

知能ロボットソフトウェア用 コンポーネント指向ミドルウェアの開発

Development of Component Based Middleware for Intelligent Robot Software

関 谷 真*

Makoto SEKIYA

宮 下 善 太*

Zenta MIYASHITA

根 木 教 男*

Norio NEKI

大 野 信 之*

Nobuyuki ONO

要 旨

ロボットの知能化が進むにつれ、機能を実現するためのソフトウェアシステムは複雑かつ大規模化しており、コンポーネント指向ミドルウェアを用いて、機能単位(コンポーネント)を適切なインターフェースで分割し、それらを結合して全体を設計する手法が有効である。

知能ロボットのソフトウェアシステムには、階層型コンポーネント、状態遷移型コンポーネントが必要となり、ミドルウェア開発にあたっては、これらのコンポーネントが特定のタイミングに同期できることと、一定時間内に一定の順序で処理を終えるために、通信を含む処理時間が最小かつ予測可能なハードリアルタイム性を保証することが課題となる。

本稿では、上記課題の解決をねらいとして独自に開発した、コンポーネント指向ミドルウェア Honda Robotic Technology Middleware について紹介する。

なお、ミドルウェアの仕様は外部機関での利用も考慮し、国際標準化団体 Object Management Group で策定された Robotic Technology Component 準拠としたが、独自拡張仕様については、Object Management Group における標準化作業を推進中である。

ABSTRACT

As robots become more intelligent, the software systems for realizing functions are becoming larger and more complex. This makes the use of component based middleware an effective means of design by dividing the overall system into function units (components) and then joining these components with the appropriate interfaces.

Intelligent robot software systems require composite components and finite-state machine components. The issues when developing middleware are the ability to synchronize these components to specific timings, and guaranteeing hard real time performance of minimum and predictable processing times including communication so that processing can finish in a certain sequence within a certain time.

This paper introduces Honda Robotic Technology Middleware, which is original component based middleware developed to address the above issues.

In consideration of use by outside organizations, the middleware specifications comply with the Robotic Technology Component specification formulated by the Object Management Group, which is an international standards organization. In addition, work is underway within the Object Management Group to standardize the original expanded specification.

1. はじめに

二足歩行制御から始まり⁽¹⁾、作業機能やインタラクション機能を進化させてきた^{(2),(3)} ASIMO のソフトウェアシステム(以下システム)は、近年その複雑さと規模が急速に拡大しており、コンポーネント指向ミドルウェアを用い

て、システム全体を適切なインターフェースで分割した機能単位(以下コンポーネント)を結合して設計することが有効である。なぜなら、システムをより小規模かつ簡潔な機能単位で理解し、検証することが可能となるからである。

ASIMO のような知能ロボットシステムに適用可能なコンポーネント指向ミドルウェアの開発にあたっては、既存の

*基礎技術研究センター

システムをコンポーネント化するために、階層型コンポーネントと状態遷移 (FSM) 型コンポーネントの設計および実行機能が必要となる。また、これらのコンポーネントが特定のタイミングに同期できることと、一定時間内に一定の順序で処理を終えるために、通信を含む処理時間が最小かつ予測可能なハードリアルタイム性を保証することが課題となる。

本稿では、上記課題の解決をねらいとして開発した、コンポーネント指向ミドルウェア Honda Robotic Technology Middleware (HRTM) について紹介する。

2. コンポーネント指向ミドルウェア

コンポーネント指向ミドルウェア (以下ミドルウェア) とは、コンポーネント指向開発を実現するための開発環境であり、Fig. 1 に示す設計ツール、コンポーネント仕様、およびフレームワークの三つの要素からなる階層構造で構成される。

設計ツールは、視覚的な手法によって、コンポーネント定義や接続関係の設計をし、コンポーネントのソースコードを生成することを目的としたツールである。デバッグ支援機能などを持つ場合もある。

コンポーネント仕様は、ミドルウェア上の共通機能単位となるコンポーネントを操作するための共通インターフェースを定義するものであり、一般的にはデータ送受信用のポートや、パラメータの取得および変更のためのインターフェースなどが定義される。

フレームワークは、コンポーネント仕様に基づいて実装された、コンポーネントの起動や通信など、共通インターフェースの実際の処理を行う特別なプログラムである。

Software In the Loop Simulation (SILS), Hardware In the Loop Simulation (HILS) およびターゲット環境におけるコンポーネントの再利用を実現するためには、フレームワークが各々の環境でコンポーネントが必要とする機能を提供しなければならない。

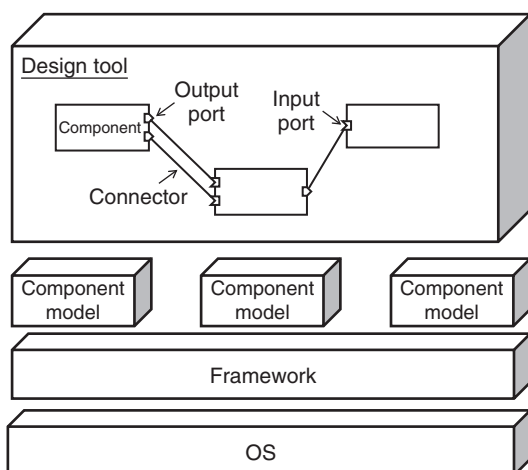


Fig. 1 Architecture of a component based middleware

3. 知能ロボット用ミドルウェアの要件

知能ロボットシステムに対するミドルウェアの役割は下記の通りである。

1. 知能ロボットシステムをコンポーネント化する
2. SILS, HILS, ターゲット環境間でコンポーネントの再利用を実現する

ここでは、上記の役割から導き出した、ミドルウェアの構成要素に対する具体的な要件について説明する。なお、本開発における特徴的な要件については、階層型コンポーネント、状態遷移型コンポーネント、複数通信プロトコル対応である。

3.1. 設計ツール要件

3.1.1. 標準的なモデル言語でシステムを設計できること

システム設計に独自のモデル言語を使用すると、ツールの使用方法の習熟や設計内容の理解が難しくなるうえ、既存の設計ツールを再利用することもできない。また、外部機関での使用を想定し、設計ツールには広く知られている標準的なモデル言語を採用する。

3.2. コンポーネントに関するミドルウェア要件

3.2.1. コンポーネントを階層的に設計できること

大規模かつ複雑化した知能ロボットシステムは、一つの機能が複数のサブ機能で階層的に実現されているため、これをコンポーネント化するためには、コンポーネントをサブコンポーネントによって階層的に設計できる必要がある。

3.2.2. 状態遷移型コンポーネントを設計できること

知能ロボットシステムの作業機能やインタラクション機能の実装には、複雑な状態遷移が使用されており、そのメンテナンス性の向上が課題となっているため、状態遷移型コンポーネントは内部の状態遷移をモデルとして設計できるようにする。

3.2.3. 標準的なコンポーネント仕様に準拠すること

外部機関で開発された多くのコンポーネントを再利用してシステムを構築することが容易になるため、広く採用されている標準的なコンポーネント仕様を採用する。

3.3. フレームワーク要件

3.3.1. コンポーネントを様々な OS 上で実行できること

知能ロボットのハードウェアシステムは、役割の異なる複数のセンサから構成されている。これらのセンサは、提供されるドライバの制約等によって、様々な OS 上で動作しており、センサからのデータを高速に処理するためのプログラム群は、センサと同じ OS 上で実行されている。

このようなシステムをコンポーネント化するためには、フレームワークに対して、様々な OS でコンポーネントの実行環境を提供するマルチプラットフォーム対応が要求される。

3.3.2. ソースコードを異なる OS 間で再利用できること
SILS, HILS, ターゲット環境間でコンポーネントの再利用を実現するために、フレームワークはコンポーネントのソースコードを異なる OS 上で再利用できるようにする必要がある。

3.3.3. コンポーネント間通信に複数のプロトコルを利用できること

知能ロボットシステム内の各プログラム同士の通信では、通常、用途に応じて TCP/IP, UDP, 共有メモリ, バス通信などが使い分けられている。

通信プロトコルを使い分ける理由は、プログラムの動作環境で利用できる最も高速な通信手法を用いることが、処理時間の短縮につながるからである。

また、SILS と HILS, ターゲット環境間でコンポーネントが使用する通信プロトコルが異なることもある。このような状況でコンポーネントの再利用と処理の高速化を実現するために、フレームワークはコンポーネント間通信の複数のプロトコルに対応し、それらをコンポーネントのコンパイル時や実行時に選択できるようにする必要がある。

3.3.4. フレームワークの処理時間を最小限にすること

コンポーネント化によって、処理時間が著しく増加してしまうと、SILS から処理能力が限られるリアルタイム OS で動作するターゲット環境への移行が難しくなる。フレームワークはコンポーネントの再利用性を高めるため、通信を含めた処理時間を最小限かつ一定にできるようにする必要がある。

4. HRTM の開発

HRTM の開発にあたり、主なロボット向けミドルウェアの機能に関する調査をした。以下、調査対象のミドルウェアを簡単に説明する。

OpenRTM-aist⁽⁴⁾ は日本の産業総合研究所が開発したロボット向けミドルウェアで、国際標準化団体である Object Management Group (OMG)⁽⁵⁾ Robotic Technology Component (RTC)⁽⁶⁾ 準拠や幅広い OS への対応、通信プロトコルに国際標準の OMG CORBA⁽⁷⁾ を採用することを特徴とする。また、ART-Linux⁽⁸⁾ 上でのハードリアルタイム対応実績がある。

ROS⁽⁹⁾ はアメリカの Willow Garage 社が開発したロボット向けミドルウェアで、現在は Open Source Robotics Foundation で開発が継続されている。オープンな体制で、多数の開発者による豊富な成果物が存在するが、正式な対応 OS は Ubuntu に限られる⁽¹⁰⁾。

OROCOS⁽¹¹⁾ は欧州連合がスポンサーとなって始まったロボット用ソフトウェア開発プロジェクトである。主要な OS に対応し、リアルタイム環境での動作実績もあるが、コンポーネントモデルを持たず、これまでは、ロボットの制

Table 1 Comparison of robotic middleware

Name	HRTM	OpenRTM-aist	ROS	OROCOS
Composite component	Yes	Up to sub-component	No	No
FSM component	Yes	No	Yes (Python)	No
Supported OS	Linux Windows Mac OSX	Linux Windows Mac OSX	Linux	Linux Windows Mac OSX
Supported protocol	CORBA shmem bus	CORBA	TCP/IP UDP	TCP/IP UDP
Supported RTOS	Proprietary RTOS	ART-Linux	No	No
System modeling language	UML	Original block diagram	No	No
Component model	RTC	RTC	Original	Original

御機能開発のためのソフトウェアライブラリという位置付けであった。現在はスペインの PAL ROBOTICS 社の支援を受けて開発が続けられており、独自のコンポーネントモデルの追加など着実な進化を遂げている。

調査の結果、3 章で述べた要件を全て満たすミドルウェアが存在しないこと、国際標準コンポーネント仕様として OMG RTC が利用可能であることがわかった。そこで、コンポーネント仕様は国際標準の OMG RTC 準拠としつつ、必要要件を満たすことをねらいとして、独自のミドルウェア HRTM の開発を開始した。

Table 1 に HRTM と主なミドルウェアの機能比較表を示す。

5. 達成機能

ここでは、HRTM の特徴をその達成機能とともに説明する。

5.1. システム設計ツール

HRTM では国際標準のモデリング言語である Unified Modeling Language (UML)⁽¹²⁾ を用いた視覚的な設計ツールによるシステム設計を可能とした。

設計ツールでは、UML の複合構造図を RTC 仕様にあわせて拡張した図を用いて、コンポーネントの入出力ポートの種類や接続関係、実行順などのパラメータを定義することができる。なお、入出力データ型の不一致など、検出可能なエラーは設計ツールによって警告される。

また、状態遷移図を UML の statechart を用いて設計し、後述する状態遷移型コンポーネントの内部設計を行うことができる。Statechart は階層構造を持つ状態遷移図であり、作業やインタラクション機能の複雑な状態遷移ルールを見通しよく設計することが可能である。

新規のコンポーネントを作成する場合は、完成した UML 図から自動生成したスケルトンコードを利用する。データ通信などコンポーネント仕様に基づく共通機能はフレーム

ワークによって処理されるため、コンポーネント開発者は、スケルトンコードに内部処理の実装を追加するだけでよい。

5.2. コンポーネント仕様

5.2.1. 国際標準仕様準拠

HRTM が準拠している RTC はロボットアプリケーション用のコンポーネント仕様として定義されているが、その拡張性は高く、ロボット以外の適用例も存在する⁽¹³⁾。RTC 仕様に準拠したミドルウェアには、前述の OpenRTM-aist など複数の実装^{(14), (15)}があり、国内外の研究所や大学で利用されている。

HRTM と OpenRTM-aist のコンポーネントは RTC 仕様に基づき相互接続が可能であるため、外部研究機関との共同研究を行う際に ASIMO ソフトウェアシステムと既存のコンポーネントを容易に連携させることができる。

5.2.2. 階層型コンポーネント

前述の通り、大規模システムのコンポーネント化において、階層化はシステムの見通しを良くし、メンテナンス性の向上を実現するうえで重要な機能であるため、HRTM では、RTC 仕様を拡張した多階層型コンポーネント機能を実装している。

RTC 仕様に基づく階層型コンポーネントは下記の構成要素からなる (Fig. 2)。

1. 親コンポーネント (Super component)
2. サブコンポーネント (Sub component)
3. 公開ポート (Public port)
4. 実行スレッド (Execution thread)

親コンポーネントは内部に一つ以上のサブコンポーネントを含む。公開ポートは、サブコンポーネントが外部コンポーネントとの通信を行うために利用する、親コンポーネントのポートである。実行スレッドは、割り当てられた一つ以上のサブコンポーネントを実行する。

RTC 仕様は、階層化コンポーネントに含まれるサブコンポーネントの実行順序を指定する手段が明示していないため、HRTM では、独自機能として実行スレッドに対して、

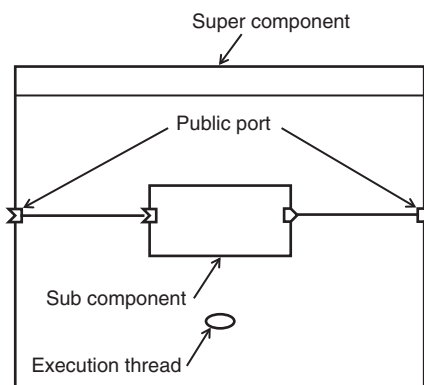


Fig. 2 Overview of a composite component

サブコンポーネントの実行順序をパラメータで指定できる順序実行スレッドを開発した。

5.2.3. 状態遷移型コンポーネント

知能ロボットシステムの作業やインタラクション機能の実装には、複雑な状態遷移が使用されており、それらのメンテナンス性向上はこれまでの課題であった。HRTM では、UML statechart による視覚的な状態遷移モデル設計を実現し、複雑な状態遷移を簡潔に表現できるようにした。

Fig. 3 に簡単な状態遷移モデルの例を示す。図中の黒丸は開始状態、角丸の矩形は状態、矢印は遷移を表す。各ラベルは対応する要素名を示している。Fig. 3 の状態遷移モデルはまず、開始状態から自動的に State A に遷移する。その後は遷移ルールに基づいて、状態が遷移していくことになる。例えば、初期状態から Event A, Event B, Event C が入力されると State B, State C, State A に遷移する。

Fig. 4 は Fig. 3 のモデルに遷移ルールを追加したモデルである。State A, State B 共に Event B によって State C に遷移する。このような状態間の共通遷移ルールは、実際のモデルにおいても、リセット処理などでよく使われている。しかし、状態や遷移ルールの数が増えると遷移ルールの付け忘れが発生しやすくなる。

UML statechart では状態を階層的に設計することで、サブ状態の共通遷移ルールをまとめることができる。Fig. 5 は Fig. 4 と同じモデルを階層化したモデルである。図中で、State A と State B は Super State のサブコンポーネントとして設計されている。UML statechart の仕様により、サブコンポーネントは親である Super State の状態遷移ルー

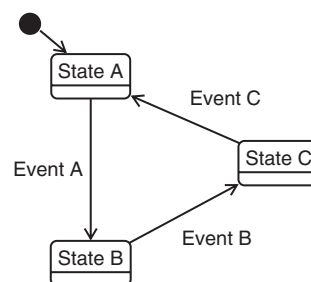


Fig. 3 UML statechart of simple finite state machine

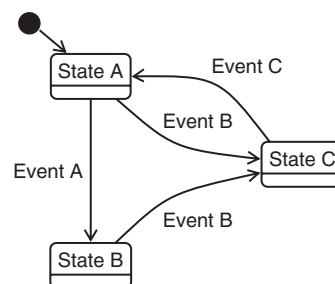


Fig. 4 UML statechart of finite-state machine in which different states have share transition rule

ルを引き継いでいるため、Event B による State C への遷移は Super State の遷移ルールとして、一つに統合された。Fig. 4 と Fig. 5 を比較すると、階層化設計によってモデルがより簡潔に記述できることがわかる。

設計した状態遷移モデルの実装は、状態遷移型コンポーネントのスケルトンコードに含まれるため、開発者が実装する必要はない。さらに、設計ツール上で状態遷移モデルのエラー検出が行われるため、不正な状態遷移モデルはスケルトンコードとして出力できない。

これらの機能によって、複雑な状態遷移のメンテナンスを容易にし、実装時の不具合を排除することができる。

5.3. フレームワーク

5.3.1. マルチプラットフォーム対応

コンポーネントを様々な OS 上で動作させることはフレームワークの必要要件であるため、リアルタイム OS を含めた複数の OS を隠ぺいする OS 抽象化層を OS とフレームワークの間に構築した (Fig. 6)。

この抽象化層を用いて作成したプログラムは、ソースコードを変更することなく、複数の OS 上でコンパイルおよび実行させることが可能であるため、移植性の高いマルチプラットフォーム対応フレームワークを実現することができた。

5.3.2. マルチ通信プロトコル対応

HRTM では、コンポーネント間の通信プロトコル指定用パラメータを、コンポーネントが持つデータベース内に格納して、実行時に選択する機能を実装した。

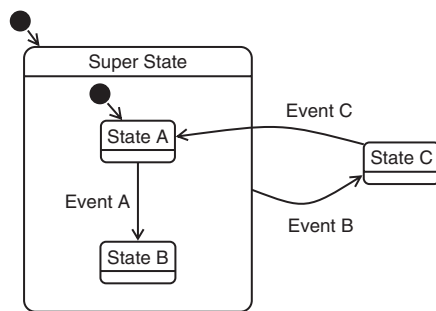


Fig. 5 UML statechart of the finite state machine in Fig. 4 with hierarchically nested states

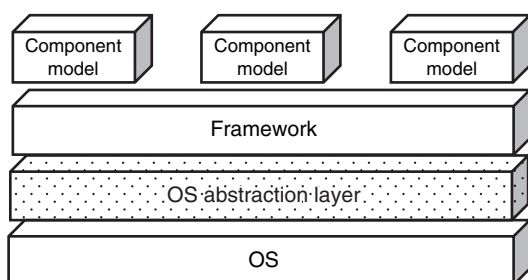


Fig. 6 OS abstraction layer of HRTM

Table 2 Processing time to send/receive 1000 bytes data [us]

	Data conversion	Data copy	Data receive	Data send	Total
Send	100	13	-	75	188
Receive	100	13	10	-	123

選択されたプロトコルによる通信処理は、フレームワークによって実行され、コンポーネントの実装は通信プロトコルに依存しないため、SILS, HILS, ターゲット環境間で利用できる通信プロトコルが異なっても、コンポーネントのソースコードを変更することなく再利用して実行することができる。

5.3.3. ハードリアルタイム性

二足歩行ロボットの制御システムは、ハードウェアの割り込み信号などの特定のタイミングに同期しながら、一定時間内に一定の順序で指令値の計算を完了するハードリアルタイム性を保証しなければならない。コンポーネントのハードリアルタイム性を保証するために、HRTM では優先度に基づいて動作する同期型順序実行スレッドの開発とフレームワークの実行処理時間の削減に取り組んだ。

同期型順序実行スレッドは、順序実行スレッドの機能に加えて、ハードウェア割り込みや内部クロックなどの同期元をパラメータで指定することが可能である。これによって、特定の周期内でコンポーネントの処理時間を最小化した実行順序を指定することができ、ハードリアルタイム性を保証できる。

フレームワークの処理時間削減は HRTM 開発において最も困難な技術課題であったため、詳細に説明する。開発の初期段階でコンポーネント実行時のフレームワークの処理時間を分析した結果、通信時間の割合が大きかったため、通信時間の削減を主な課題とした。通信時間の削減対策前に、1000 バイトのデータを TCP/IP 経由で送受信するために必要な処理時間を計測した結果を Table 2 に示す。

計測の結果、受信時は 123 μ s、送信時は 188 μ s の処理時間がかかることがわかった。これらの値は、ASIMO システムにおいて、ハードリアルタイムで動作するプログラムの平均的な処理時間に対して大きな割合を占めるため、データの入出力を伴う既存プログラムのコンポーネント化は、著しい処理時間の増加をもたらすことになる。

処理時間増加の主な要因は、RTC 仕様に基づく通信時間に含まれるデータのネットワーク形式への変換および逆変換の処理時間 100 μ s と、TCP/IP 経由のデータ送信時間 75 μ s である。データ送信時間は、通信プロトコルを TCP/IP より高速なものに置き換えることで短縮することが可能であるが、データ変換処理を省略してしまうと RTC 仕様に準拠できなくなってしまう。

そこで、HRTM では、RTC 仕様を拡張し、通信プロトコ

Table 3 Required processes and types of communication

	Send or receive	Data conversion	Data copy	Data receive	Data send
Network	Send	Yes	Yes	-	Yes
	Receive	Yes	Yes	Yes	-
Inter-process	Send	No	Yes	-	Yes
	Receive	No	Yes	Yes	-
Inner-process	Send	No	Yes (address)	-	Yes
	Receive	No	Yes (address)	Yes	-

Table 4 Processing time of intra-process communication [us]

	Data conversion	Data copy	Data receive	Data send	Total
Send	-	7	-	-	7
Receive	-	13	3	-	16

ルに加えて、ネットワーク、プロセス間、プロセス内の3種類の接続形態を選択できるようにした。この拡張によってRTC仕様のコンポーネントと接続する場合は、RTC仕様に準拠した通信手法を用い、HRTMのコンポーネント同士はデータ変換処理を省略したプロセス間通信や、データコピーをアドレス参照にしたプロセス内通信を利用できるようになり、通信時間の高速化を実現できた (Table 3)。

以上の対策により、最も高速なプロセス内コンポーネント間通信を用いた場合、通信処理時間は受信時が123 μ sから16 μ s、送信時が188 μ sから7 μ sに改善した (Table 4)。プロセス内の通信はプロセスメモリ空間のアドレス参照による通信のため、通信時間は確定的となり、ハードリアルタイム性を保証することが可能となった。

6. ASIMO への適用

ここでは、HRTMの機能実証例として、ASIMOシステムの経路追従歩行部に適用した結果について示す。経路追従歩行部はFig. 7に示すように、現在位置に対して、与えられた経路に沿った足平の着地位置を繰り返し計算し、経路への追従歩行を行うためのプログラムである。

6.1. 経路追従歩行部概要

経路追従歩行部の概要をFig. 8に示す。経路追従歩行部は、5 ms周期でタイマー割り込みを受けて、歩行状態の確

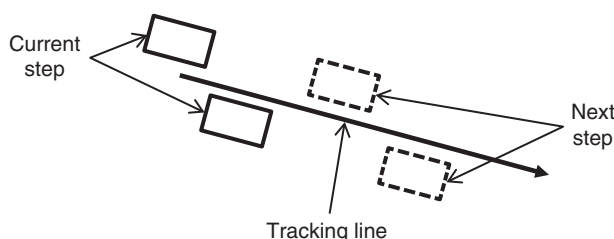


Fig. 7 Overview of path tracking walk

認と着地位置計算を繰り返しているため、通信性能を含むハードリアルタイム性の検証対象として適している。

6.2. 適用手順

経路追従歩行部へのミドルウェア適用手順は下記の通りである。

1. コンポーネント設計
2. SILS による動作確認
3. ターゲット環境上での動作確認

6.3. コンポーネント設計

経路追従歩行部のコンポーネント化前の構成をFig. 9に示す。経路追従のための着地位置決定処理は5 ms周期の割り込みシグナルを受け、Fig. 8の一連のステップを実行した後、計算結果を共有メモリを介して脚軌道制御部に渡す。

そして、同じく5 ms周期で動作する脚軌道制御部が、与えられた着地位置に足平を置くための軌道計算を行い、脚を移動することで、経路追従歩行が実現する。

コンポーネント化の方針は、着地位置決定処理部に実装されている状態遷移を、状態遷移型コンポーネントとしてモデル化することとした。そのため、着地位置決定処理を階層化し、内部を状態遷移部とロボット状態通知部に分割した (Fig. 10 青枠部分)。分割された状態遷移部と状態通知部は共に5 ms周期で動作するが、実行は状態通知、状態遷移の順でなければならないため、サブコンポーネントの実行順序をパラメータで設定した。実際に設計ツールで作成した着地位置決定部のモデルをFig. 11、状態遷移部のモデルをFig. 12に示す。

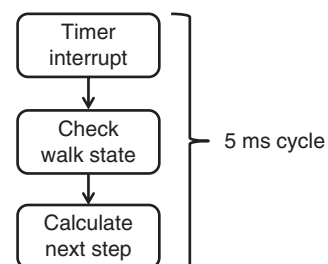


Fig. 8 Processing steps of path tracking walk

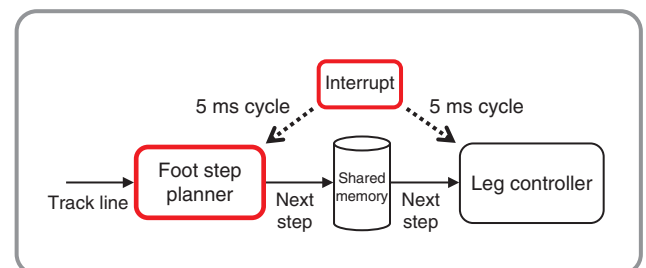


Fig. 9 Original architecture of path tracking walk (red line part)

6.4. SILS による動作確認

設計したモデルから、コンポーネントのスケルトンを自動生成し、コンポーネント化前のプログラムから SILS 環境用に機能を移植した。

完成したコンポーネントを用いて、SILS 環境上での動作をコンポーネント化前とコンポーネント化後で比較し、結果が完全に一致することを確認した。

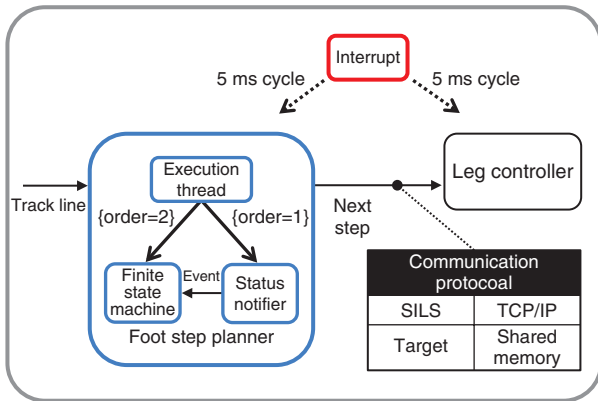


Fig. 10 Componentization of path tracking walk (blue line part)

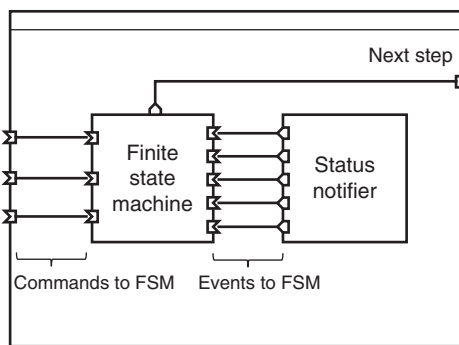


Fig. 11 UML diagram of foot step planner

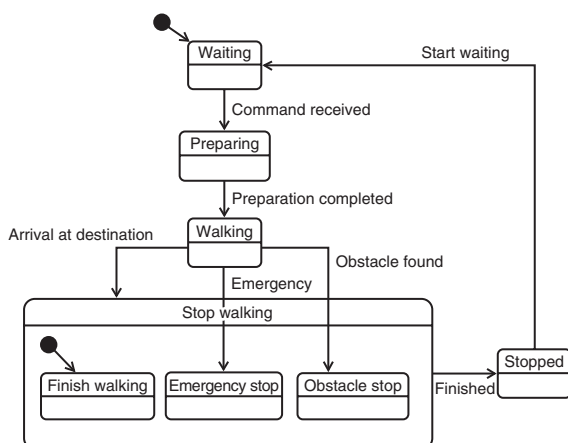


Fig. 12 UML statechart diagram of FSM component

Table 5 Comparison of processing time [us]

	Interruption processing	Input port processing	Walk status checking	Next position calculation	Total
Original	42	-	309		351
Component	22	80	50	280	432

6.5. ターゲット環境における動作確認

ASIMO 実機による動作確認時には、SILS で用いたコンポーネントソースコードを修正することなく、ターゲット環境でコンパイルし直すだけで実行することができた。

環境に依存して変化するパラメータは、全てファイルやコンポーネント内データベースに格納されており、コンポーネント再利用性を確認することができた。

6.5.1. 実行結果比較

実機による歩行テストの結果、コンポーネント化した経路追従歩行部は 5 ms 周期で動作し、状態遷移コンポーネントが設計通りに状態遷移していることを確認した。

さらに、コンポーネント化前後の脚軌道を比較し、両者が完全に一致することを確認した。

6.5.2. 処理時間計測結果

ハードリアルタイム性を満たしているかを検証するために、経路追従歩行部の処理時間をコンポーネント化前と後で計測した (Table 5)。コンポーネント化にあたり、元々のプログラムから、タイマー割り込み処理の実装が変更され、状態通知と着地位置計算が分離されたため、各部の処理時間を直接比較することは難しいが、コンポーネント化後に全体の処理時間が 81 μ s (23%) 増加している。

6.6. 考察

ASIMO の経路追従歩行部のコンポーネント化によって、設計したモデルから自動生成したコンポーネントのソースコードが修正することなく、SILS、ターゲットの両環境上で利用できることを確認することができた。

また、コンポーネント化によって、処理時間が 23% 増加した。今回の実機動作に対しては、許容可能な範囲であったが、より厳しい状況を考慮して処理時間増加分の削減手法について考察する。

コンポーネント設計の修正によって削減可能なのは、入力ポートの数である。Fig. 11 に示されているように、着地位置決定コンポーネントへの入力ポートの数は全部で 8 個あり、受信データサイズは 10 から 20 バイトの範囲である。個々の入力ポートは順番に受信処理が行われるため、ポート数と受信時間の積が合計の処理時間となる。計測の結果、プロセス内通信のデータ受信時間は、20 バイト受信時で 9.4 μ s という結果が得られており、8 ポート受信時の推定処理時間 $9.4 \times 8 = 75.2 \mu$ s と比較して Table 5 より得られる入力ポート処理時間の 80 μ s は妥当な値と考えられる。

今回のコンポーネント設計では、Fig. 11 のように指令の種類と状態通知の種類ごとにポートを作成したが、指令

と状態通知を一つのポートにまとめて設計することも可能である。この場合、Table 4 より 1000 バイトの受信時間は $16 \mu\text{s}$ であるため、処理時間の増加は $16 \times 2 = 32 \mu\text{s}$ (10%) 以下に抑えることができる。

7. 標準化活動

最後に HRTM に関する標準化活動について述べる。HRTM は RTC 仕様に対して下記の仕様を追加拡張して実装されている。

- 複数のプロトコルを扱えるデータポート仕様
- 状態遷移コンポーネントの現在状態通知および取得インターフェース仕様

これら拡張仕様は、OMG のロボティクス領域の仕様を策定する委員会である Robotics Domain Task Force に属する Infrastructure Working Group において、FSM4RTC 仕様として、2015 年中の公開仕様化を目標とした標準化作業を推進中である。

拡張仕様を標準化することで、RTC 準拠のミドルウェア間で、互換性のある高速コンポーネント間通信と状態遷移型コンポーネントを利用することができる。

8. まとめ

国際標準の RTC 仕様に準拠しつつ、独自の機能拡張を施したコンポーネント指向ミドルウェア HRTM を開発した。

また、HRTM フレームワークの性能を検証するため、ASIMO の経路追従歩行部をモデル化し、生成したスケルトンコードに既存のプログラムを移植した。検証の結果、コンポーネントが SILS 環境および実機ターゲット環境で再利用できることと、ハードリアルタイム性能を満たしていることを実証できた。

コンポーネント化は、設計ツールによって定義されたシステムモデルから生成したスケルトンコードに、既存のプログラムから通信処理などを除いたソースコードの一部を追加することで実現できるため、開発者が取り扱うソースコードの量が減少し、メンテナンス性や品質の向上が期待できる。

今後は、HRTM を用いた ASIMO の開発によるコンポーネント指向開発の効果の拡大を目指すとともに、拡張仕様である FSM4RTC の標準化を実現し、外部研究機関やロボット以外の領域への HRTM 適用範囲の拡大を図ってきたい。

参考文献

- (1) 広瀬真人, 竹中透: ヒューマノイドロボット ASIMO の開発, Honda R&D Technical Review, Vol.13, No.1, p.1-6

- (2) 重見聡史, 河口裕一郎, 吉池孝英, 川邊浩司, 小川直秀: 新型 ASIMO の開発, Honda R&D Technical Review, Vol.18, No.1, p.38-44
- (3) 重見聡史, 川邊浩司, 中村孝広: 新型 ASIMO の開発—自律機械の実現—, Honda R&D Technical Review, Vol.24, No.1, p.34-41
- (4) Ando, N., Sehiro, T., Kotoku, T.: A Software Platform for Component Based RT-System Development: OpenRTM-aist, International Conference on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS (SIMPACT 2008), p. 87-98 (2008)
- (5) <http://www.omg.org/>
- (6) <http://www.omg.org/spec/RTC>
- (7) <http://www.omg.org/spec/CORBA>
- (8) <http://www.ros.org/>
- (9) <http://www.dh.aist.go.jp/jp/research/assist/ART-Linux/>
- (10) <http://www.ros.org/repos/rep-0003.html>
- (11) Bruyninckx, H.: Open Robot Control Software: the OROCOS project, Proceedings of the 2001 IEEE International Conference on Robotics and Automation, Vol. 3, p. 2523-2528 (2001)
- (12) <http://www.omg.org/spec/UML>
- (13) <http://daqmw.kek.jp/>
- (14) <http://www.sec.co.jp/robot/openrtmnet/introduction.html>
- (15) <http://www.gostai.com/products/rtc/>

■ 著 者 ■



関谷 真



根本 教男



宮下 善太



大野 信之