RTC の繋ぎ替えによるロボット制御管理ソフトウエア「FIROSOPHY」の開発

Development of RT System Control Software which changes connections of RTCs

菅佑樹 (早大) ○，　正　菅野重樹 (早大)

Yuki Suga, Waseda University, ysuga@ysuga.net
Shigeki Sugano, Waseda University

Abstract— In this paper, the robot behavior designer called “FIROSOPHY” which sequentially changes connections of RT-components is shown. First, it is pointed out that most of the recently developed RT-systems are constructed as a central controlling system which has various problems, such as complexity of development, debugging and maintaining. Next, the FIROSOPHY is shown. In FIROSOPHY architechture, the RT-task is provided as sequentially executing the RT-systems that achieve premitive skills like reaching, walking, etc. Using FIROSOPHY, prural research results on the same hardware platform can be easily combined.

Key Words: RT Middleware

1.　はじめに

近年，ロボット技術の広範な分野への応用が期待されている．一方で我々は，国内外の研究機関がそれぞれに異なるアーキテクチャの上でロボット開発を行っているため，研究成果の検証や再利用が難しいことが業界全体の課題であると考えている．

これに対し，アクチュエータなどの機能を別個にコンポーネント（RT Component, RTC）として開発することで要素技術の再利用性を高めることを目的とし，産業技術総合研究所などが中心となってまとめた，いわゆる「RT ミドルウエア」規格が提案されている．そしてその実装として，産業技術総合研究所が開発している OpenRTM-aist や，株式会社セックが開発している OpenRTM.NET などがある [1] [2]．この RT ミドルウエアを用いたシステム（以下，RT システム）では，RTC の持つデータポートおよびサービスポートを接続し，データの入出力関係を定義することで，ブロック線図のように明快にシステム内のデータの流れを設定することができる．またそれぞれの RTC は active, inactive, error, および created の「状態 (state)」を持っており，さらに制御パラメータを調整するコンフィグレーション (configuration) の機能を持っている．これらの状態変数を RT System Editor 等のツールから制御することで，RT システムの振る舞いを開始，停止，および調整することができる．

これらの RT ミドルウエアの実装を用い，すでに多くのロボットが開発・運用されている．しかし我々は，それらの多くのシステムが中央集権的なシステム構成を取っているため，RT ミドルウエアの特徴の一つである分散制御型アーキテクチャの利点を生かし切れていないと考えている．本稿で述べる中央集権型 RT システムとは，出力ポートを持つセンサ系 RTC と，入力ポートを持ち，アクチュエータ等の効果器を司るアクチュエータ系 RTC を複数配置し，それらのデータの入出力が中央制御 RTC に接続される形を持つ（図 1）．

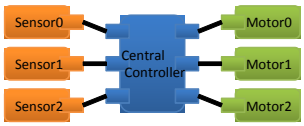


Fig.1 Center-Oriented RT System

この構成をもつ場合，単一障害点の顕在化やメンテナンスおよびデバッグの煩雑化などの問題が予期される．たとえばヒューマノイドのように複数の動作を選択的かつ順序的，もしくは並列的に実行する場合，中央制御 RTC は認識動作や，歩行動作などのすべての動作制御アルゴリズムを内包している必要があり，またそれらの選択的実行に係わるアーキテクチャを持つ必要がある．

これに対して RT ミドルウエア規格では，FSM コンポジットの枠組みが提案されているが，この実装は現在のところ存在せず，おおくの研究機関がすで DataFlowComponent のみを利用した RT システムの構築を行っている．

一方で本研究では RT システムを，RTC の状態を分岐条件としながら分解・再構築を自動的に行うアーキテクチャを提案する（図 2）．この方法では，リーチングや移動，しゃがみ動作や認識などの動作を一つの RT システムに対応させ開発することができる．以下本稿では，このような単一の RT システムと結びついた動作を RT スキルと呼び，本稿で提案するアーキテクチャによって開発された一連の動作の流れを RT タスクと呼ぶこととする．

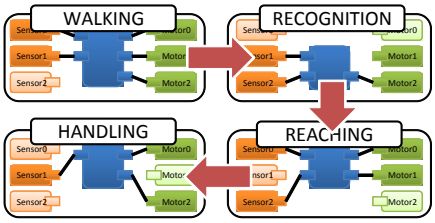


Fig.2 FIROSOPHY

本提案のアーキテクチャを用いた場合，同一プラットフォームを用いた複数のチームが開発した異なる RT スキルを簡単に再利用することができる．また，本提案のアーキテクチャは，同一の RT タスクを提供する異なるプラットフォームを用いた場合においても移動やハンドリングなどの RT スキルに対応する RT システムを用意できれば再利用することが容易であり，RT タスクのレベルでの開発資産の劣化を防ぐことができる．

本研究では，このアーキテクチャを FIROSOPHY（FInite RObotics States OPeration Handler sYstem) と名付けた．本稿では，FIROSOPHY の概要と，実装について解説する．

2.　提案手法

FIROSOPHY では，RTC と同じく有限状態マシンの考え方を適用している．それぞれの状態 (state) は，active および inactive の 2 種類の状態を持ち，RT スキルを担う要素である RT システムと結びついている．active な状態であれば，RT システムは接続およびコンフィグされた状態にあり，基本的に RT システムに参加し

ているすべての RTC が active 状態での制御を行い，RT スキルの完遂を目指す．また FIROSOPHY における遷移 (transition) は RT システムの繋ぎ替えに相当し，遷移が起こる条件 (guard) として，RTC の状態 (active, inactive, error)，RT システム構築からの経過時間，およびその組み合わせを指定することが出来る．

RT スキルを実行する RT システムを実装する場合は，一つ，ないしは複数の RTC がスキル完遂の監視を行い，スキルの完遂後に deactivate，もしくは失敗で error 状態に移行するようにしておく．これにより，RT スキルの達成度を RTC の状態監視によって行い，スキル完遂後に次の状態に遷移することにより，複数のスキルを選択・実行することができる．

3. 提案手法の実装

本研究で開発した FIROSOPHY の実装である FIROSOPHY (ver.0.2.0) を図 3 に示す．FIROSOPHY の本実装は OpenRTM-aist の Java 版（ver.1.0RC1）を使用しており，シンプルな GUI を持ち，UML で定義されるステートマシン図を描くのと同じ感覚で制御アルゴリズムを実装することができる．RT システムは OpenRTM-aist プロジェクトの成果である RT System Editor と同形式の XML ファイルを使うことができる．また，設定画面により，遷移が起こった場合に，遷移元の RT システムに含まれる RTC の deactivate 処理や接続の破棄，遷移先の RT システムの RTC のリセット，接続の破棄などの細かい設定が選択できる．

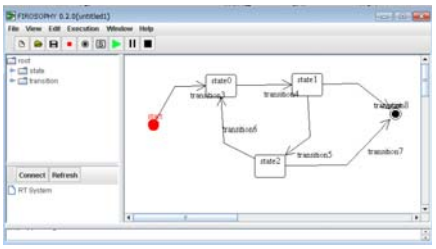


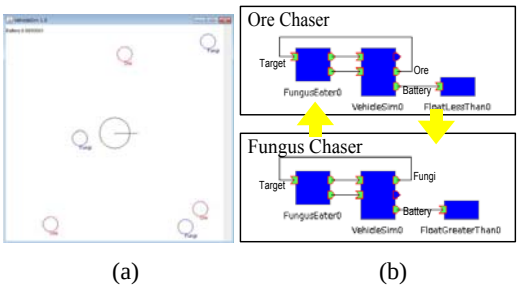
Fig.3 FIROSOPHY ver.0.2.0

4. 提案手法を用いたアーキテクチャの一例

本稿では FIROSOPHY を用いたシステムの一例として，R.Pheifer のキノコ喰いロボットの構築例を挙げる [3]．Fungus Eater (キノコ喰いロボット) は，ある惑星に送り込まれ，惑星に生えるキノコ (fungi) をエネルギ源としながら鉱石 (ore) を集める架空のロボットである．

図 4 にキノコ喰いロボットのシステムを示す．図 4a は，シミュレーション環境で，中央のロボット 1 台とキノコおよび鉱石が複数個配置されている．図 4b はキノコ喰いロボットの RT システムで，図 4b 上は鉱石を追うシステム，図 4b 下はキノコを追うシステムであり，二つの RT システムは，同じ制御 RTC（FungusEater0.rtc）を使いながら目標値を鉱石位置およびキノコ位置を与えている．またバッテリ残量を監視する RTC を，閾値よりも少なくなると inactive になる FloatLessThan（閾値 0.5）と，その逆で閾値よりも大きくなると inactive になる FloatGreaterThan（閾値 0.7）を使っている．これにより，OreChaser では鉱石を追いかけ，バッテリ残量が 50[%] 未満になると FloatLessThan コンポーネントが inactive に，FungiChaser ではキノコを追いかけ，バッテリが 70[%] 以上になると FloatGreaterThan コンポーネントが inactive になる．

図 5 に，Fungus Eater の FIROSOPHY 図を示す．ここでは，二つの状態（頂点の丸い長方形）を持っており，それぞれ鉱石を追う OreChaser の RT システムとキノコを追う FungiChaser の RT システムと結びついている．それぞれを結ぶ遷移は，バッテリ電圧の状態を監視する RTC の状態に応じて起こるものとした．これにより，OreChaser の状態においては，バッテリ残量が 0.5 よりも小さくなると FloatLessThan コンポーネントが inactive になるため，FungiChaser 状態に遷移し，また FungiChaser 状態ではキノコを追って充電を行うが，バッテリ残量が 0.7 よりも大きくなった時点で，OreChaser 状態へと遷移を行う．



(a) (b)

Fig.4 Fungus Eater Simulator

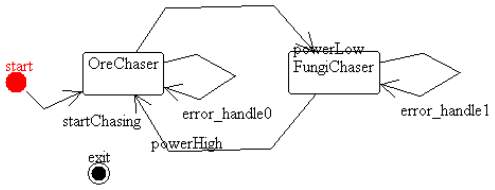


Fig.5 Fungus Eater System

また，図 5 中の二つの状態は自分自身への遷移を持っているが，これらは RT システムに含まれる RTC のいずれかが error 状態になった場合に発動し，自動的にリセット後，接続を張り直して activate を行うことで，不意のエラーに備えるものとした．本稿で用いたシステムは，任意の確率で error 状態になる機能を備えており，これによってエラー状態からの自動復帰動作も確認した．

このように FIROSOPHY を使えば，タスク遂行監視を行う RTC の状態変化に応じて，複数の RT システムを使い分けるシステムの構築が可能であり，またエラー状態の監視を行うことで，自動的にコンポーネントをリセットしたり，エラーから復帰するための RT システムを追加することもでき，より堅牢で分散的なロボットシステムの構築・運用が期待できる．

5. まとめと今後の展望

本稿では，RT ミドルウエアの枠組みを用いたマルチタスクなロボットにおけるサービスの実装方法として FIROSOPHY を提案した．FIROSOPHY では，タスクの進行に応じて RT システムの繋ぎ換えを行うことによって，複雑な一連のタスク実行を分割して実装することができ，同一プラットフォームを用いた共同開発事例などで，成果を再利用する際に有効に働くと考えられる．

今後は RT システムの読み込みだけでなく，編集機能を実装してより使いやすい環境を目指すことと，フォークとジョインによる並列的な RT スキル実行を可能とすることが課題である．

参考文献

[1] RT-Middleware: OpenRTM-aist Official Website: URL: http://www.is.aist.go.jp/rt/OpenRTM-aist/ (last accessed 2010/03/08)

[2] 長瀬雅之，中本啓之，池添明弘：”はじめてのコンポーネント指向ロボットアプリケーション開発 RT ミドルウェア超入門”，株式会社 毎日コミュニケーションズ，2008 年

[3] Pfeifer, R.: The“Fungus Eater”Approach to Emotion: A View from Artificial Intelligence, Cognitive Studies, 1994

[4] FIROSOPHY: http://www.ysuga.net/robot/rtm/FIROSOPHY