Erlangに基づいたRTミドルウェア用コンポーネントマネージャ

A component manager for RT-Middleware based on Erlang

○ BIGGS Geoffrey（産総研）　安藤 慶昭（産総研）　神徳 徹雄（産総研）

Geoffrey BIGGS, National Institute of Advanced Industrial Science and Technology
Noriaki ANDO, National Institute of Advanced Industrial Science and Technology
Tetsuo KOTOKU, National Institute of Advanced Industrial Science and Technology

The deployment and configuration of the software components that make up a complete robotic system using a component-based framework is an important step in using that system. We have developed an experimental tool for the organisation and management of components based on RT-Middleware. It is based on the supervision tree concept from the Erlang programming language. Components are organised into a hierarchy of sub-systems and managed in groups. The supervision tree ensures that dependencies between components are accurately managed when errors occur.

Key Words: RT ミドルウェア、RT コンポーネント

1. はじめに

現在、高度なサービスを行う次世代ロボットを効率的に開発するため、OpenRTM-aist を始めとする様々なソフトウエアプラットフォームの開発が盛んに行われている。コンポーネント指向で開発されたシステムにおいては、多数のソフトウエアコンポーネントを適切なノードへ配置し、設定を行う、いわゆるデプロイメントとコンフィギュレーションの機能が重要となる。デプロイメントおよびコンフィギュレーションのプロセスには、コンポーネントを実行するノードへのソフトウエアリソースのコピー、コンポーネントのインスタンス化、要求に応じたパラメータのセット、およびコンポーネント間の接続等が含まれる。コンポーネントネットワークが大きいほど、このタスクは複雑になる。

特にエラーが生じた場合において、コンポーネントネットワークをシステム状態の変化に応じてダイナミックに変更する必要があるため、適切なシステム管理手法の導入が不可欠である。発生したエラーによっては、影響を受けたコンポーネントあるいは関連するコンポーネントをリセットし、あるいは新たなコンポーネントを起動し、コンポーネントネットワークを変更する必要があるかもしれない。

本稿では、著者らが開発するコンポーネントフレームワーク OpenRTM-aist の RT コンポーネントのライフサイクルを管理するツールを提案する。本ツールを利用することで、コンポーネントをインスタンス化、コンポーネント間のポートの接続、およびシステム全体の動作を開始すめるために、必要なコンポーネントをアクティブ化する、といった一連の操作を一括して行うことができる。さらに、コンポーネントがもはや必要でない場合、コンポーネントを非アクティブ化しインスタンスを削除することもできる。最も重要なことは、コンポーネントネットワークを適切に再構成することにより、エラー状態に入るコンポーネントに対応することができる機能である。ツールは Erlang と呼ばれる言語において利用されている階層化管理構造：スーパービジョンツリー (Supervision tree) に基づき実装する。

2. スーパービジョンツリー

スーパービジョンツリーはソフトウェアエンティティの階層的構造である。ソフトウェアはワーカーとスーパーバイザから構成される。ワーカーはソフトウェア・アプリケーションの機能の提供に責任を負う。一方、スーパーバイザは、それらが遷移しうるあらゆるエラー状態を含むワーカーのライフサイクルの管理に責任を負う。階層はツリーを構成し、ツリーの枝はワーカー、節点はすべてスーパーバイザである。この構造を図 1 に示す。

アプリケーションは一番上の節点、すなわちツリーの最上位に位置し、これは一つのスーパーバイザでもある。このスーパーバイザは、アプリケーション全体のライフサイクルを管理する。これがエラーとなる場合、アプリケーション全体におけるエラーを意味する。

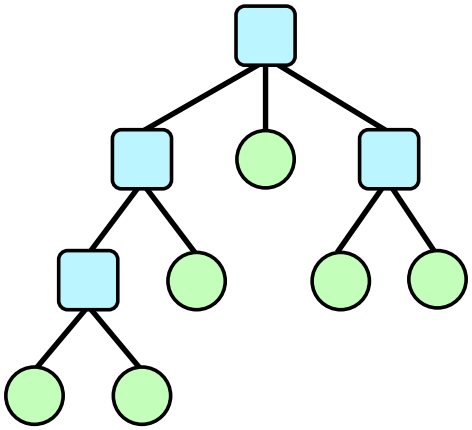

Fig.1: スーパービジョンツリーの構造。四角はスーパーバイザ。円はワーカー。

スーパービジョンツリーは Erlang Open Telephony Platform　ライブラリ [1] の主たる特徴である。この考え方は Ericsson 社がフォールトトレラント分散システム開発するために開発したプログラミング言語 Erlang concurrent programming language [2] において実装され、実際に 10 年間以上交換機等のミッション・クリティカル・システムにおいて使用されている。スーパービジョンツリーを使用することで、分散システムのソフトウエアコンポーネントを比較的容易に組織化することができる。

2.1 スーパーバイザー

Erlang におけるスーパーバイザは、他のプロセスのライフサイクルを監視することがその唯一の責務であるようなプロセスである。スーパーバイザが開始される時、どのような子プロセスが配下にあるか、またエラー時に何を行うかについての設定をロードする。その後、設定に従い予め決められた順序で子プロセスを順次同期的に開始する。スーパーバイザの子プロセスはワーカーあるいは自分の子プロセスを管理する他のスーパーバイザである。同期的起動はスーパービジョンツリーのワーカーが下位から順に起動されることを保証する。これによりワーカーが起動する時、コンポーネント間の依存関係を満たすことができる。

子プロセスが失敗した場合、スーパーバイザはそれを再起

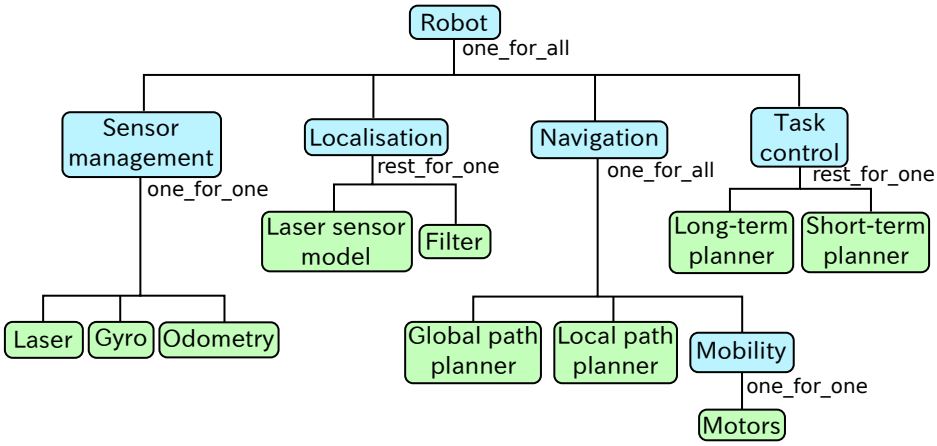

Fig.2: 簡単なロボットのスーパービジョンツリー。

動し、子プロセスの間の依存性を保証するために、さらに他の子プロセスを再起動するかもしれない。エラーがスーパーバイザが扱える事前に設定された周期より頻繁に発生する場合、スーパーバイザもエラーとなる。この場合、エラーがなかった子プロセスを含むすべてのコンポーネントをすべてシャット・ダウンし、親スーパーバイザへエラーであった旨を知らせる。エラーハンドリングはその親スーパーバイザの仕事になる。そのエラーを扱うことができるスーパーバイザが見つかるまでエラーはスーパービジョンツリーを伝搬する。一般的に、致命的なエラーほどツリーの上位まで伝搬する。

2.2 ワーカー

Erlang におけるワーカーはそれぞれ個別のプロセス[1]である。それらは互いにサービスを提供しあい、Erlang アプリケーションの機能を構成する。すべてのワーカーは必ず唯一のスーパーバイザを持ち、そのスーパーバイザにより管理される。このポリシーにより、万一ワーカーがエラーになった場合、必ずそのエラーを捕捉し、適切な処置を講ずることを保証することができる。

3. RT コンポーネントのスーパービジョン

スーパービジョンツリーのコンセプトは、その生存期間中に協調しあう複数のソフトウエアコンポーネントから構成される一般的なアプリケーションに適用可能である。ロボットシステムも、多数の協調しあうサブシステムから構成され、一つのエラーが他のサブシステムへ波及する可能性のある、こうしたシステムの一種である。図 2 に、簡単なロボットシステムにおけるスーパービジョンツリーの例を示す。

本稿では、OpenRTM-aist のためのコンポーネント管理のための枠組みを、Erlang 言語上のスーパービジョンツリー上に実装した。

3.1 スーパーバイザー

Erlang のスーパーバイザ [3] は RT コンポーネントを管理するために試用した。スーパーバイザ起動に、XML ファイルから管理のためのスペックを読みこむ。スペックはスーパーバイザの子要素のエラー処理のポリシーについて記述されている。ここで子要素とは、RT コンポーネントおよび他のスーパーバイザである。

XML ファイル内の子要素に関する情報には、構成要素のインプリメンテーションを提供する共有モジュール名、コンポーネントがインスタンス化される時のインスタンス名が含まれる。子要素のスーパーバイザに関する情報は、再帰的階層的構造を持つ XML の個別ノードとして存在する。

Erlang 上に実装されたスーパーバイザがそのスペックをロードすると、同期的にその子要素を起動する。スーパーバイザはこの処理を行うと同時に、ワーカープロセスには OpenRTM-aist マネージャー内でコンポーネントをインスタンス化するために必要な情報が与えられ、子要素のスーパーバイザは、スペックを含む解析済の XML ノードが渡される。

3.2 ワーカー

今回実装した RT コンポーネントのためのスーパービジョンツリーにおいては、それぞれのワーカープロセスは、それぞれの RT コンポーネントに 1 対 1 で対応する。ワーカーがスーパーバイザの子プロセスとして作られると同時に、OpenRTM-aist のマネージャーでコンポーネントインスタンスが作成される。それ以降、ワーカーは定期的にコンポーネントのステータスを確認する。

コンポーネントがエラー状態に入った場合、モニタリング・プロセスはエラー終了する。これは、親であるスーパーバイザのエラー管理処理のトリガとなる。これによって、RTC のエラーハンドリングメカニズムを使用せずに済まないため、コンポーネントのインスタンスは削除されない。代わりに、新たなモニタリングプロセスがスーパーバイザにより生成されると同時に、それはコンポーネントがすでに存在することを検知し、コンポーネントのリセットを試みる。

3.3 エラー管理

コンポーネントがエラー状態に遷移する時、モニタリング・プロセスはエラー終了し、これは最終的にスーパーバイザのエラーハンドリングを呼び出す。その時の振る舞いを図 3 に示す。

スーパーバイザは、エラーに応答する方法として、予め決められた 3 つのポリシーのうちの 1 つを取ることができる。これは Erlang のスーパーバイザと同様である。

one_for_one スーパーバイザは、単に死んだ子要素を再開する。

one_for_all スーパーバイザはその子要素をすべてシャット・ダウンし、あらためてそれらをすべて再起動する。

rest_for_one スーパーバイザは、失敗した子要素の後に起動された子プロセスをシャット・ダウンし、エラーになった子要素を再起動後にそれらを再起動する。

本稿における実装では、スーパーバイザはすべての再起動された子要素のコンポーネントに対して新たにモニタリング・プロセスを起動する。

さらに、スーパーバイザはエラーをどの程度の頻度で扱うことができるか定義するポリシーを持つ。エラー発生の頻度が設定値より大きければ、スーパーバイザはすべての子要素をシャットダウンし、その後自身をシャットダウンしたうえで、親スーパーバイザに対してエラーを報告する。その後、

[1] ここでいうプロセスは Erlang 言語におけるプロセスであり、OS が提供するプロセスとは一般には別のものである。

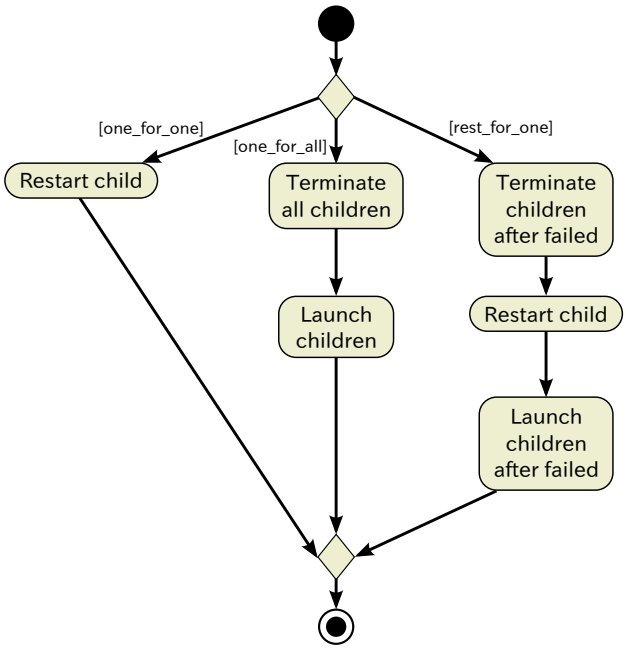

Fig.3: エラーが起こる時の反応。

さらに親となるスーパーバイザは、自身のエラーハンドリングポリシーに従ってエラーを取り扱う。このツリーにそって行われるエラー伝搬により、エラー処理中にコンポーネント間の依存性が正確に扱われることが保証される。

図 2 のスーパービジョンツリーのエラー状態の反応の例は Listing 1 に示される。

3.4 RT コンポーネントのインスタンス作り

OpenRTM-aist において、RT コンポーネントは共有ライブラリモジュールとして提供される。コンポーネントマネージャは当該 RT コンポーネントの共有ライブラリモジュールをロードし、インスタンスの生成を行う。Erlang 上では、コンポーネントマネージャを Erlang Native Instruction Format モジュールを用いてインターフェイスし、コンポーネントインスタンスはインスタンス名によって参照される。

4. おわりに

コンポーネントの状態の変化（特にエラー状態）に応じて、コンポーネントネットワークをダイナミックに管理することはコンポーネント指向システムを使用する上で重要な部分である。本稿で提案したスーパービジョンツリーはシステムのコンポーネントのライフ・サイクルを管理する一つの方法である。Erlang のスーパービジョンツリーに基づいたコンポーネントライフサイクルマネージャーを実装し、コンポーネントをツリー構造で管理方することで、子要素で発生したエラーを上位のスーパーバイザに伝搬させ、コンポーネント間の依存性考慮しつつ適切にエラーが処理されることを示した。

他の配置と設定ツールと比べて、スーパービジョンツリーはエラーを補足し、コンポーネント間の依存関係を考慮しつつ適切なエラー処理をする異ガ可能である。例えば、ROS[4] の `roslaunch` ツールはスタティックである。配置後のシステム状態変化に反応することができない。

謝辞

本研究は科研費（21700230）の助成を受けたものである.

参献

[1] Erlang/OTP R14B01, http://www.erlang.org/doc/, 2011.
[2] Erlang Programming Language, Official Website, http://www.erlang.org/, 2011
[3] Erlang – supervisor, http://www.erlang.org/doc/man/supervisor.html, 2010
[4] ROS Wiki, http://www.ros.org, 2011.

Listing 1: スーパービジョンツリーのエラー反応の例。12 行目に「laser」コンポーネントにエラーが起こる。16 行目に「global_pp」コンポーネントにエラーが起こる。左側の数字はタイムスタンプである。

```
1  {1299,890269,406989}: Component "short_term"
       monitor starting
2  {1299,890269,483709}: Component "long_term"
       monitor starting
   {1299,890269,489114}: Component "local_pp"
       monitor starting
4  {1299,890269,493642}: Component "global_pp"
       monitor starting
   {1299,890269,500778}: Component "motors"
       monitor starting
6  {1299,890269,507620}: Component "filter"
       monitor starting
   {1299,890269,518561}: Component
       "laser_sensor_model" monitor starting
8  {1299,890269,528334}: Component "odometry"
       monitor starting
   {1299,890269,544259}: Component "gyro"
       monitor starting
10 {1299,890269,583013}: Component "laser"
       monitor starting

12 {1299,890274,644635} Component "laser" is in
       error
   {1299,890274,644840} Component "laser"
       terminating abnormally: comp_error
14 {1299,890274,648108}: Component "laser"
       monitor starting

16 {1299,890280,501509} Component "global_pp"
       is in error
   {1299,890280,501629} Component "global_pp"
       terminating abnormally: comp_error
18 {1299,890280,504418} Component "motors" shut
       down
   {1299,890280,508650} Component "local_pp"
       shut down
20 {1299,890280,512055}: Component "local_pp"
       monitor starting
   {1299,890280,530745}: Component "global_pp"
       monitor starting
22 {1299,890280,538439}: Component "motors"
       monitor starting
```