実行コンテキストが管理する RTC の状態として, "Inactive", "Active" および "Error" 状態, スレッドの状態として "Stopped" および "Running" 状態が存在する.

これらの状態は, RTC のオブジェクトが生成された後での状態であり, 生成される以前の, 生成そのものおよび破棄も含めたライフサイクルについては標準で定められていない. また, RTC はある種のファクトリオブジェクトにより生成されるが, ライフサイクルを考えるとき, こうしたファクトリオブジェクトの生成・削除についても考慮する必要がある.

以上から, RTC 自身のライフサイクルの外側には, 以下の状態が定義可能である.

- ファクトリオブジェクトの生成
- RTC インスタンスの生成
- RTC インスタンスの破棄
- ファクトリオブジェクトの破棄

著者らがオープンソースで配布している RTM: OpenRTM-aist では, 上記のファクトリの生成と破棄は, RTC を含む共有オブジェクト (または DLL) のロードやアンロードと共に行われている.

また, ファクトリや RTC 自体はプロファイル情報を持ち, 動的な環境を構築する上ではこれらの情報を取得する, いわゆるイントロスペクション (Introspection) 機能も必要となる. したがって, こうした RTC の管理を行うサービスを考える場合, このほかの機能として, 下記のような機能が必要とされる.

- システム情報の取得
- モジュールのロード・アンロード
- モジュールに関する情報取得
- RTC の生成・破棄
- RTC に関する情報取得
- サービスの終了

RT システムにおけるノードには, 以上の機能を提供するサービスが必要である. 次節では, こうしたサービス機能を提供する "RTC マネージャ" を提案する.

3. RTC マネージャ

RTC マネージャは, 前述の RTC のライフサイクルの管理および各種情報取得の機能を提供するサービスである. 各種操作は, CORBA を介してネットワーク上の任意のプログラムからの操作が可能であり, アプリケーションプログラム等から動的に RTC の生成・削除を行うことができる.

3.1 マネージャインターフェース

前述の機能要件定義から以下のインターフェースを定義した.

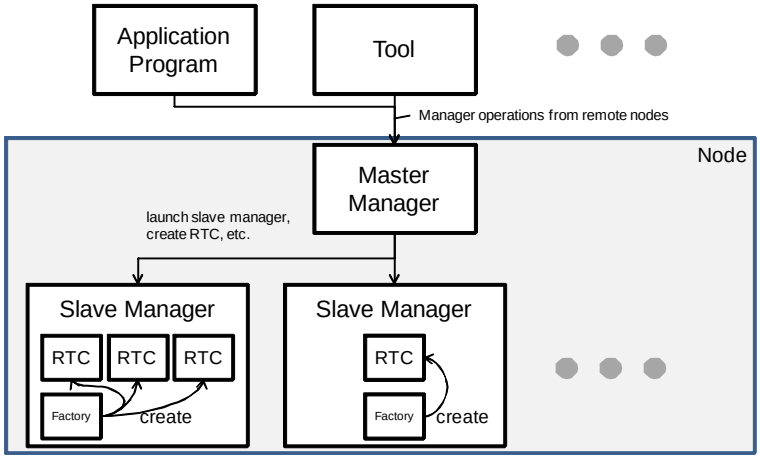

Fig.1 The relation among master manager and slave managers.

```
interface Manager
{
  ReturnCode_t load_module(in string pathname,
                           in string initfunc);
  ReturnCode_t unload_module(in string pathname);
   : (中略)
  RTObject create_component(in string module_name);
  ReturnCode_t delete_component(in string instance_name);
  RTCList get_components();
  ComponentProfileList get_component_profiles();
   : (中略)
  boolean is_master();
  ManagerList get_master_managers();
  ReturnCode_t add_master_manager(in Manager mgr);
  ReturnCode_t remove_master_manager(in Manager mgr);
  ManagerList get_slave_managers();
  ReturnCode_t add_slave_manager(in Manager mgr);
  ReturnCode_t remove_slave_manager(in Manager mgr);
};
```

主たる機能として, 1) モジュールのロード・アンロード・情報取得, 2) コンポーネントの生成・削除・情報取得, 3) マスタマネージャとスレーブマネージャの関係構築, 等が定義されている.

マネージャのマスタ-スレーブ機構はシステムの安定運用とRTC のグルーピングを実現するための機能として導入された. 次節では, マネージャのマスタ・スレーブ機能について言及する.

3.2 マスタ・スレーブマネージャ

マネージャの運用方法として, もっとも簡単なものは, 単一のマネージャが常に起動してサービスを提供し, 外部からのコマンドを受け取りコンポーネントの起動・終了等を行う形態である. 一つのプロセス上で複数の RTC が起動・終了されシステムが運用される.

しかしながら, この方法では 1 つの不安定な RTC がメモリアクセス違反 (いわゆる Segmentation Fault) などを起こした場合でも, 問題のない他の RTC も一緒に強制終了される. 一方で, 複数の RTC が同一プロセスにある場合には, RTC 間の通信が単なる関数呼び出しになるため, 密に連携する複数の RTC を同一プロセス上で動作させることで, 一層の性能向上が望める. OpenRTM-aist の RTC は, 本来任意のプロセス上で動作可能であり, マネージャによって起動される RTC についてもこうした利点を生かして, RTC を動作させるプロセスを任意に指定できる機能を提供することが望ましい.

こうした理由から, 図 1 に示すように, RTC マネージャをマスターとスレーブの 2 種類に分け, それぞれに以下のような機能を分担させることとした.

マスターマネージャ 外部からのマネージャに対する全ての要求を受け付ける. 要求に応じて, スレーブマネージャの起動や任意のスレーブマネージャ上での RTC の生成を行う.

スレーブマネージャ マスターマネージャからの要求に従い, RTC の起動, 各種情報の提供をマネージャに対して行う.

4. 実装

本節では, マネージャの具体的な実装方法について述べる.

4.1 corbaloc URL スキーム

上述のマネージャのマスタ-スレーブ構造を実現させるためには, マスタマネージャとスレーブマネージャ間で, ネゴシエーションを行う必要がある. また, マネージャは, ネットワーク上の任意のアプリケーションに対して, 直接的アクセスを提供する手段を用意すべきである.

こうした観点から, CORBA において URL 文字列によってオブジェクトリファレンスを指定する手法である, INS (Interoperable Naming Service) の corbaloc スキームを利用することとした [4].

通常 CORBA オブジェクトは, IOR (Interoperable Object Reference) と呼ばれるエンコードされた文字列によりオブジェクトを特定する機能が提供されている. しかしながら, 通常のオブジェクトは, 生成される度にランダムな “Object Key” が与えられ, この Object Key も含めて IOR にエンコードされているため, 生成されるたびに異なる IOR を持つ. 一方 corbaloc スキームは CORBA オブジェクトに対して,

corbaloc:iiop:<*host_name*>:<*port_number*> /<*object id*>

という形式で, 判読可能な URL によるアクセスを可能にする.

corbaloc を利用することにより, ポート番号とオブジェクト ID を予め決めておけば, 任意のホスト上のオブジェクトに直接アクセスが可能になる. 本稿では, マスタマネージャのポート番号を “2810”, オブジェクト ID を “manager” とし,

corbaloc:iiop:localhost:2810/manager

により, 直接アクセス可能な CORBA オブジェクトとして実装した.

4.2 マスタ・スレーブ間ネゴシエーション

corbaloc スキームにより, スレーブマネージャはネームサービス等外部のサービスを利用することなく, 直接マスターマネージャにアクセス可能となった. スレーブマネージャは起動後, 同一ホスト上の起動済みマスターマネージャを探し, 自身をマスタマネージャに対して登録する. マスタマネージャは, 登録を要求してきたスレーブマネージャを, 自身が管理す

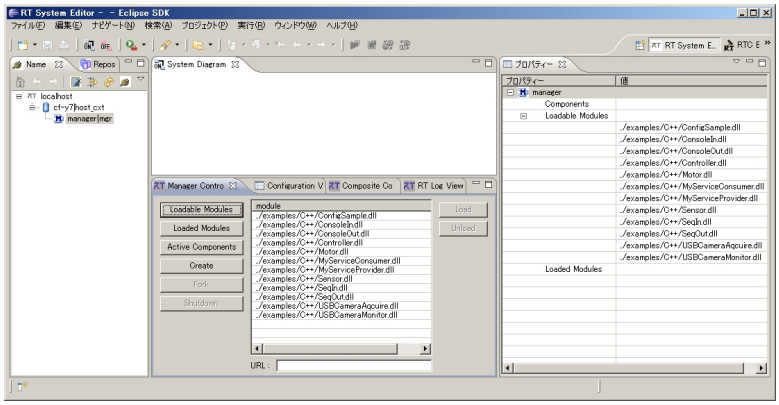

Fig.2 Manager Control View of the RTSystemEditor

Table 1 Commands list of the “rtcshell”

Command	Description
rtact	Activate a component.
rtcat	List component information.
rtcon	Connect two ports together.
rtconf	Manipulate configuration parameters.
rtcwd	Change the current working directory in the RT tree.
rtdeact	Deactivate a component.
rtdis	Disconnect a connection.
rtfind	Search the RTC tree for components.
rtls	List RTCs of the RT tree.
rtmgr	Control managers.
rtpwd	Print the current working directory of the RT tree.
rtreset	Reset a component.

るスレーブマネージャのリストに登録することで, 両者のマスタ・スレーブネゴシエーションが完了する.

以降, マスタマネージャは, 全てのスレーブに対して同一の呼び出しを行うことで, 自身が管理する全てのスレーブマネージャ上の RTC に関する情報等を要求側に提供することができる.

4.3 操作ツール (RTSystemEditor)

OpenRTM-aist では, RT コンポーネントおよびシステムの構築･操作を行うための GUI ツールとして, RTSystemEditor (以降 RTSE) を提供している. RTSE は, オープンソースのソフトウエア開発環境 Eclipse のプラグインとして実装されているため, 他の Eclipse の数多くのプラグインと連携可能である.

Eclipse では, 一連の操作を行う画面のセットをパースペクティブと呼ぶ. パースペクティブは, いくつかのビュー (View) とエディタ (Editor) から構成される. RTSE は一つのパースペクティブとして構成されており, ネームサービスビューやシステムエディタ等から構成される. ここに, RTC マネージャを操作するためのマネージャビューを追加した.

図 2 に新たに実装したマネージャビューを示す. これにより, モジュールのロード, コンポーネントの生成, コンポーネント間の接続等システム構築の一連の作業を 1 つのパースペクティブ上で行うことが可能となった.

4.4 操作ツール (rtcshell)

“rtcshell” はコマンドラインから RT コンポーネントの各種操作を行うためのコマンド群である.

rtcshell では, 表 1 に示すコマンド群が提供されており, コマンドの組み合わせで, コンポーネントの接続や状態変更, コンフィギュレーションパラメータの操作等を行うことができる.

rtcshell にはマネージャを操作するための “rtmgr” と呼ばれるコマンドが提供されている. rtcmgr にはサブコマンドとして load, unload, create, delete が提供され, RTC のモジュールのロード・アンロード, RTC の生成・削除を行うことができる.

このコマンド群により, RTC の生成, RTC 同士の接続や各種パラメータの設定等, システム運用に必要な操作をスクリプト等から行うことが可能である.

5. おわりに

本稿では, RTM により構築された RT システムを運用する上で必要となる基盤サービスとして, RTC マネージャを提案した. RTC マネージャは, RTC の生成・削除等のライフサイクル管理や, アプリケーションプログラム等への情報提供を行う.

マネージャをマスタとスレーブに分割し, それぞれをサービスの提供と RT コンポーネントのホスティングに特化させることにより, より安定した運用を可能にした. また, CORBA の機能の一つである INS の corbaloc スキームを利用することにより, ネームサービス等に依存しない, マネージャへの直接アクセスを提供することで, マスタ・スレーブ構造や, 外部アプリケーションに対して利便性を提供することができた.

また, これにより RTC の起動等を上位のアプリケーションプログラムから一括して行うことができるため, 多数のノードから構成される大規模な RT システムであっても, 運用が非常に容易に行える.

謝辞

本研究の一部は平成 21 年度, NEDO 次世代ロボット知能化技術開発プロジェクト (平成 19〜23 年度) の補助を受けた.

文献

[1] Object Management Group, Deployment and Configuration of Component-based Distributed Applications, formal/2006-04-02 (2006)
[2] 安藤他, “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 ミドルウェアの基本機能に関する研究開発（その 8）−”, 第 22 回 日本ロボット学会学術講演会予稿集, p.1C26, 2004
[3] Object Management Group, Robotic Technology Component Specification Version 1.0, formal/2008-04-04 (2008)
[4] Object Management Group, Interoperable Naming Service, formal/00-11-01 (2000)