組込みリアルタイムシステムにおける
メモリ保護機能対応プラットフォーム
および確率的応答時間解析

石川 拓也

概要

大規模・複雑化が進む組込みリアルタイムシステムの開発において，開発コストの低減や開発期間の短縮が求められている．ソフトウェアの開発に関して，開発効率向上のために，リアルタイムオペレーティングシステム（OS）や，組込みシステム向けのソフトウェアコンポーネント技術が提案されている．既存のコンポーネント技術は，アプリケーションソフトウェアの開発を支援するものが多いが，ミドルウェアやデバイスドライバなどのソフトウェアプラットフォームを開発できるものもある．ソフトウェアプラットフォームをコンポーネントベース開発することにより，システムのハードウェア構成やアプリケーションの要求に応じて，ソフトウェアプラットフォームの構成を容易に変更できる．また，近年では，安全性が求められるシステムのアプリケーション開発に適用するためのリアルタイム OS やコンポーネント技術が提案されている．

一方で，組込みリアルタイムシステムでは，システム内のタスクが，個々のデッドラインを満たすかどうかを検証することが重要である．従来のリアルタイムシステムにおけるスケジューラビリティ解析では，タスクの応答時間を解析する際に，タスクの最大実行時間など，一意的な値が用いられることが多い．しかし，これらの手法で仮定されている，実行時間が最大となるような状況は通常発生することはなく，これらの手法によって解析された結果はあまりに悲観的であるため，そのシステムに求められる性能を過大に見積もってしまい，開発コストの増加や開発期間の増加を起こしてしまう可能性がある．

本研究では，組込みリアルタイムシステムにおける開発効率の向上や開発コストの削減を目的として，大きく分けて，2 つの研究課題について取り組む．まず 1 つ目に，リアルタイム OS やソフトウェアコンポーネント技術などのアプリケーション開発支援技術についての課題に取り組む．この研究では，組込みシステム向けのコンポーネント技術である TECS を対象とし，TECS によりソフトウェアプラットフォームをコンポーネントベース開発することの有用性を示し，さらに，メモリ保護を考慮した組込みシステム向けのコンポーネント技術，HR-TECS を提案する．また，HR-TECS を実現するために，メモリ保護機能を持ったリアルタイム OS，TOPPERS/HRP2 カーネル（HRP2 カーネル）を提案する．そして 2 つ目に，リアルタイムシステムにおける応答時間解析技術について，悲観的な解析結果を改善し，リアルタイムシステムの開発コストや開発期間の増加を抑えるための課題に取り組む．この研究では，周期タスクで構成されるシステムを対象とし，タスクの応答時間分布を解析する手法を提案する．

既存のコンポーネント技術は，アプリケーションソフトウェアの開発を支援するものが多いが，ミドルウェアやデバイスドライバなどのソフトウェアプラットフォームを開発できるものもあると述べた．本研究で対象とした組込みシステム向けのコンポーネント技術 TECS では，実行時間やメモリ使用量のオーバヘッドが小さく，デバイスドライバのようなソフトウェアの細部までコンポーネント化できるとされているが，実際に TECS によりプラットフォームを開発し，その有用性を評価することはされていない．

そこで本研究では，組込みシステムにおけるソフトウェアプラットフォーム開発において TECS を適用することの有効性を示した．そのために，組込みシステム教材として用いられている LEGO 社製 MindstormsNXT におけるソフトウェアプラットフォームを，TECS コンポーネント技術により開発した事例について述べ，TECS が，ソフトウェアプラットフォームの開発に有用であることを併せて示した．そして，プ

ラットフォームをコンポーネント化することにより，容易にプラットフォームの構成を変更でき，無駄なメモリ使用量や実行時間を削減できることを評価により示した．

一方で，近年，高い安全性が求められる組込み制御システムの開発では，安全に関する国際規格に準拠することが求められている．これらの国際規格に準拠してソフトウェアを開発する場合，その設計やテストに対して厳格な検証が求められており，ソフトウェアの開発コストが高くなり，開発期間も増大する．ソフトウェアの開発コストを低減し，開発期間を短縮するためには，パーティショニング機構を用いて，高い安全性が求められるソフトウェア部分をできる限り局所化し，厳格な検証が必要な部分を少なくすることが有効である．そのため，安全性が求められる組込みソフトウェアの開発効率を向上するために，パーティショニング機構を提供するリアルタイム OS が提案されている．

パーティショニング機構の重要な機能の 1 つとして，メモリ保護機能がある．メモリ保護を実現するためには，メモリ管理ユニット（MMU）やメモリ保護ユニット（MPU）といった，専用ハードウェアが用いられることが多い．しかしながら，厳しいリソース制約やハードリアルタイム性が要求される組込みリアルタイムシステムでは，MMU は適さない．一方，MPU は，メモリ保護機能の実現を支援するが，MMU と異なり TLB のようなキャッシュを必要とせず，MPU の設定を切り替えるための時間は固定であるため，リアルタイム性の保証がしやすい．ハードリアルタイムシステムでは，メモリ保護のために MPU を用いる場合があるが，現在文献等で公表がされており，MPU をサポートするリアルタイム OS では，MPU の隠蔽が不十分であり，アプリケーション開発の生産性が低いという問題がある．

そこで本研究では，MPU を用いたメモリ保護機能を提供するリアルタイム OS，HRP2 カーネルを提案した．HRP2 カーネルでは，静的コンフィギュレーションにより，メモリ配置を静的に行う．そして，MPU を用いたメモリ保護を実現するとともに，メモリ保護に必要な情報を静的に生成し，RAM 使用量のオーバヘッドを抑える

ことを可能とした．本研究では，SH2A と ARM Cortex-M3 の MPU を用いて，HRP2 カーネルを実現する方法を示した．そして，HRP2 カーネルの実用性を確かめるために，メモリ保護機能を持つことによって生じる実行時間やメモリ使用量のオーバヘッドを，メモリ保護機能を持たないリアルタイム OS と比較評価し，実行時間の予測が可能であること，RAM 使用量のオーバヘッドが小さいことを示した．

また，組込みシステム向けのコンポーネント技術において，メモリ保護を考慮したソフトウェアのコンポーネントベース開発が可能な技術が提案されている．メモリ保護を考慮したコンポーネント技術は，メモリ保護機能を提供するリアルタイム OS をベースとして用い，コンポーネントに対するアクセス権やコンポーネントのパーティションへの割当てを静的に指定することで，リアルタイム OS のみを用いる場合と比較して，高い抽象度でのソフトウェア開発を可能としている．また，異なるパーティションに属するコンポーネント間の通信処理についても隠蔽することができるため，パーティションへの配置を意識することなくコンポーネントを開発でき，ソフトウェアコンポーネントの再利用性を維持できる．しかしながら，既存のコンポーネント技術で，組込みシステムに適した実装や評価がなされており，かつ，ソフトウェアプラットフォームを開発することが考慮されているものはない．

そこで本研究では，メモリ保護を考慮した組込みシステム向けのコンポーネント技術，HR-TECS を提案した．HR-TECS では，TECS コンポーネント技術を拡張することで，メモリ保護を考慮したコンポーネントベース開発を可能とし，また，HRP2 カーネルの機能を利用することで，メモリ保護機能を実現した．そのために，HR-TECS のコンポーネント記述から，HRP2 カーネルの設定ファイルを自動生成する．コンポーネント間の通信について，通信処理本体をコンポーネント記述から自動生成することで，個々のコンポーネントにおける通信 API の記述は，配置されるパーティションに依存せず，共通の API で開発することができる．また，HR-TECS は，非特権モードで動作するアプリケーションだけでなく，特権モードで動作するミドルウェアやデバ

イスドライバなどのプラットフォームの開発にも利用できる．そして，評価実験により，実行時間のオーバヘッドが小さく，かつ，予測が可能であること，メモリ使用量のオーバヘッドが小さいことを示した．

一方で，リアルタイムシステムにおける応答解析において，悲観的な解析結果を改善するために，タスクの実行時間を確率変数として扱い，リアルタイムシステムの応答時間を確率的に解析する手法が提案されている．この手法では，タスクの実行時間のみを確率変数として扱い，タスクの応答時間分布を数学的に解析する手法を提案している．ここで，確率変数を用いて解析を行う場合，解析手法が複雑になり，解析時間が長くなってしまうことが問題とされる．そのため，複雑さを軽減し，短時間で解析可能な手法として，応答時間分布を悲観的に近似する手法が提案されている．しかしながら，これまでに提案されている解析手法では，周期タスクの初期位相が確率変数である場合が考慮されておらず，初期位相が定数であるタスクセットしか解析することができない．

そこで本研究では，初期位相分布を考慮した，周期タスクの応答時間分布を解析する手法を提案した．タスクの初期位相を確率変数として扱う場合，各タスクの初期位相が取りうる値の任意の組合せに対して，応答時間分布を解析する必要があり，解析時間が長くなると考えられる．そこで，本論文で提案する手法では，タスクの実行時間分布と初期位相分布を悲観的に離散化し，さらに，応答時間分布の終端部分に着目して数学的に解析することで，応答時間分布の解析時間を短縮した．モンテカルロシミュレーション結果との比較評価により，提案手法では，応答時間分布を悲観的に近似でき，かつ，高速に解析できることを示した．

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

序論

1.1 研究の背景

大規模・複雑化が進む組込みリアルタイムシステムの開発において，システムの開発コストの低減や開発期間の短縮が求められている．組込みシステムにおけるソフトウェアの開発に関しては，開発効率向上のために，リアルタイムオペレーティングシステム（OS）が使用されることが多い．リアルタイム OS は，プロセッサ，メモリ，周辺デバイスなどのハードウェアを隠蔽，抽象化して，アプリケーションソフトウェアに提供する．また，近年では，組込みシステム向けのソフトウェアコンポーネント技術が提案されている [38] [21] [6] [31] [11]．コンポーネント技術とは，ソフトウェアを機能ごとに分割したソフトウェア部品（コンポーネント）を組み合わせることによってソフトウェアを開発することで，開発を効率化させる技術である．既存のコンポーネント技術は，アプリケーションソフトウェアの開発を支援するものが多いが，ミドルウェアやデバイスドライバなどのソフトウェアプラットフォームを開発できるものもある [38][54]．組込みリアルタイムシステムでは，メモリやデバイスなどのハードウェア資源が少なく，また，システムによってハードウェア構成やアプリケーションが要求する機能が異なることが多いため，ソフトウェアプラットフォームも開発対

象となることが多い．よって，組込みリアルタイムシステムでは，アプリケーションソフトウェアだけでなく，ソフトウェアプラットフォームの開発にも利用できるコンポーネント技術が重要である．

一方で，組込みリアルタイムシステムでは，処理を終える時刻の正確さが重要な要素である．そこで，リアルタイムシステムの設計においては，満たすべき時間制限（以降，デッドラインと呼ぶ）を決める必要がある．一般的に，デッドラインは，タスク毎に決められており，タスクは，決められたデッドラインまでに実行を終えることを要求される．よって，リアルタイムシステムの設計において，システム内のタスクが，個々のデッドラインを満たすかどうかを検証すること（スケジューラビリティ解析と呼ばれる）は，重要なプロセスである．従来のリアルタイムシステムにおけるスケジューラビリティ解析では，タスクの応答時間を解析する際に，タスクの最大実行時間など，一意的な値が用いられることが多い．このような解析手法は，短時間での解析が可能であり，文献 [37] や文献 [34] をはじめとして，現在までに多くの手法が提案されている．しかし，これらの手法で仮定されている，実行時間が最大となるような状況は通常発生することはなく，これらの手法によって解析された結果はあまりに悲観的である [39][63]．応答時間の解析結果が悲観的である場合，そのシステムに求められる性能を過大に見積もってしまい，開発コストの増加や開発期間の増加を起こしてしまう可能性がある．

本研究では，組込みリアルタイムシステムにおける開発効率の向上や開発コストの削減を目的として，大きく分けて，2 つの研究課題について取り組む．まず 1 つ目に，安全性が求められるシステムを対象とし，そのアプリケーションソフトウェアを効率的に開発するためのリアルタイム OS やソフトウェアコンポーネント技術についての課題に取り組む．そして 2 つ目に，リアルタイムシステムにおける応答時間解析について，悲観的な解析結果を改善し，リアルタイムシステムの開発コストや開発期間の増加を抑えるための課題に取り組む．

航空機や自動車などの組込み制御システムにおいては，システムの安全性が求めら

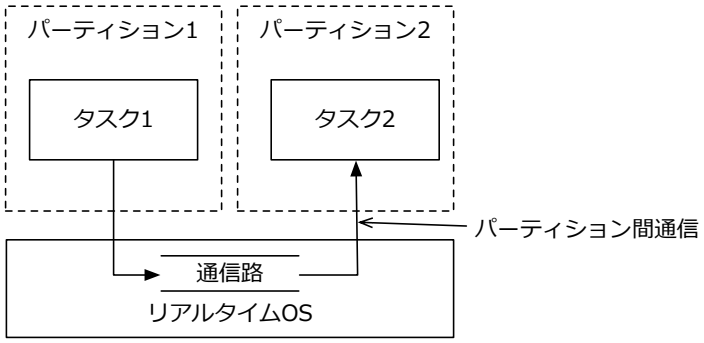

Figure 1.1: パーティション間通信

れる．近年，高い安全性が求められる組込み制御システムの開発では，IEC61508[26] や ISO26262[27]，DO178C[51] といった，電気・電子システムの安全に関する国際規格に準拠することが求められている [70]．これらの国際規格に準拠してソフトウェアを開発する場合，その設計やテストに対して厳格な検証が求められており，ソフトウェアの開発コストが高くなり，開発期間も増大する．ソフトウェアの開発コストを低減し，開発期間を短縮するためには，高い安全性が求められるソフトウェア部分をできる限り局所化し，厳格な検証が必要な部分を少なくすることが有効である．しかし，異なる安全性のレベルのソフトウェアが，単一のプロセッサ上で混在するシステムを構築する場合，低い安全性のソフトウェアの故障が，高い安全性のソフトウェアの故障に影響することがないように，ソフトウェアのパーティショニング機構を入れることが求められる．パーティショニング機構は，ソフトウェアをいくつかのパーティションに分割し，各パーティションに割り当てられた CPU 時間やメモリが，他のパーティションのソフトウェアから干渉を受けないように保護する．そのため，安全性が求められる組込みソフトウェアの開発効率を向上するために，パーティショニング機構を提供するリアルタイム OS が提案されている [1] [35]．また，パーティショニング機構を提供するリアルタイム OS は，異なるパーティションに属するソフトウェア間での通信を行うための機能（図 1.1）を提供していることが多い．

パーティショニング機構の重要な機能の 1 つとして，メモリ保護機能がある [16] .

メモリ保護機能は，各パーティションに対して割り当てられたメモリ領域に対し，異なるパーティションから不正なアクセスがないかどうかを監視し，メモリ空間に対するパーティション間の干渉を防止するための機能である．ここで，メモリ領域に対するアクセスを監視するためには，メモリ管理ユニット（MMU）やメモリ保護ユニット（MPU）といった，専用ハードウェアが用いられることが多い．これらの専用ハードウェアを抽象化し，メモリ保護が必要なソフトウェアの開発を効率化するために，メモリ保護機能を持ったリアルタイム OS が提案，開発されている [17] [7] [23]．汎用システムでは，メモリ保護機能を実現するために，MMU を用いることが多い．組込みシステム向けのプロセッサでは，MMU を搭載するものもあるが，リソース制約が厳しく，MMU を搭載できないものもある．また，MMU を使用してメモリ保護を実現する場合，ページテーブルやセグメントテーブルをソフトウェアで持つ必要があり，メモリ使用量のオーバヘッドが大きい．さらに，MMU を用いる場合には，アドレス変換を高速化するために，TLB（Translation Lookaside Buffer）という，アドレス変換の結果を保持しておくキャッシュを用いるが，TLB ミスの発生を予測することは困難であり，リアルタイム性の保証が困難となる．これらの理由から，厳しいリソース制約やハードリアルタイム性が要求される組込みリアルタイムシステムでは，MMU は適さない [13] [30] [15].

組込みシステム向けプロセッサには，MMU ではなく，メモリプロテクションユニット（MPU）というハードウェアを搭載したものが存在する [57][56][49][50]．MPU は，メモリ保護機能の実現を支援するが，MMU と異なり，アドレス変換は行わず，TLB のようなキャッシュを必要としない．また，MPU の設定を切り替えるための時間は固定であるため，リアルタイム性の保証がしやすい．MPU は，アドレス空間をいくつかの領域に分け，その領域ごとにアクセス権を設定可能である．MPU で扱うことのできる領域の数は有限であり，多くの場合，8 領域程度である．そのため，同じアクセス権を設定する必要のあるコードやデータを，連続した番地に配置するように，静的にメモリ配置を行い，MPU に設定するメモリ領域の数を，MPU が扱うこと

のできる領域数に収まるようにする必要がある．ハードリアルタイムシステムでは，メモリ保護のために MPU を用いる場合があるが，MPU をサポートするリアルタイム OS は少ない．また，現在文献等で公表がされていて，MPU をサポートするリアルタイム OS では，MPU の隠蔽が不十分であり，アプリケーション開発の生産性が低いという問題がある．

組込みシステム向けのコンポーネント技術において，メモリ保護を考慮したソフトウェアのコンポーネントベース開発が可能な技術として，ARINC-653 Component Model や CAmkES，AUTOSAR のコンポーネント技術が提案されている．メモリ保護を考慮したコンポーネント技術は，メモリ保護機能を提供するリアルタイム OS をベースとして用いることでメモリ保護機能を実現している．そして，コンポーネントに対するアクセス権やコンポーネントのパーティションへの割当てを静的に指定し，その設定からリアルタイム OS においてメモリ保護を実施するために必要な情報を生成することで，リアルタイム OS のみを用いる場合と比較して，高い抽象度でのソフトウェア開発を可能としている．また，異なるパーティションに属するコンポーネント間の通信処理についても隠蔽することができ，個々のコンポーネントにおいては，共通の API でコンポーネント間の通信を開発できるため，パーティションへの配置を意識することなくコンポーネントを開発でき，コンポーネントの再利用性を維持できる．しかしながら，既存のメモリ保護を考慮したコンポーネント技術では，組込みシステムに適した実装や評価がされておらず，また，ソフトウェアプラットフォームを開発することが考慮されているものはない．

一方で，リアルタイムシステムにおける応答解析において，悲観的な解析結果を改善するために，タスクの実行時間を確率変数として扱い，リアルタイムシステムの応答時間を確率的に解析する手法が提案されている．文献 [19]，[28] では，単一のプロセッサ内において，タスクの実行時間のみを確率変数として扱い，プリエンプティブな固定優先度ベースのスケジューリングのもとで，タスクの応答時間分布を数学的に解析する手法を提案している．そして，文献 [62]，[63] では，文献 [19]，[28] で提

案されている手法をベースとして，分散リアルタイムシステムにおける端点間処理の応答時間分布を数学的に解析する手法を提案している．ここで，確率変数を用いて解析を行う場合，解析手法が複雑になり，解析時間が長くなってしまうことが問題とされる．そのため，複雑さを軽減し，短時間で解析可能な手法として，応答時間分布を悲観的に近似する手法が提案されている [20][39]．しかしながら，これまでに提案されている解析手法では，対象とするタスクセットのモデルにおいて，周期タスクの初期位相が確率変数である場合が考慮されておらず，初期位相が定数であるタスクセットしか解析することができない．例えば，エンジン制御のアプリケーションにおいて，周期が一定のタスクと，エンジンの回転に同期して周期の変動するタスクとが混在する場合，タスク間の位相差がランダムな値となる．

1.2 研究の概要

本研究では，組込みシステム向けのコンポーネント技術である，TECS（TOPPERS Embedded Component System）[65] に基づき，メモリ保護を考慮した組込みシステム向けのコンポーネント技術，HR-TECS（High Reliable TECS）を提案する．さらに，HR-TECS を実現するために，メモリ保護機能を持ったリアルタイム OS，TOPPERS/HRP2 カーネル（HRP2 カーネル：High Reliable system Profile カーネル version 2）を提案する．HRP2 カーネルは，MPU を用いたメモリ保護の実現と，静的メモリ配置のサポートが可能な，メモリ保護機能を持ったリアルタイム OS である．TECS と HRP2 カーネル，HR-TECS の関係を，図 1.2 に示す．さらに，本論文では，周期タスクの応答時間分布を解析する手法を提案する．提案する手法では，タスクの実行時間に加えて，初期位相を確率変数として扱い，タスクの応答時間分布を解析する．

本論文の具体的な内容は，次の 4 つである．

ソフトウェアプラットフォームのコンポーネントベース開発

1 つ目は，組込みシステムにおけるソフトウェアプラットフォーム開発におい

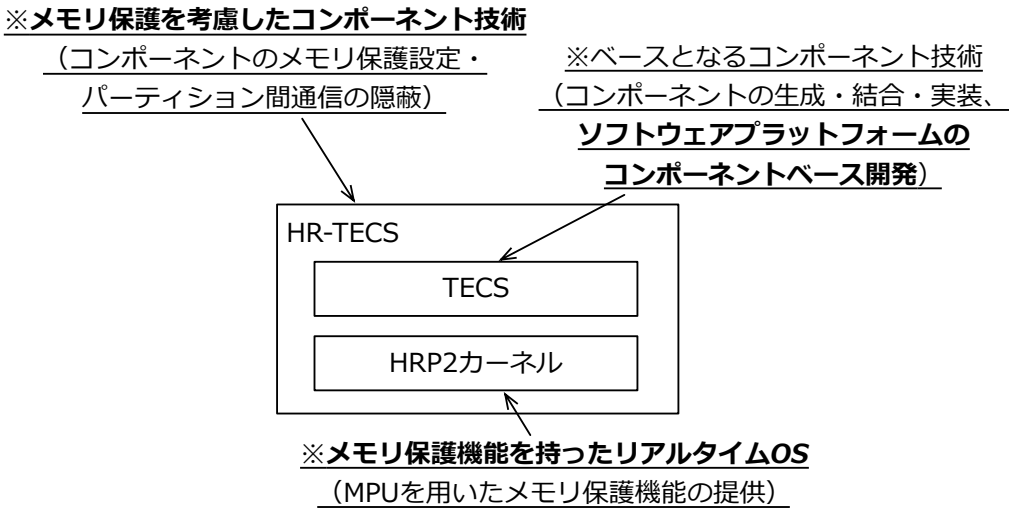

Figure 1.2: TECS，HRP2 カーネル，HR-TECS の関係

てコンポーネント技術を適用することの有効性を示すことである．そのために，組込みシステム教材として用いられている LEGO 社製 MindstormsNXT [33] におけるソフトウェアプラットフォームを，TECS を用いて開発した事例，および，評価を行った結果について述べる．また，本事例により，TECS がソフトウェアプラットフォームの開発に有用であることも併せて示す．この研究による貢献は，以下のとおりである．

(1-1) TECS によりソフトウェアプラットフォームを開発する事例を示す．

(1-2) ソフトウェアプラットフォームをコンポーネント技術を用いて開発することの利点を明らかにする．

メモリ保護機能を持ったリアルタイム OS

2 つ目は，MPU を用いたメモリ保護の実現と，静的メモリ配置のサポートが可能な，メモリ保護機能を持ったリアルタイム OS，HRP2 カーネルを提案することである．HRP2 カーネルでは，静的コンフィギュレーションにより，メモリ配置を静的に行う．そして，MPU を用いたメモリ保護を実現するとともに，メモ

リ保護に必要な情報を静的に生成し，RAM 使用量のオーバヘッドを抑えることを可能とする．HRP2 カーネルの実用性を確かめるために，メモリ保護機能を持つことによって生じる実行時間やメモリ使用量のオーバヘッドを，メモリ保護機能を持たないリアルタイム OS と比較することで評価する．この研究による貢献は，以下のとおりである．

(2-1) ソフトウェアのコードやデータに対するメモリ保護の設定を静的に行い，それらのメモリ配置を静的に決定する方法を示す．

(2-2) MPU を抽象化してメモリ保護機能を提供する方法を示す．

(2-3) 提案するリアルタイム OS の実現方法を示し，メモリ使用量と実行時間のオーバヘッドを明らかにする．

メモリ保護を考慮したコンポーネント技術

3 つ目は，メモリ保護を考慮した組込みシステム向けのコンポーネント技術，HR-TECS を提案することである．HR-TECS では，TECS コンポーネントモデルを拡張することで，メモリ保護を考慮したコンポーネントベース開発を可能とし，また，HRP2 カーネルの機能を利用することで，メモリ保護機能を実現する．そのために，HR-TECS のコンポーネント記述から，HRP2 カーネルの静的コンフィギュレーションのための設定ファイルを自動生成する．コンポーネント間の通信について，通信処理本体をコンポーネント記述から自動生成することで，個々のコンポーネントにおける通信 API の記述は，配置されるパーティションに依存せず，共通の API で開発することができる．また，HR-TECS は，ソフトウェアプラットフォームの開発にも利用できる．この研究による貢献は，以下のとおりである．

(3-1) メモリ保護を考慮したコンポーネント記述方法を提案する．

(3-2) コンポーネントの所属するパーティションに依存しない，コンポーネント

間通信を実現する方法を示す.

(3-3) 提案するコンポーネント技術の実現方法を示し，メモリ使用量と実行時間のオーバヘッドを明らかにする.

周期タスクにおける応答時間の確率的解析

4 つ目は，初期位相分布を考慮した，周期タスクの応答時間分布を解析する手法を提案することである．タスクの初期位相を確率変数として扱う場合，各タスクの初期位相が取りうる値の任意の組合せに対して，応答時間分布を解析する必要があり，解析時間が長くなると考えられる．そこで，本論文で提案する手法では，(1) タスクの実行時間分布と初期位相分布を離散化すること，および，(2) 応答時間分布の終端部分に着目して統計的に解析することで，応答時間分布の解析時間を短縮する．(1) については，まず，タスクの実行時間分布を離散化することにより，応答時間分布を数値解析で近似的に求めることができる．そして，初期位相分布を離散化することにより，応答時間分布を解析する初期位相の組合せを有限とすることができる．タスクの実行時間分布と初期位相分布の離散化については，離散化した確率分布を用いて解析した応答時間分布が，正確な応答時間分布よりも悲観的に近似されるように離散化誤差を丸める．(2) については，応答時間分布の終端部分に着目して解析することによって，解析する初期位相の組合せ，および，解析する応答時間分布の区間を限定する．ここで，応答時間分布の終端部分とは，最大応答時間に近い区間を指す（図 1.3）．リアルタイムシステムの開発では，応答時間がデッドラインを越えるかどうかを検証することが重要であり，応答時間が長い区間について，確率分布を求めることが重要であると考える．そして，応答時間分布の終端部分は，確率密度が低く，シミュレータなどにより分布を測定する場合，解析時間が長くなってしまう．そこで，応答時間分布の終端部分に着目して，統計的に解析する手法を提案することを考えた．この研究による貢献は，以下のとおりである.

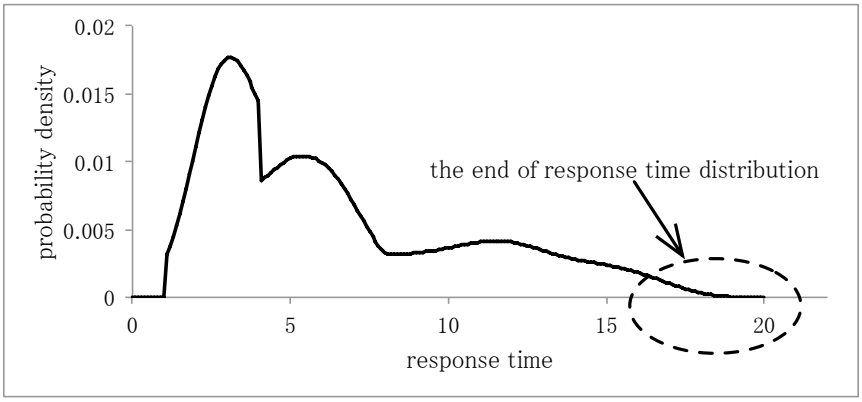

Figure 1.3: 応答時間分布の終端部分

(4-1) 周期タスクの初期位相と実行時間を悲観的に離散化する方法を示す.

(4-2) 周期タスクの応答時間分布の終端部分を，初期位相と実行時間の確率分布から高速に解析する方法を示す.

1.3 論文の構成

本論文の構成は，次のとおりである．まず，第 1 章では，本研究の背景と概要について述べた．第 2 章では，既存のリアルタイム OS とソフトウェアコンポーネント技術について述べた後，組込みシステム向けのソフトウェアコンポーネント技術である TECS について述べる．第 3 章では，組込みシステム向けのソフトウェアプラットフォームを，TECS を用いてコンポーネントベース開発した事例について述べる．そして，ソフトウェアプラットフォームをコンポーネント技術により開発することの有効性，および，TECS がソフトウェアプラットフォーム開発に有用であることを示す．第 4 章では，メモリ保護機能を持ったリアルタイム OS である HRP2 カーネルの仕様，および，実装方法について述べ，実行時間やメモリ使用量のオーバヘッドを評価する．第 5 章では，メモリ保護を考慮したコンポーネント技術である，HR-TECS について，その要件を整理し，TECS からの拡張点，および，HRP2 カーネルを利用した実現方

法について述べ，実行時間やメモリ使用量のオーバヘッドを評価する．第 6 章では，周期タスクの実行時間分布と初期位相分布を離散化する手法について述べ，周期タスクの応答時間分布を解析する手法について述べる．最後に，第 7 章で，本研究の結論と今後の課題について述べる．

CHAPTER 2

リアルタイムOSとソフトウェアコンポーネント技術

2.1　リアルタイムOS

組込みシステムにおけるソフトウェアの開発では，開発効率向上のために，リアルタイムOSが使用されることが多い．リアルタイムOSは，汎用システムにおけるOSのように，複数の処理単位を切り替えながらシステムを実行したり，処理単位間で同期や通信を行う機能を提供する．また，メモリ，周辺デバイスなどのハードウェアを隠蔽，抽象化して，アプリケーションに提供する．リアルタイムOSでは，汎用システムにおけるOSとは異なり，処理単位の実行にかかる時間を予測でき，リアルタイムシステムで求められる時間制約を満たすかどうかを検証できることが重要である．また，組込みシステムでは，システムに求められるアプリケーションの構成や機能が静的に決定されることが多いため，アプリケーションで使用されるリアルタイムOSのリソース（タスクやセマフォなどのカーネルオブジェクト）を静的に生成し，そのために使用するメモリ領域を静的に確保できることも，汎用システムのOSとの違いである．また，組込みシステムでは，使用できるメモリ使用量，特にRAM使用

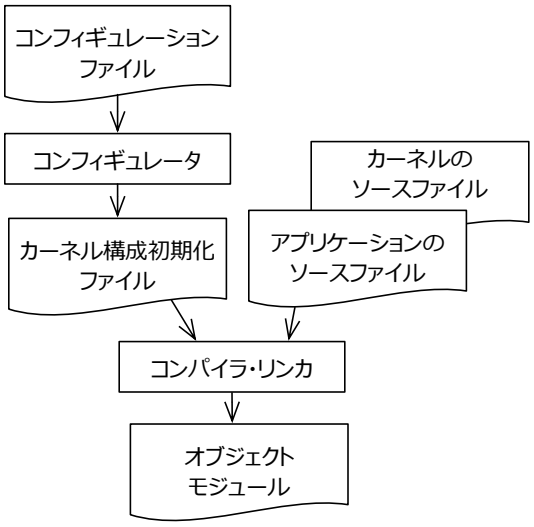

Figure 2.1: ASP カーネルにおける静的コンフィギュレーション

量の制約が厳しく，カーネルオブジェクトを静的に生成することで RAM 使用量を抑え，さらに，必要なデバイスドライバのみをリアルタイム OS に含めることなどにより，メモリ使用量を小さくすることが重要である．

カーネルオブジェクトを静的に生成するリアルタイム OS として，TOPPERS/ASP カーネル（以下，ASP カーネル）や AUTOSAR OS がある．

ASP カーネル [58] は，μ ITRON4.0 仕様 [64] をベースとした TOPPERS 新世代カーネル統合仕様 [60] に準拠したリアルタイム OS である．ASP カーネルでは，カーネルオブジェクトの生成情報や初期状態などを，静的 API と呼ばれるインタフェースにより静的に生成する．カーネルオブジェクトを生成するための静的 API の呼出しが記述されたファイルは，コンフィギュレーションファイルと呼ばれ，このコンフィギュレーションファイルをコンフィギュレータと呼ばれるツールにより解釈し，コンフィギュレータは，タスクのスタック領域や起動番地など，カーネルオブジェクトを扱うために必要となる情報を，ファイルに出力する．ここで，ASP カーネルを用いたソフトウェア開発における，オブジェクトモジュールの生成手順の概略を，図 2.1 に示す．まず，アプリケーション開発者は，コンフィギュレーションファイルにタスク

などのカーネルオブジェクトの生成情報や初期状態を記述する．次に，コンフィギュレーションファイルを，コンフィギュレータに入力し，カーネル構成初期化ファイルを生成する．ここで，カーネル構成初期化ファイルとは，タスクのスタック領域や起動番地など，カーネルオブジェクトを管理するための情報を出力するファイルである．最後に，カーネル構成初期化ファイルと，カーネルのソースコード，アプリケーションのソースコードをコンパイルし，それらのオブジェクトファイル群をリンクしてオブジェクトモジュールを生成する．上記のような，オブジェクトモジュールを得るまでの一連の流れを，静的コンフィギュレーションと呼ぶ．

AUTOSAR[6] は，自動車向けソフトウェアプラットフォームの標準仕様を策定している団体である．AUTOSAR の定めるソフトウェアプラットフォーム仕様において，リアルタイム OS （AUTOSAR OS）の仕様が定義されている [7]．AUTOSAR OS では，カーネルオブジェクトの生成情報を，独自の XML により記述する．そして，この XML 記述から，カーネルオブジェクトを管理するためのソースコードを生成することで，カーネルオブジェクトの静的生成を実現する．AUTOSAR OS を用いたアプリケーション開発は，カーネルオブジェクトを生成するための記述方法が異なるものの，ASP カーネルとほぼ同じ静的コンフィギュレーションの流れで行われる．

2.2 ソフトウェアコンポーネント技術

ソフトウェアコンポーネント技術は，ソフトウェアのコンポーネントベース開発を支援するものである．ソフトウェアのコンポーネントベース開発とは，ソフトウェアをコンポーネント単位で開発し，それらのコンポーネントを組み合わせて，ソフトウェア全体を開発することである．コンポーネントは，明確に定義されたインタフェース集合を提供，要求する，ソフトウェアの構成単位である．ここで，個々のコンポーネントは，自身の持つインタフェースを介して，他のコンポーネントが提供する機能を利用するため，他のコンポーネントに依存せずに，自身の提供する機能を開

発でき，作業分担が容易となる．さらに，インタフェースが明確に定義されているため，コンポーネントの再利用や，コンポーネントの取替えや修正による機能変更が容易にできる．また，ソフトウェアの機能を分割したコンポーネントを組み合わせてソフトウェア全体を構成することで，ソフトウェア全体を可視化し，ソフトウェア構造の理解しやすさを向上でき，抽象度の高いソフトウェア設計ができる．これらの利点から，ソフトウェアのコンポーネントベース開発により，開発コストの低減や，開発期間の短縮ができる．

コンポーネントベース開発を支援するコンポーネント技術として，汎用システムの分野では，COM[42] や CORBA コンポーネントモデル [46] などがある．汎用システムのコンポーネント技術では，システムの実行中に，動的にコンポーネントのインスタンス化（生成）や結合が行われる．しかしながら，組込みシステムでは，システムの実行前に，あらかじめ機能が定義されていることが多く，コンポーネントを静的に生成，結合できることが多い．そのため，動的にコンポーネントを生成や結合する方法では，実行時間やハードウェアリソース消費に無駄が生じる可能性がある．さらに，組込みシステムでは，時間制約やハードウェアリソース制約が厳しいため，実行時間やハードウェアリソース消費の無駄はできる限り避ける必要がある．よって，汎用システムのコンポーネント技術を，組込みシステム開発に適用することは困難である．

2.3 組込みシステム向けのコンポーネント技術

これまでに，組込みシステム開発に適用することを目的とした，ソフトウェアのコンポーネント技術が提案されている．さらに，近年では，安全性が求められる組込みシステム開発を支援するために，メモリ保護を考慮したソフトウェアの開発が可能なコンポーネント技術も提案されている．

アーキテクチャ記述言語 [41] である，AADL[52] や SysML[47]，MARTE[45] は，

コンポーネントモデルにより，ソフトウェアを含めたシステム全体をモデルで記述することができる．しかしながら，これらのアーキテクチャ記述言語の対象とする範囲は，V 字開発プロセスにおける，アーキテクチャ設計までの上流設計工程であり，ソフトウェアの実装方法までは規定されていない [53]．アーキテクチャ設計よりも下流の開発工程の，ソフトウェア実装を考慮したコンポーネント技術として，THINK コンポーネントフレームワーク，REMORA，ARINC-653 Component Model，CAmkES，AUTOSAR，TECS がある．本研究における対象領域は，ソフトウェアの実装を考慮したコンポーネント技術であり，ここでは，これらの既存コンポーネント技術について詳しく述べる．

THINK コンポーネントフレームワーク [38] は，FRACTAL コンポーネントモデル [14] を，組込みシステム開発向けに実装したコンポーネント技術である．THINK は，アプリケーションソフトウェアだけでなく，リアルタイム OS である，μC/OS-II をコンポーネントベースで開発（コンポーネント化）しており，タスクなどのリアルタイム OS のリソースをコンポーネントとして扱うことができる．さらに，THINK では，コンポーネント化に伴う，実行時間やメモリ使用量の増加を抑えることができる．しかしながら，THINK では，メモリ保護を考慮したソフトウェアの開発をサポートしていない．

REMORA[54] は，ワイヤレスセンサネットワーク機器のような組込みシステム開発向けのコンポーネント技術である．REMORA では，リアルタイム OS の提供する機能を抽象化するためのコンポーネントを用いることで，ミドルウェアのように，リアルタイム OS に依存したソフトウェア部分をコンポーネント化することができる [55]．ミドルウェアをコンポーネント化することで，ターゲットアプリケーションの要求に応じたソフトウェアのカスタマイズが容易にできる．さらに，REMORA を用いてコンポーネント化したことによる，CPU 時間やメモリ使用量のオーバヘッドは小さい．しかしながら，REMORA では，メモリ保護を考慮したソフトウェアの開発をサポートしていない．

ARINC-653 Component Model（ACM）[21] は，CORBA コンポーネントモデルと，ARINC653 仕様 [1] のプラットフォームに基づいたコンポーネント技術である．ARINC653 は，メモリ空間と CPU 時間のパーティショニング機能を提供する，航空機分野におけるソフトウェアプラットフォーム仕様である．ACM では，CORBA コンポーネントモデルに基いて，ソフトウェアのコンポーネントベース開発が可能であり，個々のコンポーネントは，ARINC653 仕様プラットフォームのパーティションに，静的に配置される．そして，コンポーネント間の通信は，ARINC653 仕様プラットフォームの機能を用いて実現され，コンポーネント開発者からはこの通信処理は隠蔽される．しかしながら，ACM では時間保護機能に主眼が置かれており，メモリ保護を実現する方法の具体的な記述はなく，また，プラットフォームのコンポーネント化についても考慮されていない．さらに，評価実験においても，Linux を ARINC653 仕様の OS の代わりに用いて実施しており，また，メモリ消費量が評価されていないため，組込みシステムにおける評価が十分でない．

CAmkES[31] は，L4 マイクロカーネル [17] をベースのリアルタイム OS として用いる，メモリ保護を考慮した組込みシステム向けのコンポーネント技術である．CAmkES では，個々のコンポーネントが持つインタフェースに含まれる個々のメソッドに対してアクセス権を設定し，システム実行時における不正なコンポーネントアクセスを，L4 マイクロカーネルの機能により検出することが可能である．しかしながら，アクセス権の設定が細かくできる反面，そのために，コンポーネント間通信において，L4 マイクロカーネルの機能を常に利用しており，実行時間やメモリ使用量のオーバヘッドが大きい．さらに，CAmkES におけるコンポーネントは，すべて非特権モードで動作するため，特権モードで動作するミドルウェアやデバイスドライバのようなソフトウェア部分を開発する必要がある組込みシステムには不十分である．

AUTOSAR の定める仕様では，メモリ保護機能を持ったリアルタイム OS が提供されており，そのリアルタイム OS 上で動作するアプリケーションソフトウェアは，コンポーネント技術を用いて開発する．アプリケーションソフトウェアのコンポーネン

トは，OS アプリケーションと呼ばれる，AUTOSAR OS の提供するパーティションに配置される．OS アプリケーションには，信頼 OS アプリケーションと非信頼 OS アプリケーションの 2 種類が存在し，所属するコンポーネントはそれぞれ，特権モード，非特権モードで動作する．コンポーネント間の通信は，RTE（Run Time Environment）と呼ばれる層によって抽象化されており，個々のコンポーネントは RTE に対する API を用いて開発される．RTE の内部処理は，コンポーネントの設定情報から自動生成されるため，コンポーネント開発者が意識する必要はない．しかしながら，AUTOSAR OS 仕様では，メモリ保護のための定義が不十分であり，さらに，AUTOSAR 仕様に基づき，メモリ保護を考慮したコンポーネント技術について，その実装方法や評価が公表されている文献は発見できなかった．さらに，AUTOSAR 仕様では，コンポーネントベースで開発するソフトウェア部分は，RTE よりも上位のアプリケーション部分のみであり，RTE よりも下位の，リアルタイム OS やミドルウェア，デバイスドライバ（BSW：Basic SoftWare と呼ばれる）は，コンポーネントベースで開発しない．また，BSW の仕様は，AUTOSAR 仕様で，自動車システムに特化して定義されているため，自動車以外の組込みシステム開発への適用は困難である．

文献 [24] では，AUTOSAR の BSW を，COMPASS コンポーネント技術 [18] によりコンポーネント化している．この文献では，BSW に含まれる，FlexRay[40] のドライバ，および，通信ミドルウェアを，実行時間やメモリ使用量のオーバヘッドを抑えつつ，コンポーネント化している．さらに，アプリケーションの要求に応じて，不必要なコンポーネントを容易に取り外すことができ，無駄なメモリ使用量や実行時間を削減できることを示している．しかしながら，このコンポーネント技術では，メモリ保護を考慮したソフトウェア開発をサポートしていない．

TECS[65] は，組込みシステム開発向けのコンポーネント技術であり，コンポーネントをすべて静的に生成し，コンポーネント間の結合（関数呼出しによる通信処理）を最適化することで，コンポーネント化に伴う実行時間やメモリ使用量のオーバヘッドを抑えることが可能である [9]．さらに，TECS では，結合されたコンポーネント間

Table 2.1: 組込みシステム向けコンポーネント技術の比較

	THINK	REMORA	ACM	CAmkES	AUTOSAR	COMPASS	TECS
プラットフォーム開発	○	○	×	×	×	○	○
メモリ保護	×	×	○	○	○	×	×

に，共通のインタフェースを持つコンポーネントを容易に挿入する機能を持ち，この機能により，コンポーネント間の結合を抽象化することができる [8]．また，TECS では，リアルタイム OS のリソースや，TCP/IP プロトコルスタックのようなミドルウェアを，実行時間やメモリ使用量のオーバヘッドを抑えてコンポーネント化することができ，リアルタイム OS のリソースをコンポーネントとして扱ったり，ミドルウェアを容易にカスタマイズすることが可能である [10][68]．しかしながら，このコンポーネント技術では，メモリ保護を考慮したソフトウェア開発をサポートしていない．

これら既存のコンポーネント技術について，プラットフォームのコンポーネントベース開発を考慮しているかどうか，メモリ保護を考慮しているかどうかという観点でまとめた表を，表 2.1 に示す．

2.4 TECS

ここで，本研究において，メモリ保護を考慮したコンポーネント技術の基となる TECS のコンポーネントモデルについて述べる．

2.4.1 コンポーネントモデル

TECS では，インスタンス化されたコンポーネントをセルと呼ぶ[1]．セルは，自身の機能を提供するインタフェースである受け口，他のセルの機能を利用するためのインタフェースである呼び口，セルの付随情報（定数）を表す属性，セルの内部状態を

[1]オブジェクト指向の用語との対比では，コンポーネントがオブジェクトに，後述のセルタイプがクラスに，セルがインスタンスに対応する．

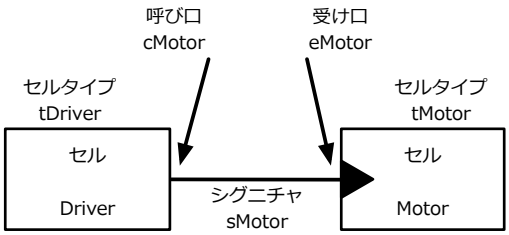

Figure 2.2: TECS コンポーネント図の例

表す変数で構成される．受け口は，セルの機能を提供する関数の集合であり，呼び口は，他のセルの機能を利用するための関数呼出しの集合である．1 つのセルは，複数の呼び口や受け口を持つことができる．また，セルの提供する機能は，C 言語によって実装される．

受け口と呼び口の型は，シグニチャと呼ばれる関数ヘッダの集合で定義される．セルの呼び口は，同一のシグニチャを持つ他のセルの受け口と結合できる．これにより，前者のセルから後者のセルの関数群を呼び出すことが可能になる．ここで，TECS におけるセル間の通信は，呼び口から受け口への関数呼出しによってのみ行われる．あるセルから，自身の呼び口に含まれる関数（呼び口関数）を呼び出すことで，その呼び口に結合された受け口に含まれる関数（受け口関数）が呼び出され，セル間の通信が行われる．図 2.2 は，セル Driver の呼び口 cMotor とセル Motor の受け口 eMotor を結合したことを，TECS コンポーネント図によって表している．

セルの型，すなわち，セルが持つ受け口，呼び口，属性，変数の組を，セルタイプと呼ぶ．1 つのセルタイプから，複数のセルをインスタンス化することができる．1 つのセルのみをインスタンス化できるセルタイプを，シングルトンと呼ぶ．

2.4.2 コンポーネント記述

コンポーネントの要素は，TECS の仕様で定義されたコンポーネント記述を用いて表される．コンポーネント記述は，シグニチャ記述，セルタイプ記述，組上げ記述

```
1: /*モータのシグニチャ*/
2: signature sMotor{
3:   void setPWM([in] int32_t pwm,
4:               [in] bool_t brakeMode);
5:   void setRevolution([in] int32_t rev);
6:   void getRevolution([out] int32_t *rev);
7: };
```

Figure 2.3: シグニチャ記述

に分類される．下記で図 2.2 を例として，それぞれのコンポーネント記述を説明する．

シグニチャ記述

シグニチャ記述は，セルのインタフェースを定義するために用いられる．signature キーワードに続けてシグニチャ名（通例，接頭辞 “s” をつける）を記述し，そのシグニチャが持つ関数ヘッダを中括弧内に列挙する．図 2.3 の例では，関数 setPWM と setRevolution と getRevolution を持つシグニチャ sMotor を定義している．

TECS では，インタフェースの定義を明確にするために，図 2.3 の 3 行目の pwm で見られるような，引数の指定子を用意している．指定子は引数が，in が入力，out が出力であることをそれぞれ示す．

セルタイプ記述

セルタイプ記述は，呼び口，受け口，属性，変数を用いてセルタイプを定義するのに用いられる．celltype キーワードに続けてセルタイプ名（通例，接頭辞 “t” をつける）を記述し，セルタイプが持つ要素を列挙する．呼び口は，call キーワード，シグニチャ名，呼び口名（通例，接頭辞 “c” をつける）の順に記述する．受け口は，entry キーワード，シグニチャ名，受け口名（通例，接頭辞 “e” をつける）の順に記述する．属性は attr，変数は var キーワードを用いてそれぞれ列挙する．属性，変数が存在しない場合は省略することができる．

図 2.4 の例では，シグニチャが sMotor の呼び口 cMotor を持つセルタイプ tDriver と，シグニチャが sMotor の受け口 eMotor，属性，変数を持つセルタイプ tMotor を定

```
 1: /*制御器の定義*/
 2: celltype tDriver{
 3:   /*呼び口の宣言*/
 4:   call sMotor cMotor;
 5:   var{ /*変数*/
 6:     int32_t speed = 50;
 7:     int32_t turn;
 8:   };
 9: };
10: /*モータの定義*/
11: celltype tMotor{
12:   /*受け口の宣言*/
13:   entry sMotor eMotor;
14:    /*省略：他の受け口，呼び口*/
15:   attr{ /*属性*/
16:     uint32_t portNumber;
17:   };
18:   var{ /*変数*/
19:     int32_t revolution;
20:     int32_t pwm;
21:     uint32_t preEdge;
22:   };
23: };
```

Figure 2.4: セルタイプ記述

義している．

また，特別なセルタイプ記述として，複合セルタイプや optional 指定子，呼び口配列がある．複合セルタイプは，複数のセルを組合わせた，一つの新しいセルタイプを定義するものである．optional 指定子は，呼び口を未結合のままにしてもよいことを指定するための指定子である．呼び口配列は，配列化された呼び口であり，配列添数により呼び先のコンポーネントを切替えることができる．また，呼び口配列の宣言時に要素数を省略することもでき，その場合には，セルの生成時に配列の要素数を決定する．これらの記述例は，3.5.3 節で示す．

組上げ記述

組上げ記述は，セルタイプをインスタンス化してセルを生成し，セル同士の結合関

```
 1: /*モータのインスタンス化*/
 2: cell tMotor Motor{
 3:   /*属性の設定*/
 4:   portNumber = C_EXP("NXT_PORT_B");
 5: };
 6: /*制御器のインスタンス化*/
 7: cell tDriver Driver{
 8:   /*呼び口と受け口の結合*/
 9:   cMotor = Motor.eMotor;
10: };
```

Figure 2.5: 組上げ記述

係を定義してソフトウェア全体を構築するために用いられる．cell キーワードに続けてセルタイプ名，セル名を記述し，中括弧内には，自身の呼び口と他のセルの受け口との結合を列挙する．結合は，呼び口名，=，結合先の受け口の順に記述する．なお，呼び口のないセルには結合の記述は必要ない．図 2.5 の例では，セルタイプ tMotor のセル Motor とセルタイプ tDriver のセル Driver が，それぞれインスタンス化されている．またセル Driver の呼び口 cMotor は，セル Motor の受け口 eMotor と結合している．

2.4.3 TECS ジェネレータプラグイン

TECS ジェネレータは，シグニチャ記述，セルタイプ記述，組上げ記述のコンポーネント記述を入力とし，セルの生成とセル間の結合部分，および，セルの提供する機能（セルタイプの受け口関数）のテンプレートとなる C 言語のソースコードを生成する，Ruby 言語によって実装されたツールである．

TECS ジェネレータには，数種類のプラグインを適用することができ，使用するターゲットに依存した独自の機能を Ruby コードで実装し，TECS ジェネレータの機能を拡張することができる．TECS ジェネレータに適用できるプラグインの種類に，through プラグインや，celltype プラグインがある．through プラグインは，組上げ記述におけるセル間の結合に対して適用することができ，そのセル間に定型的なセルを

```
 1: cell tCallee Callee{
 2: };
 3: cell tCaller Caller{
 4:   /*through プラグインの指定*/
 5:   [through(TracePlugin,"")]
 6:   cCall = Callee.eEntry;
 7: }
 8: /*celltype プラグインの指定*/
 9: [/*省略：他の指定子*/,
     generate(HRP2TaskPlugin, "")]
10: celltype tTask{
11:   /*省略：呼び口や受け口などの定義*/
12: }
```

Figure 2.6: プラグインの指定

挿入するために用いられる．一方，celltype プラグインは，セルタイプ記述におけるセルタイプの定義に対して適用することができ，そのセルタイプからセルを生成した場合に，外部ファイルに情報を出力するために用いられる．

through プラグインと，celltype プラグインを使用するためのコンポーネント記述例を，図 2.6 に示す．5-6 行目は，TracePlugin という through プラグインを，セル Caller の呼び口 cCall とセル Callee の受け口 eEntry との結合部分に対して適用することを指定している．この記述により，これらの呼び口と受け口の間に，同じシグニチャの受け口と呼び口を持つ別のセル（例えば，セルの呼出しをするときにトレースログを出力する機能を持ったセル）を挿入することができる．9-12 行目は，HRP2TaskPugin という celltype プラグインを，tTask というセルタイプに対して適用することを指定している．この記述により，tTask からセルを生成した場合に，TECS ジェネレータにおいて，HRP2TaskPlugin に定義された，独自の出力（例えば，リアルタイム OS にタスクの生成情報を渡すための API を出力するなど）をさせることができる．

2.4.4 カーネルオブジェクトのコンポーネント化

TECS において，タスクやセマフォなどのカーネルオブジェクトをコンポーネントとして扱い，ソフトウェアを開発する手法が提案されている [10]．カーネルオブジェクトをコンポーネントとして扱うことにより，ソフトウェアの構造が分かりやすくなるだけでなく，タスクやセマフォの構成情報を記述するファイルを TECS ジェネレータによって生成することができる．カーネルオブジェクトを TECS 仕様に基づいてコンポーネント化したリアルタイム OS として，ASP カーネルがある．

2.4.5 TECS における課題

2.3 節で述べたとおり，TECS では，ソフトウェアのコンポーネント化に伴う実行時間やメモリ使用量のオーバヘッドが小さく，デバイスドライバのようなソフトウェアの細部までコンポーネント化しても，システム全体のオーバヘッドを小さく抑えられると考えられる．また，デバイスドライバなどを含むソフトウェアプラットフォームをコンポーネント化することによって，システムのハードウェア構成や要求される機能に応じて，容易にソフトウェアプラットフォームの構成を変更できる．しかしながら，TECS によってソフトウェアプラットフォームをコンポーネント化した事例はなく，実行時間やメモリ使用量のオーバヘッド，および，ソフトウェアプラットフォームの変更容易性は評価されていない．

また，TECS は，メモリ保護のようなパーティショニングが要求されるソフトウェア開発に対応しておらず，コンポーネントをパーティションに分けて配置するためのコンポーネント記述も提供していない．

CHAPTER 3

ソフトウェアプラットフォームのコンポーネントベース開発

3.1 概要

本章では，LEGO 社の Mindstorms NXT を対象とし，組込みシステム向けのソフトウェアプラットフォームを，コンポーネント技術 TECS により開発した事例について述べる．本事例により，ソフトウェアプラットフォームをコンポーネントベース開発することの有効性，および，TECS をソフトウェアプラットフォーム開発に適用できることを示す．

LEGO 社の Mindstorms は，モータやセンサ，LEGO ブロックといった部品を組み合わせてロボットを作成することができ，さらに，ユーザの開発したソフトウェアによって，そのロボットを制御することができる．Mindstorms は二種類のモデルがあり，Mindstorms NXT（以下，NXT）は，第二世代のモデルである．Mindstorms は，組込みソフトウェア開発の教材として広く使われている [66][75][29].

NXT では，多数のデバイスが用意されているため，使用するデバイスの組み合わせは多様である．一方，NXT に搭載されているメモリは容量が小さいため，すべて

のデバイスの組み合わせに対応できるよう，すべてのデバイスドライバを含めてしまうと，ユーザが開発できるアプリケーションの規模が小さくなってしまう．これまでに開発されたソフトウェアプラットフォームでは，個々のデバイスドライバの取り外しができず，実際には使用しないデバイスドライバもソフトウェアに含まれてしまう．そのため，TECS を用いて開発したソフトウェアプラットフォームにより，ソフトウェアに含めるデバイスドライバを容易に選択できること，および，無駄な実行時間やメモリ使用量を削減できることを，既存のソフトウェアプラットフォームとの比較評価によって示すことができる NXT を開発対象とすることが適切であると考えた．

また，NXT を用いたロボットコンテストである ET ロボコン [22] では，モデルを用いた組込みソフトウェアの分析や設計に関する教育を目的としており，参加者にはモデルを用いたソフトウェア開発を行うことが求められる．さらに，ET ロボコンでは，倒立二輪ライントレースロボットの制御ソフトウェアを開発することが求められる．この制御ソフトウェアでは，倒立二輪制御と，ライントレース制御というリアルタイム性が求められる処理を，並行に行う必要がある．組込みソフトウェア開発においては，このような制御ソフトウェアの実現を容易とするために，リアルタイム OS を用いることが一般的である．したがって，モデルを用いたソフトウェア開発との親和性があり，かつリアルタイム OS の機能を用いることが可能なソフトウェアプラットフォーム開発においても TECS が有用であることを示す必要があると考える．

ソフトウェアプラットフォーム開発における TECS の有用性を評価するために，NXT 用ソフトウェアプラットフォーム（以下，ATON：ASP+TECS On NXT）を TECS の仕様に基いて開発した．TECS により開発したソフトウェアは，リアルタイム OS の機能を使用可能で，実装時にはリアルタイム OS 上で実行される．ATON には，すべてのデバイスドライバが TECS の仕様に基づいたコンポーネントとして含まれているため，必要なデバイスドライバコンポーネントの選択をコンポーネントレベルで行うことが可能である．ATON では，TECS を用いることで，ソフトウェアの構造をコンポーネントレベルでモデル化でき，ソフトウェアの開発効率を向上させることがで

きる．

本章の研究内容による貢献は，以下のとおりである．

(1-1) TECS によりソフトウェアプラットフォームを開発する事例を示す．

(1-2) ソフトウェアプラットフォームをコンポーネント技術を用いて開発することの利点を明らかにする．

(1-2) については，ソフトウェアプラットフォームのカスタマイズが容易にでき，さらに，そのカスタマイズにより，メモリ使用量や実行時間を削減できることを評価して示す．

本章の構成は次のとおりである．まず，NXT について説明したあと，NXT 用のソフトウェアプラットフォームの要件，および，既存のソフトウェアプラットフォームについて述べ，要件を満たすソフトウェアプラットフォームの設計方針，および，ベースとなる技術について述べる．次に，我々が開発したソフトウェアプラットフォームについて述べ，事例により，ATON の有効性について述べる．

3.2 MINDSTORMS NXT

3.2.1 ハードウェア構成

NXT は，プロセッサを内蔵した NXT 本体（図 3.1）に，モータや各種センサ，および，LEGO ブロックを組み合わせて接続することによってロボットを自作できるキットであり，教育目的などで使用されている．

NXT 本体には，LCD ディスプレイ，4 つのボタン（ENTER，EXIT，RUN，STOP），サウンド出力がユーザインタフェースとして取り付けられている．そして，モータを取り付けるためのポートが 3 つ，センサを取り付けるためのポートが 4 つ存在する．また，ホスト PC から NXT 本体に内蔵されている FLASH メモリへロードモジュールをダウンロードするための USB ポートが存在する．

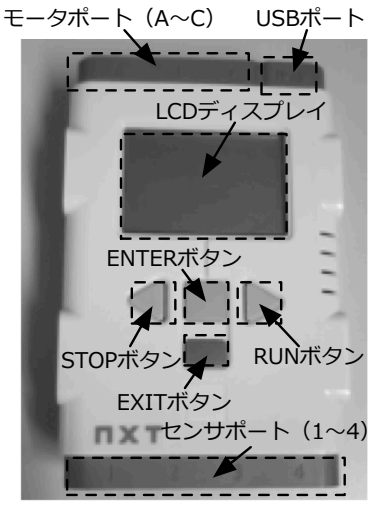

Figure 3.1: NXT 本体

NXT 本体に内蔵されているプロセッサは，メインプロセッサとコプロセッサの 2 種類がある．メインプロセッサは，ユーザのプログラムを実行し，コプロセッサはメインプロセッサからの指示により，モータの PWM 制御等を行い，ユーザによるプログラミングは不可能である．メインプロセッサは，周波数 48MHz の Atmel 社製 32 ビット ARM プロセッサ AT91SAM7S256[5] であり，搭載しているメモリは，256KB の FLASH ROM，64KB の RAM である．コプロセッサは，周波数 8MHz の Atmel 社製 8 ビット AVR プロセッサ ATmega48 である．

NXT 本体には，Bluetooth デバイスが搭載されており，NXT 本体とホスト PC との間で Bluetooth 通信（仮想シリアル通信）を行うことが可能である．

NXT 本体内部における，プロセッサと各種デバイスとの接続関係を図 3.2 に示す．LCD ディスプレイ，サウンド，Bluetooth は，メインプロセッサの SPI（Serial Peripheral Interface），SSC（Synchronous Serial Controller），USART1 にそれぞれ接続されている．NXT 本体の 4 つのボタン，モータポート，センサポートは，コプロセッサと接続されており，コプロセッサは，メインプロセッサの TWI（Two-Wire Interface）に接続されている．メインプロセッサからボタン押下状態やセンサ値の取得，モータの回転速度の設定をするために，メインプロセッサとコプロセッサとの間で通信を行う．

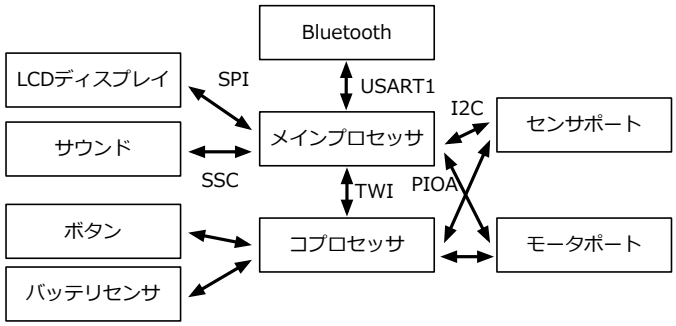

Figure 3.2: NXT の内部構成

モータポートとセンサポートは，メインプロセッサの PIOA（Parallel I/O controller A）にも接続されており，PIOA を介した通信では，モータの回転数取得やセンサの電源投入，I2C 通信によるセンサ値の取得を行う．

3.2.2 既存のソフトウェア開発

NXT 用のソフトウェア開発環境として，ROBOLAB や教育用 NXT ソフトウェアが LEGO 社から提供されている [32]．これらの環境は，GUI ベースの言語を採用することにより，プログラミングを簡単化している．しかしながら，プログラミングが簡単である反面，ユーザ定義の関数や変数を使用することができず，C 言語や Java などのプログラミング言語のように細かい制御を実現することはできない．

そのため，C 言語や Java を用いたプログラミングが可能なオープンソースの NXT 用のソフトウェアプラットフォームが開発されている．既存の NXT 用ソフトウェアプラットフォームとしては，LeJOS NXJ[36]，nxtOSEK[44]，TOPPERS/JSP プラットフォーム [59] が挙げられる．

また，第一世代の Mindstorms である RCX 用のソフトウェア開発環境として，Lisp 処理系の XS[73] がある．

3.3 NXT用ソフトウェアプラットフォーム

NXT用ソフトウェアプラットフォームを開発するにあたって，定めた要件について述べ，既存のソフトウェアプラットフォームの問題について説明する．

3.3.1 プラットフォームの要件

NXT用ソフトウェアプラットフォームに求められる要件を，以下のように定義した．

要件1. C言語ベースでプログラムできること

要件2. オブジェクトサイズが小さいこと

要件3. デバイスドライバが取り外しできること

要件4. デバイスドライバを取り外すことによって，他のデバイスドライバやミドルウェアのソースコードに影響がでないこと

要件5. デバッグのために任意のログを出力できること

要件6. モデルを用いたソフトウェア開発との親和性があること

要件7. リアルタイムOSの機能を用いることが可能であること

要件1に関しては，現在，多くの組込みソフトウェアがC言語ベースで開発されている[67]．組込みソフトウェア開発の教材に用いるためには，C言語ベースの開発環境を提供する必要があると考え定めた．

要件2に関しては，NXT上でソフトウェアを動作させる際には，そのロードモジュール全体をNXTに搭載されているメモリにロードしなければならない．NXTに搭載されているメモリは，FLASH ROMが256KB，RAMが64KBと小さいため，ロー

ドモジュールのオブジェクトサイズが大きくなってしまうと，NXT に搭載されているメモリにロードできなくなってしまうために定めた．

要件 3 に関して，NXT には複数のデバイスが搭載されており，作成するアプリケーションによっては，使用しないデバイスが存在する．使用しないデバイスのデバイスドライバをロードモジュールに含めてしまうと，オブジェクトサイズや実行時間に無駄が生じてしまう可能性がある．要件 2 を満たすためにも，使用するデバイスドライバのみをロードモジュールに含め，使用しないデバイスドライバは含めないという設定ができることが求められる．

要件 4 に関しては，要件 3 で定めたデバイスドライバの取り外しによって，プラットフォームの提供する他のデバイスドライバやミドルウェアを変更する必要が生じてしまうと，デバイスドライバやミドルウェアの再利用性が低下してしまう．よって，デバイスドライバの有無が，他のデバイスドライバやミドルウェアに影響しないことが必要である．

要件 5 に関しては，現状では，NXT 上で動作するソフトウェアのデバッグを支援するツールは存在しない．そのため，NXT 上で動作させるソフトウェアのデバッグ手法としては，ソフトウェアの実行中に，デバッグのためのログを出力する方法が一般的である．よって，プラットフォームでデバッグのために任意のログを出力できるような機構を提供する必要がある．ログを出力できるデバイスとして，LCD ディスプレイと Bluetooth が考えられるが，LCD ディスプレイは表示できる領域が小さいことから，Bluetooth 通信を用いて，NXT からホスト PC にログを出力する方法が適切である．

要件 6 に関して，ET ロボコンでは，モデルを用いたソフトウェア開発を主目的としている．C 言語のみを用いたソフトウェア開発では，モデルと C 言語の間に乖離があるため，モデルから C 言語への変換が困難である．よって，ET ロボコンでの開発を支援するためには，モデルを用いたソフトウェア開発と親和性のあるプラットフォームが必要である．

要件 7 に関して，ET ロボコンでは，リアルタイム性の求められる制御ソフトウェアを開発する．そのためには，マルチタスクや周期ハンドラ，プリエンプティブな固定優先度ベーススケジューリングといったリアルタイム OS の機能が必要である．

なお，要件 6，7 は，ET ロボコンでの使用を考慮して定義した要件であるが，ET ロボコンにおいて必須の要件ではなく，ソフトウェア開発を容易化するために定義した要件である．さらに，要件 6，7 は，ET ロボコン以外であっても，モデルやリアルタイム OS を用いた設計やその教育において有用であると考えられる．

3.3.2 既存のプラットフォーム

ET ロボコンで用いられている既存の NXT 用ソフトウェアプラットフォームとして，leJOS，nxtOSEK，TOPPERS/JSP プラットフォームがある．

leJOS NXJ（以下，leJOS）[36] は，Java ベースでプログラミングを行うためのソフトウェアプラットフォームであり，Java 仮想マシン，NXT のデバイスドライバ群から構成される．

nxtOSEK[44] と TOPPERS/JSP プラットフォーム（以下，JSP プラットフォーム）[59] は，C 言語ベースでプログラミングを行うためのソフトウェアプラットフォームであり，リアルタイム OS，NXT のデバイスドライバ群，ミドルウェアから構成される．nxtOSEK では OSEK/VDX 仕様 [48] の TOPPERS/ATK1（旧：TOPPERS/OSEK カーネル）を，JSP プラットフォームでは μ ITRON4.0 仕様 [64] の TOPPERS/JSP カーネル（以下，JSP カーネル）を，それぞれリアルタイム OS として用いている．

要件 1 に関して，leJOS は Java ベースのプラットフォームであるが，nxtOSEK と JSP プラットフォームは C 言語ベースのプラットフォームである．

要件 2 に関して，上記すべてのプラットフォームでは，ARM プロセッサの Thumb [1]を用いることによって，オブジェクトサイズを小さくしている．

要件 3，要件 4 に関して，上記すべてのプラットフォームでは，個々のデバイスド

[1]命令長が 16 ビットのモード．命令長が 32 ビットのモードは ARM モードである．

ライバが取り外しできるような構造になっていない．個々のデバイスドライバを取り外すように修正を加えることも可能だが，変更は容易ではない．

要件 5 に関して，leJOS と nxtOSEK において，ログを出力するためには，専用のツール（NXTGamePad）が必要であり，さらに，ログの形式も指定されている．JSP プラットフォームでは，JSP カーネルのシステムログ機構を拡張することによって，任意のログを出力できる．

要件 6 に関して，leJOS は，Java ベースのプラットフォームであり，オブジェクト指向モデルを用いたソフトウェア開発との親和性があると考えられる．nxtOSEK と JSP プラットフォームは，リアルタイム OS とデバイスドライバのみを提供しており，モデルから C 言語の変換をサポートするような機構は提供していない．

要件 7 に関しては，nxtOSEK と JSP プラットフォームは，それぞれ OSEK 仕様と ITRON 仕様の OS の上でソフトウェアを実行するため，リアルタイム OS の機能を使用することが可能である．一方，leJOS は，Java ベースのプラットフォームであり，リアルタイム OS の機能を使用することは不可能である．

以上のように，既存のソフトウェアプラットホームですべての要件を満たすものは存在しない．

3.4 プラットフォームの設計方針

3.4.1 基本方針

TECS により実現するソフトウェアプラットフォームによって，前章で挙げた要件を満たすための方針について述べる．

TECS は，既存の多くのコンポーネントシステムと異なり，C 言語でのソフトウェア開発が可能であり，要件 1 を満たす．また，TECS で開発したコードはリアルタイム OS 上で実行されるため，要件 7 を満たす．

TECS では，静的なコンポーネントモデルを採用しており，インスタンス化や結合にかかる実行時オーバヘッドをなくすことができる．さらに，コンポーネントの結合の最適化を行うことで RAM の使用量や実行時オーバヘッド削減できる [65]．これにより要件 2 を満たせる．

コンポーネント化に伴うオーバヘッドが小さいことにより，粒度の小さなソフトウェアモジュールや資源まで，コンポーネントとして扱うことができる．コンポーネントの粒度が小さいことは，システムの細部にわたる最適化がコンポーネントのレベルで行えることを意味し，システムの目的に合わせて細部まで最適化する必要がある組込みシステムには重要な利点となる．これにより要件 3 を満たせる．また，要件 4 の実現を容易化できる．

TECS では，コンポーネントの定義や生成，結合を表すための独自のコンポーネント記述言語が規定されている．このコンポーネント記述言語により記述したファイルを，TECS ジェネレータと呼ばれるツールに入力することによって，コンポーネントの生成と結合部分，および，コンポーネントの提供する機能のテンプレートとなる C 言語のソースコードを生成する．

3.4.2 ASP+TECS による要件の実現方針

TECS と ASP カーネル，および，カーネルオブジェクトのコンポーネントを提供するプラットフォーム（以降，ASP+TECS）を用いることによって，要件を実現する．

要件 1 に関して，TECS は C 言語ベース開発を目的としたコンポーネントシステムであるため，C 言語でのアプリケーション開発が可能である．

要件 2 に関して，TECS では，コンポーネント間の結合に関して，TECS ジェネレータが最適化を行うため，コンポーネント化に伴うオブジェクトサイズや実行時間のオーバヘッドが小さい [9][65].

また，既存の ARM プロセッサ対応 ASP カーネルでは，Thumb 対応がなされておらず，Thumb モードでプログラムをコンパイルできないため，ソフトウェアのオブ

ジェクトサイズが大きくなってしまう．そこで，ASP カーネルを Thumb 対応することにより，この問題を解決する．

要件 3 に関して，デバイス単位でデバイスドライバを TECS の仕様に基づいてコンポーネント化し，コンポーネントレベルでのデバイスドライバの取り外しを可能にする．

また，デバイスドライバは，デバイスドライバ本体や初期化ルーチン，割込みサービスルーチンなど，そのデバイスに関するすべてのコンポーネントから構成される一つのコンポーネント（複合コンポーネント）とする．

要件 4 に関して，TECS では，TECS ジェネレータによってコンポーネント間を結合するためのインタフェースコードを生成すること，および，コンポーネントの未結合を許容するための optional 指定子や，任意の数の結合を許容するための呼び口配列 [43] を用いることによって，デバイスドライバの取り外しが他のデバイスドライバやミドルウェアに影響しないようにすることができる．

要件 5 に関して，ASP カーネルのシステムログ機構を用いることで，任意のログを出力することができる．既存の ASP カーネルにおけるシステムログ機構では，ネットワークを介したログの出力に対応しておらず，ネットワーク通信の同期を待つことができないため，ASP カーネルのシステムログ機構を拡張し，Bluetooth 通信によるログ出力を可能とする．

ASP カーネルはログを可視化するためのツールである TLV（Trace Log Visualizer）[69] に対応したログを出力できるようになっている．そのため，ASP カーネルのシステムログ機構を用いることにより，NXT 上のプログラムが出力したログを可視化し，デバッグを効率化することが可能である．

さらに，TECS の through プラグインを用いることで，結合されたセルとセルの間に，それらのセル間のインタフェースに対応したログを出力するセルを挿入することができ，各コンポーネントのインタフェースに対応したログを出力することができる．例えば，呼び出されたセル名，受け口名，関数名をログとして出力することができる

Table 3.1: NXT 用ソフトウェアプラットフォームの比較

ソフトウェアプラットフォーム	要件 1	要件 2	要件 3	要件 4	要件 5	要件 6	要件 7
leJOS	×	○	×	×	△	○	×
nxtOSEK	○	○	×	×	△	×	○
JSP プラットフォーム	○	○	×	×	○	×	○
ASP+TECS プラットフォーム	○	○	○	○	○	○	○

[10]．ここで，through プラグインを用いて挿入したセルに対応するソースコードと，コンポーネント記述は，TECS ジェネレータによって，自動生成することができる．

要件 6 に関して，TECS では，ソフトウェアの構造モデルを記述するための，コンポーネント図やコンポーネント記述が規定されているため，モデルを用いたソフトウェア開発との親和性があるといえる．ソフトウェアの分析と設計を行ったモデルを，TECS 仕様で規定されたコンポーネント図やコンポーネント記述で記述する必要はあるが，この記述の変換は，モデル同士での変換であるため，モデルから C 言語に直接変換するよりも容易である．例えば，オブジェクトモデルのクラスを，TECS コンポーネントモデルのセルタイプに一対一で対応させることにより，機械的な変換が可能である [72]．このコンポーネント記述から，TECS ジェネレータによって，コンポーネントの生成と結合部分，および，コンポーネントの提供する機能のテンプレートとなる C 言語のソースコードを生成できる．

ASP+TECS を用いて開発したソフトウェアプラットフォームと既存のソフトウェアプラットフォームとを比較した結果を表 3.1 に示す．

3.5 ATON

3.5.1 ATON の概要

ATON は，ASP+TECS をベースとした，NXT 用のソフトウェアプラットフォームであり，リアルタイム OS である ASP カーネルと，TECS 仕様に基いて開発された，

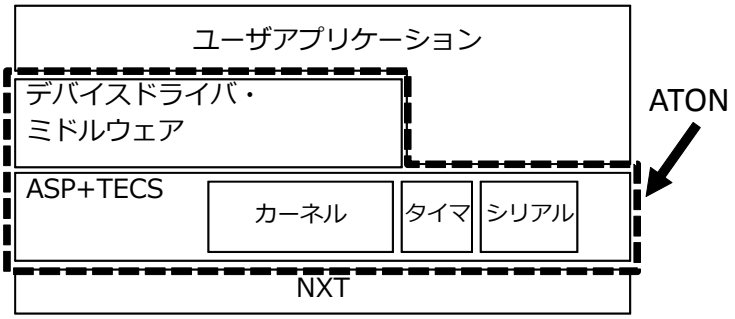

Figure 3.3: ATON の構成

ASP カーネルのカーネルオブジェクト，NXT のデバイスドライバ，および，ミドルウェアのコンポーネント群から構成されている（図 3.3）．

ASP カーネルは，システム時刻管理機能を実現するために，タイマドライバを，そして，システムログ機能を実現するために，シリアルドライバをそれぞれ内包している．また，ASP カーネルのカーネルオブジェクトは，TECS の仕様に基づいてコンポーネント化されている [10].

NXT のデバイスドライバ群，および，ミドルウェアは，TECS の仕様に基づいてコンポーネント化した．また，デバイスドライバについては，要件 3 を満たすように，デバイス単位でコンポーネントを作成しており，デバイス単位での取り外しや取り付けが可能である．

次に，ATON を用いたアプリケーションの開発の流れを図 3.4 に示す．まず，アプリケーション独自のインタフェースとコンポーネントの定義を，シグニチャ記述とセルタイプ記述により行う（図 3.4 の手順 1）．そして，アプリケーション独自のコンポーネントと ATON で提供するコンポーネントを組み合わせることによるソフトウェアの構築を，組上げ記述により行う（図 3.4 の手順 2）．これらのコンポーネント記述を TECS ジェネレータに入力することで，コンポーネントの生成と結合部分（インタフェースコード）とコンポーネントのソースコードのテンプレートが生成される．ここで生成されるテンプレートを編集することによって，アプリケーションのソース

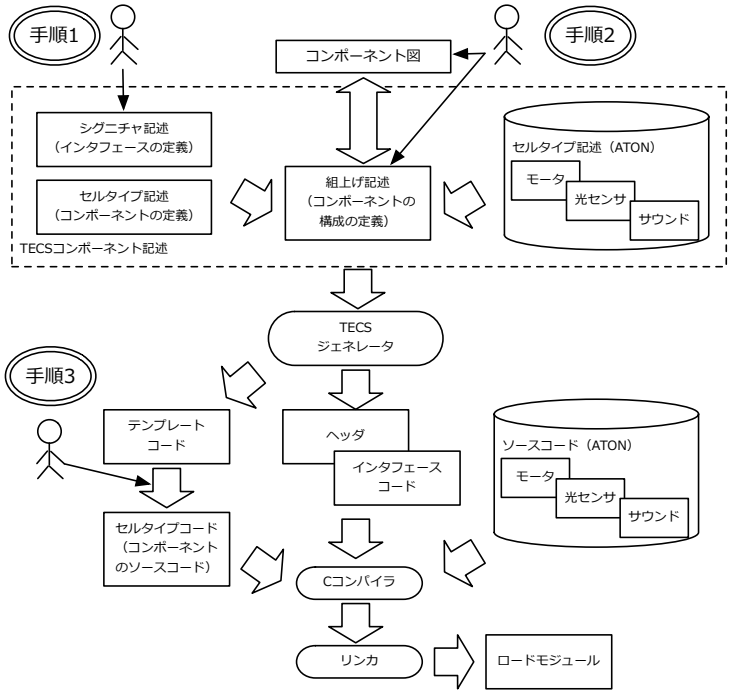

Figure 3.4: ATON を用いた開発の流れ

コードを作成する（図 3.4 の手順 3）．最後に，アプリケーションのソースコードとインタフェースコード，ATON の提供する ASP カーネルやデバイスドライバ，ミドルウェアのソースコードをコンパイル，リンクすることで，NXT 上で実行可能なロードモジュールを生成する．

3.5.2 ASP カーネルのポーティング

ASP カーネルを NXT 上で動作させるために，ASP カーネルのポーティング，および，拡張を行った．

ASP カーネルの NXT 上へのポーティングでは，NXT に依存する部分の実装を行った．具体的には，タイマドライバ，シリアルドライバ，スタートアップルーチン，割込みハンドラの出入り口を実装した．

さらに，要件 5 を満たすために，ASP カーネルのシステムログ機構を拡張し，ま

た，要件 2 を満たすために，ASP カーネルを ARM の Thumb に対応させた．

3.5.2.1 ASP カーネルのシステムログ機構の拡張

NXT では，メインプロセッサの USART1（以下，シリアルポート）と Bluetooth デバイスが接続されており，Bluetooth 通信を用いてログ出力を行う．そこでまず，シリアルポートのデバイスドライバ（ターゲット依存部）と，Bluetooth のデバイスドライバを開発した．

また，ログを受信するホスト PC と，ログを出力する NXT との間で Bluetooth 通信が確立するまでは，Bluetooth 通信を開始する目的でシリアルポートを使用するため，ログ出力を目的としたシリアルポートの利用ができない．そのため，Bluetooth 通信が確立しない内に，シリアルポートの利用開始処理を始めてはならない．

Bluetooth 通信が確立するまで，シリアルポートの利用開始処理を始めないようにするため，シリアルポートと Bluetooth との間で，セマフォによる同期をとることとした．セマフォの初期値を 0 とし，Bluetooth の通信が確立したときに，セマフォを返却するサービスコールを発行する．そして，シリアルポートの利用を開始する前に，セマフォを獲得するサービスコールを発行することによって，Bluetooth 通信が確立するまでシリアルポートの利用開始処理を待ち状態にし，Bluetooth の通信が確立した段階で，シリアルポートの利用開始処理を進めることができる．

しかし，既存の ASP カーネルのシステムログ機構では，ネットワークを介してログを出力することは考慮されておらず，シリアルポートの利用開始処理において，ターゲット依存部の処理に移行する前に，CPU ロック状態[2]に移行してしまい，ターゲット依存部で Bluetooth 通信が確立するまでシリアルポートの利用開始処理を待ち状態とすることができない．

そこで，Bluetooth の通信確立待ちを実現するための方法として，Bluetooth 通信が確立するまで待つための API を用意する方法と，シリアルポートの利用開始を行うた

[2]すべてのカーネル管理の割込みが禁止され，ディスパッチが保留される状態．

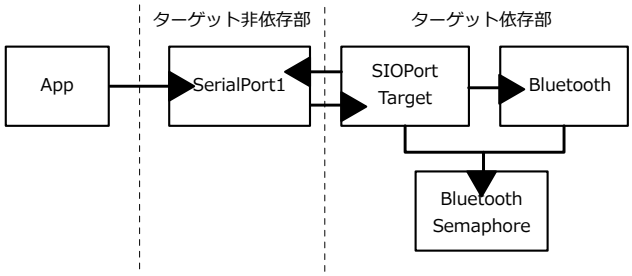

Figure 3.5: シリアルポートのコンポーネント図

めの API を拡張し，Bluetooth 通信が確立するまで待つことができるようにする方法の 2 つを検討した．前者の方法では，ASP カーネルに修正を加える必要はないが，アプリケーション側で，シリアルポートの利用開始を行う API を呼ぶ前に，Bluetooth の通信確立待ちを行う API を呼ばなければならない．一方，後者の方法では，アプリケーション側では，シリアルポートの利用開始を行う API を呼ぶだけでよいが，ASP カーネルに修正を加える必要がある．両者を検討した結果，前者の方法では，既存のアプリケーションを修正する必要が生じ，アプリケーションの移植性が低下してしまうことが考えられるため，後者の方法をとることとした．

Bluetooth 通信が確立するまで待つことができるように拡張したログ出力機構のコンポーネント図を図 3.5 に，シリアルポートの利用開始を行う API の流れを図 3.6 にそれぞれ示す． 図 3.5，図 3.6 において，App がシリアルポートの利用開始を行うアプリケーション，SerialPort1 がシリアルポートのターゲット非依存部，SIOPortTarget がシリアルポートのターゲット依存部を表す．既存の ASP カーネルでは，ターゲット依存部の処理（open）を呼び出す前に，CPU ロック状態に移行するため，ターゲット依存部において，Bluetooth 通信確立のための待ち状態となることができない．そこで，ターゲット非依存部において，CPU ロック状態に移行する前に，ターゲット依存部の処理（waitReady）を呼び出し，Bluetooth 通信が確立するまで待つようにした．

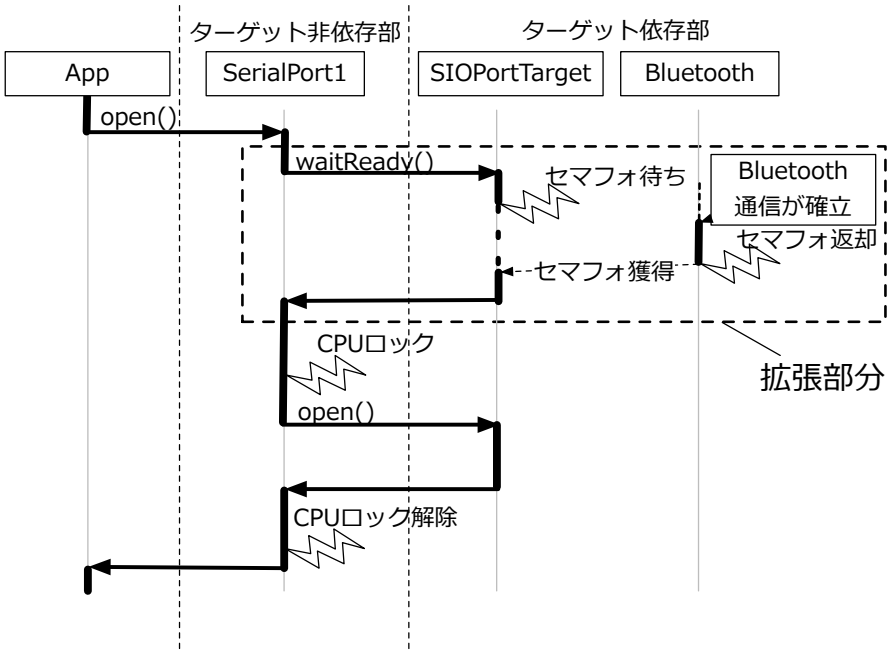

Figure 3.6: シリアルポート利用開始の流れ

3.5.2.2　ASP カーネルの THUMB 対応

NXT 上で動作させるソフトウェアを Thumb モードで実行できるように，ASP カーネルのコードを ARM の Thumb モードに対応させた．まず，Thumb モードで実行できない命令（mrs や msr），および，irq ハンドラや例外ハンドラを実行するコードは，ARM モードで実行するように記述した．そして，ARM モードで実行するコードから Thumb モードで実行するコードへジャンプする際には，モードを切り替えるために，bx 命令を使用するようにした（図 3.7）．なお，Thumb モードで実行するコードから ARM モードで実行するコードへジャンプする際は，コンパイラとリンカによって挿入されるグルーコードによってモードの切り替えが行われる．

3.5.3　デバイスドライバコンポーネントの設計

ここでは，デバイスドライバの設計方針について述べる．対象とするデバイスは，コプロセッサ，ボタン，バッテリセンサ，LCD ディスプレイ，サウンド，モータ，各

```
1:      .text
2:      .align 2
3:      .global current_sr
4: current_sr:
5:      mrs   r0, cpsr
6:      bx    lr
```

Figure 3.7: ARM モードのコード

種センサ（光センサやジャイロセンサなど）である．

3.5.3.1 コンポーネントの構成

デバイスドライバには，デバイスを操作するためのインタフェース（デバイスドライバ本体）だけではなく，初期化ルーチンや割込みサービスルーチンなど[3]が含まれる．そのため，デバイスドライバを取り外す際には，そのデバイスに対応する，初期化ルーチンや割込みサービスルーチンも取り外す必要がある．

デバイスドライバの取り外しを容易とするために，デバイスドライバコンポーネントは，そのデバイスに関連するすべてのコンポーネントから構成される複合コンポーネントとした．これは，デバイスドライバコンポーネントのセルタイプを，TECS 仕様における複合セルタイプ [43] とすることによって実現した．

デバイスドライバコンポーネントの定義の例として，サウンドのデバイスドライバコンポーネント（セルタイプ名：tSound）の定義を示す．tSound のコンポーネント記述を図 3.8 に，tSound のセル（Sound）のコンポーネント図を図 3.9 にそれぞれ示す．

なお，サウンドなどのデバイスドライバについては，高々1 つしかインスタンス化しないため，そのセルタイプをシングルトンとして定義した（図 3.8 の 2 行目[4]）．モータやセンサなどのデバイスドライバについては，同じセルタイプから複数のセルをインスタンス化できるが，同じセルタイプからインスタンス化されたセルの間では，

[3] ASP カーネルのカーネルオブジェクト．初期化処理や割込み処理を呼び出す主体である．
[4] active は，受け口関数が呼び出されない場合であっても，セルが動作することを表す指定子である．

```
 1: /*複合セルタイプの定義（シングルトン）
      ：サウンド*/
 2: [active, singleton]
 3: composite tSound{
 4:   entry sSound eSound; /*サウンド出力API*/
 5:   /*デバイスドライバ本体*/
 6:   cell tSoundBody SoundBody{
 7:   };
 8:   /*初期化ルーチン*/
 9:   cell tInitializeRoutine
              InitializeSound{
10:     cInitializeRoutine =
11:       SoundBody.eInitialize;
12:    };
13:    /*割込みサービスルーチン*/
14:    cell tISRWithConfigInterrupt ISRSound{
15:      ciBody = SoundBody.eiBody;
16:      interruptNumber =
17:        C_EXP("INTNO_SSC_PID");
18:      interruptAttribute =
19:        C_EXP("(TA_ENAINT|INTATR_SOUND)");
20:      interruptPriority =
21:        C_EXP("INTPRI_SOUND");
22:    };
23:    /*受け口のエクスポート*/
24:    eSound => SoundBody.eSound;
25:  };
```

Figure 3.8: サウンドドライバのコンポーネント記述

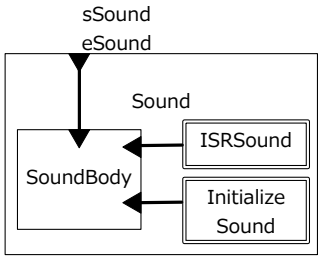

Figure 3.9: サウンドドライバのコンポーネント図

コードを共有する．そのため，複数のセルを生成したとしても，それらのコードがバイナリに重複して含まれることはない．

3.5.3.2 デバイスドライバの分類

デバイスを，デバイスドライバコンポーネントの設計方針により，次のように分類した．

必須デバイス

どのアプリケーションでも必要となるであろうデバイス．具体的なデバイスは，コプロセッサ，ボタン，バッテリセンサ，Bluetooth である．

取り外し可能デバイス

取り外し可能であり高々1 つしか使用しないデバイス．具体的なデバイスは，LCD ディスプレイ，サウンドである．

可変数デバイス

取り外し可能であり複数使用するデバイス．具体的なデバイスは，モータ，各種センサである．

必須デバイスは取り外し可能とする必要はないため，デバイスドライバのコンポーネント記述として特別な記法を用いる必要はない．

取り外し可能デバイスは，アプリケーションによっては，そのデバイスドライバコンポーネントを生成しないデバイスである．そのため，デバイスドライバコンポーネントの受け口に結合されうる呼び口に対して，未結合のままにしてもよいことを表す指定子である，optional をつける．

可変数デバイスは，アプリケーションによって，使用する数が異なるデバイスである．そのため，デバイスドライバコンポーネントの受け口に結合されうる呼び口は，任意の数のコンポーネントを結合できるように呼び口配列を用いる．

```
1: /*セルタイプの定義：ミドルウェア*/
2: celltype tInitializeTaskNXTBody{
3:     /*省略*/
4:    /*サウンドの受け口と結合する呼び口*/
5:    [optional] call sSound cSound;
6:     /*省略*/
7: };
```

Figure 3.10: optional 指定の呼び口

3.5.3.3 必須デバイス

コプロセッサと Bluetooth のデバイスドライバは，3.5.3.1 節で述べたように複合コンポーネントとして開発した．

ボタンとバッテリセンサのデバイスドライバは，複合コンポーネントとする必要がなかったため，単体のコンポーネントとして開発した．

3.5.3.4 取り外し可能デバイス

LCD ディスプレイとサウンドのデバイスドライバは，3.5.3.1 節で述べたように複合コンポーネントとして開発した．

そして，要件 4 を満たすために，これらのデバイスドライバの受け口と結合するミドルウェアの呼び口は，未結合のままとしてもよいこととした．例として，サウンドを使用するミドルウェアのコンポーネント記述の一部を図 3.10 に，ミドルウェアの実装コードの一部を図 3.11 にそれぞれ示す． 図 3.11 中の 5 行目で呼び口が結合されているかどうかを判定しており，呼び口が結合されている場合のみ，7 行目でサウンドの API を呼び出す．関数 is_cSound_joined は，TECS ジェネレータによって生成される関数であり，呼び口が結合されているかどうかを返す．関数 cSound_beepTone は，TECS ジェネレータによって生成される関数であり，サウンドと結合している場合には，サウンドの受け口関数に，サウンドと結合していない場合には，NULL ポインタにそれぞれ割り当てられる．

```
 1: /*ミドルウェアの実装コード*/
 2: void eInitializeTaskNXT_main(){
 3:     /*省略*/
 4:    /*呼び口が結合されている場合のみ実行する*/
 5:    if(is_cSound_joined()){
 6:      /*サウンドを出力*/
 7:      cSound_beepTone(660, 100, 30);
 8:    }
 9:     /*省略*/
10: }
```

Figure 3.11: サウンドの使用

3.5.3.5　可変数デバイス

モータとセンサのデバイスドライバは，次に示す 2 種類のコンポーネントによって構成される．

(i) 各ポートに接続されているモータやセンサに対するドライバ

(ii) モータやセンサのポート全体に対するドライバ

(i) のコンポーネントは，アプリケーションからモータ回転速度の設定，モータ回転数の設定と取得，センサ値の取得を行うための API を提供し，さらに，初期化ルーチンや割込みサービスルーチンの各ポートに対する処理を行うためのコンポーネントである（以降，モータコンポーネント，または，センサコンポーネントと呼ぶ）．(ii) のコンポーネントは，初期化ルーチンや割込みサービスルーチンを呼び出す主体となるコンポーネント（以降，ポートコンポーネントと呼ぶ）であり，ポート全体の処理を行い，(i) のコンポーネントの処理を呼び出す．このような構成にした理由は，モータ（ロータリエンコーダ）とセンサ（I2C 通信）のための割込み要求ラインが，それぞれのポートで一つずつしかないためである．

例として，モータを 2 つ使用する場合のコンポーネント図を図 3.12 に示す．図 3.12 中の RightMotor と LeftMotor が，モータコンポーネントのセルであり，API となる受け口 eMotor をもつ．RightMotor と LeftMotor は，モータの回転速度を設定するため

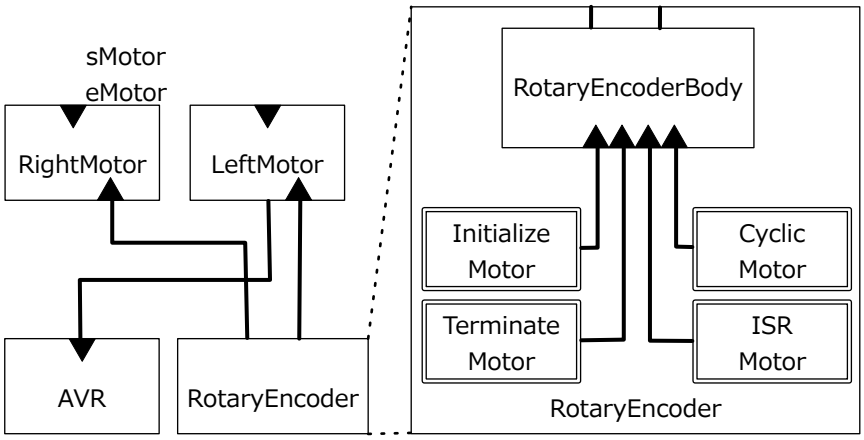

Figure 3.12: モータのコンポーネント図

に，コプロセッサのデバイスドライバ AVR と結合する．RotaryEncoder が，モータのポートコンポーネントのセルであり，デバイスドライバ本体，初期化ルーチン，終了処理ルーチン，周期ハンドラ，割込みサービスルーチンから構成される複合コンポーネントである．RotaryEncoder は，モータコンポーネントの処理を呼び出すために，RightMotor と LeftMotor と結合する．

ここで，要件 3，要件 4 を満たすために，必要なモータコンポーネントやセンサコンポーネントのみを生成できるようにし，さらに，生成するモータコンポーネントやセンサコンポーネントの数が，ポートコンポーネントの実装に影響しないことが必要である．これは，TECS における呼び口配列によって実現した．ポートコンポーネントのセルタイプの定義において，モータ（センサ）コンポーネントの受け口と結合する呼び口を呼び口配列として宣言し，さらに，その配列の要素数を省略することによって，組上げ記述で配列の要素数を決定することができる．その結果，実際に使用するモータ（センサ）コンポーネントの数に，呼び口配列の要素数を合わせることができる．例として，モータドライバにおけるポートコンポーネントのセルタイプ記述を図 3.13 に，ポートコンポーネントにおける割込みサービスルーチンの実装コードを図 3.14 にそれぞれ示す． 図 3.14 中で，9 行目の N_CP_ciMotorInterrupt は，TECS

```
 1: /*複合コンポーネントの定義：モータ（ポート）*/
 2: composite tRotaryEncoder{
 3:   /*呼び口配列*/
 4:   call siMotorInterrupt
           ciMotorInterrupt[];
 5:   /*ロータリエンコーダ*/
 6:   cell tRotaryEncoderBody
            RotaryEncoderBody{
 7:     ciMotorInterrupt =>
          composite.ciMotorInterrupt;
 8:     PIOBase = composite.PIOBase;
 9:     interruptNumber =
10:       composite.interruptNumber;
11:   };
12:    /*省略*/
13: };
```

Figure 3.13: モータのコンポーネント記述

```
 1: /*ポートコンポーネントの実装コード*/
 2: void eiInterruptBody_main(){
 3:   uint32_t pins;
 4:   int32_t i;
 5:   /*省略：CPU ロック*/
 6:   /*省略：pins にレジスタの値を読み込み*/
 7:   /*省略：割込み禁止に関する処理*/
 8:   /* Motor の回転数を更新 */
 9:   for(i=0;i < N_CP_ciMotorInterrupt;i++){
10:     /*呼び口配列の結合先を呼び出す*/
11:     ciMotorInterrupt_quadDecode(i, pins);
12:   }
13:   /*省略：CPU ロック解除*/
14: }
```

Figure 3.14: モータの割込みサービスルーチン

ジェネレータによって生成されるマクロであり，呼び口配列の要素数を示す．11 行目は，呼び口配列の i 番目の要素の結合先を呼び出すための記述である．よって，9-12 行目のように実装することで，呼び口配列の要素に依存せず，結合先の処理を呼び出すことができる．

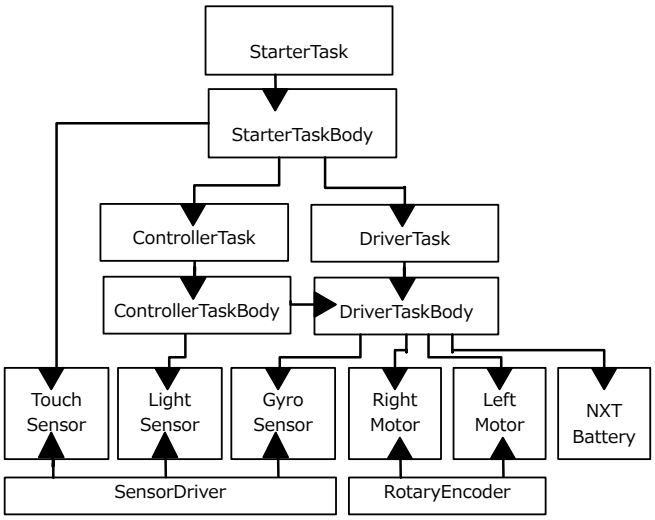

Figure 3.15: アプリケーションのコンポーネント図

3.6 事例および評価

本節では，ET ロボコンで用いられる，倒立二輪ライントレースロボットの制御ソフトウェアのサンプルプログラムをベースとした小規模なアプリケーションを事例として，ATON の有用性を示す．

ATON の有用性を評価するために，デバイスドライバの取り外しを行い，その容易性を，コード変更量が小さいことによって示す．さらに，メモリ使用量と割込みサービスルーチンの実行時間についても評価し，メモリ使用量や実行時間が小さいこと，および，デバイスドライバの取り外しによりメモリ使用量や実行時間を削減できることを示す．

3.6.1 ATON を用いたソフトウェアの構築

ATON を用いて，図 3.15 のようなコンポーネント図で表される構成のアプリケーションを構築することを考える．図 3.15 は，ソフトウェアの一部を表しており，ミドルウェアとログ出力機構のコンポーネント，および，モータ，センサ，バッテリセン

```
 1: /*右モータドライバ*/
 2: cell tMotor RightMotor{
 3:   cAVR = AVR.eAVR;
 4:   portNumber = C_EXP("NXT_PORT_B");
 5: };
 6: /*左モータドライバ*/
 7: cell tMotor LeftMotor{
 8:   cAVR = AVR.eAVR;
 9:   portNumber = C_EXP("NXT_PORT_C");
10: };
11: /*モータのポートドライバ*/
12: cell tRotaryEncoder RotaryEncoder{
13:   ciMotorInterrupt[0] =
14:     RightMotor.eiMotorInterrupt;
15:   ciMotorInterrupt[1] =
16:     LeftMotor.eiMotorInterrupt;
17: };
```

Figure 3.16: モータの組上げ記述

サとコプロセッサとの結合を省略してある．

RightMotor，LeftMotor，LightSensor，GyroSensor，TouchSensor，NXTBattery は，ATON の提供するデバイスドライバコンポーネントをインスタンス化したものである．RightMotor と LeftMotor はモータコンポーネントに，LightSensor，GyroSensor，TouchSensor は，光センサ，ジャイロセンサ，タッチセンサのセンサコンポーネントに，NXTBattery は，バッテリセンサのコンポーネントにそれぞれ対応する．NXTBattery は，必須コンポーネントであるため，アプリケーション開発者が生成する必要はない．モータに関する組上げ記述を図 3.16 に示す．

StarterTask，DriverTask，ControllerTask はタスクであり，ATON の提供するタスクコンポーネントを用いる．そして，それぞれのタスクコンポーネントに対応する，タスク本体のセルタイプを定義し，組上げ記述によって，タスクのセルと結合する．これらのタスクコンポーネントに関するコンポーネントの定義に関する記述の一部を図 3.17 に，組上げ記述の一部を図 3.18 にそれぞれ示す．

```
 1: /*定義：ドライバのシグニチャ*/
 2: signature sDrive{
 3:   void setTurn([in] int32_t turn);
 4: };
 5: /*定義：倒立制御タスクの本体*/
 6: celltype tDriverTaskBody{
 7:   /*タスクと結合する受け口*/
 8:   entry sTaskBody eBody;
 9:   /*sDrive 型の受け口*/
10:   entry sDrive eDrive;
11:   /*モータコンポーネンと結合する呼び口*/
12:   call sMotor cLeftMotor;
13:   /*省略：他の呼び口*/
14:   attr{ /*属性*/
15:     int32_t gyroOffset;
16:   };
17:   var{ /*変数*/
18:     int32_t turn;
19:   };
20: };
21: /*定義：制御計算タスクの本体*/
22: celltype tControllerTaskBody{
23:   /*sDrive 型の呼び口*/
24:   call sDrive cDrive;
25:   /*省略：受け口，他の呼び口，属性*/
26: };
```

Figure 3.17: タスクに関するコンポーネントの定義

3.6.2 デバイスドライバの取り外し

ATON の初期設定では，モータと各種センサを除く，提供するすべてのデバイスドライバコンポーネントを生成している．モータと各種センサについては，アプリケーション開発者によって，アプリケーションで必要なコンポーネントのみ生成する仕様としているため，初期設定では，すべてのモータとセンサに関するコンポーネントを生成していない．ここでは，アプリケーションにおいて，生成する必要のないデバイスドライバの取り外しについて述べる．

取り外しのできるデバイスドライバは，3.5.3.4 節で述べたように，LCD ディスプレイとサウンドのデバイスドライバである．これらのデバイスドライバコンポーネン

```
 1: /*倒立制御タスクの本体*/
 2: cell tDriverTaskBody DriverTaskBody{
 3:   /*モータコンポーネントとの結合*/
 4:   cLeftMotor = LeftMotor.eMotor;
 5:   /*省略：他の結合*/
 6:   /*属性の設定*/
 7:   gyroOffset = 599;
 8: };
 9: /*周期タスク：倒立制御タスク*/
10: cell tCyclicTask DriverTask{
11:   /*タスク本体との結合*/
12:   cBody = DriverTaskBody.eBody;
13:   /*タスク属性の設定*/
14:   priority = 3;
15:   /*省略：他の属性の設定*/
16: };
17: /*制御計算タスクの本体*/
18: cell tControllerTaskBody
          ControllerTaskBody{
19:   /*呼び口の結合*/
20:   cDrive = DriverTaskBody.eDrive;
21:   /*省略：他の結合，属性の設定*/
22: };
```

Figure 3.18: タスクに関する組上げ記述

```
1: /*サウンドドライバの生成*/
2: cell tSound Sound{
3: };
4: /*ミドルウェアの生成*/
5: cell tInitializeTaskBody
         InitializeTaskBody{
6:    /* サウンドドライバとの結合 */
7:   cSound = Sound.eSound;
8:   /*省略：他の結合*/
9: };
```

Figure 3.19: サウンドドライバの生成と結合

トを取り外すためには，組上げ記述の中から，コンポーネントを生成する記述と，コンポーネントを結合する記述をそれぞれ削除する．例として，サウンドのデバイスドライバコンポーネントを生成する記述，および，結合する記述を図 3.19 に示す．サウンドのデバイスドライバを取り外すためには，図 3.19 における，2-3 行目のデバイ

Table 3.2: メモリ使用量の比較

	FLASH ROM (byte)	RAM (byte)
(a)	29,264	10,720
(b)	26,448	9,872
(c)	22,784	8,896

スドライバコンポーネントを生成する記述，および，7 行目のようなデバイスドライバコンポーネントと結合する記述をすべて削除すればよい．これらの記述を削除することにより，デバイスドライバのオブジェクトモジュールがリンク対象外となり，初期化ルーチンや割込みサービスルーチンが登録されなくなる．さらに，TECS ジェネレータによって，インタフェースコードが生成されるため，ミドルウェアのソースコードを変更する必要がない．

ここで，デバイスドライバを取り外すことの有効性を示すために，比較実験を行った．比較内容は，3.6.1 節で述べたアプリケーションのメモリ使用量とし，比較対象は次の 3 種類とした．

(a) JSP プラットフォームを用いて開発したもの

(b) ATON を用いて開発し LCD ディスプレイとサウンドのデバイスドライバを取り付けたもの

(c) ATON を用いて開発し LCD ディスプレイとサウンドのデバイスドライバを取り外したもの

ここで，(b) から (c) に変更するために必要な修正行数は，8 行であった．メモリ使用量を比較した結果を表 3.2 に示す．表 3.2 より，ATON は，JSP プラットフォームと比較して，FLASH ROM で約 3KB，RAM で約 1KB のメモリ使用量を削減できたことが分かる．さらに，LCD ディスプレイとサウンドのデバイスドライバを取り外すことによって，FLASH ROM で約 3.5KB，RAM で約 1KB のメモリ使用量を削減できたことが分かる．

このような結果から，ATON では，数行のコンポーネント記述を削除することで，デバイスドライバを取り外すことができ，メモリ使用量を削減できることが確認できた．

次に，モータとセンサのコンポーネントについて，デバイスドライバを取り外すことの有効性を示すために，比較実験を行った．比較内容は，センサに関する割込みサービスルーチンの平均実行時間とし，比較対象は，ATON を用いて開発した 3.6.1 節で述べたアプリケーションについて，(d) 超音波センサを取り付けたものと，(e) 超音波センサを取り付けないものとした．割込みサービスルーチンの平均実行時間を比較した結果，(d) の実行時間は 4.88 μ秒，(e) の実行時間は 2.99 μ秒であった．この結果から，超音波センサを取り付けない場合，取り付けた場合と比較して，割込みサービスルーチンの実行時間が約 2 μ秒削減できていることが分かる．これは，超音波センサを一つ取り付けることによって，割込みサービスルーチン内で，超音波センサの値を取得するための通信処理が呼ばれたためである．

NXT では，超音波センサのようなデジタルセンサを用いる場合，センサの値を取得するために，メインプロセッサとセンサの間で I2C 通信を行う必要がある．ATON では，この I2C 通信を行うために，タイマ割込みを周期的に発生させ，そのタイマ割込みの割込みサービスルーチンの中で，センサポートごとに通信処理のためのサブルーチンを呼ぶ．この方法は，nxtOSEK や JSP プラットフォームにおける，I2C 通信処理においても用いられている．ただし，nxtOSEK や JSP プラットフォームでは，センサを使用していない場合，また，I2C 通信を行う必要がないセンサを用いた場合であっても，すべてのセンサポート（1〜4）に対して，上記のサブルーチンを呼び出し，I2C 通信処理を行ってしまう．そのため，センサを取り外したとしても，割込みサービスルーチンの実行時間は変化しない．一方，ATON では，センサが接続されていない場合はサブルーチンを呼ばず，I2C 通信を行う必要のないセンサが接続されている場合は空のサブルーチンを呼ぶようにした．そのため，センサを取り外すことにより，割込みサービスルーチンの実行時間を削減できた．

このように，デバイスドライバを外すことによって，割込みサービスルーチンの実行時間を削減できることが確認できた．

3.7 まとめ

本章では，TECS を用いて開発した，Mindstorms NXT 用ソフトウェアプラットフォーム ATON について述べた．ATON は，リアルタイム OS である ASP カーネルと，NXT のデバイスドライバ群とミドルウェアから構成されており，タスクやセマフォなどのカーネルオブジェクト，および，デバイスドライバ，ミドルウェアは，TECS の仕様に基づいてコンポーネント化されている．

ATON では，実際に使用する NXT のハードウェアに合わせて，アプリケーションソフトウェアで使用するデバイスドライバを選択できるようにした．また，TECS を用いているため，C 言語ベースでのソフトウェア開発が可能であり，モデルを用いたソフトウェア開発との親和性もある．

ET ロボコンのサンプルとして用いられている，倒立二輪ライントレースロボットの制御ソフトウェアを例に，ATON を用いたソフトウェア開発について紹介した．さらに，不要なデバイスドライバをソフトウェアから取り外すことによって，メモリ使用量や割込みサービスルーチンの実行時間を削減できることを示した．

CHAPTER 4

メモリ保護機能を持ったリアルタイム OS

4.1 概要

本章では，メモリ保護機能を持ったリアルタイム OS である，HRP2 カーネルについて述べる．HRP2 カーネルでは，静的コンフィギュレーションにより，メモリ配置を静的に行う．そして，MPU を用いたメモリ保護を実現するとともに，メモリ保護に必要な情報を静的に生成し，RAM 使用量のオーバヘッドを抑えることを可能とする．HRP2 カーネルの実用性を確かめるため，メモリ保護を実現したことによって生じる，実行時間やオブジェクトサイズのオーバヘッドを評価する．本章の研究内容による貢献は，以下のとおりである．

(2-1) ソフトウェアのコードやデータに対するメモリ保護の設定を静的に行い，それらのメモリ配置を静的に決定する方法を示す．

(2-2) MPU を抽象化してメモリ保護機能を提供する方法を示す．

(2-3) 提案するリアルタイム OS の実現方法を示し，メモリ使用量と実行時間のオー

バヘッドを明らかにする．

本章の構成は次のとおりである．まず，組込みリアルタイムシステムにおけるメモリ保護について述べる．次に，HRP2 カーネルの仕様について述べたあと，関連研究について述べ，HRP2 カーネルの実現方法について述べる．そして，HRP2 カーネルの実行時間やメモリ使用量のオーバヘッドを評価する．

4.2 組込みリアルタイムシステムにおけるメモリ保護

近年，組込みリアルタイムシステムは複雑化しており，また，高信頼化が求められるようになっている．汎用システムでは，複雑化するシステムの信頼性確保のために，メモリ領域への不正なアクセスを防ぐ，メモリ保護機能を持つことが一般的となっている．同様に，組込みリアルタイムシステムでも，メモリ保護機能を持つことが必要な場合がある．汎用システムでは，メモリ保護を実現するために，MMU を用いることが一般的であるが，MMU を用いる場合，TLB ミスの影響により実行時間を予測することが困難であり，厳しい時間制約のあるハードリアルタイムシステムでは，MMU を用いることは難しい．一方，組込みシステム向けのプロセッサには，MPU を搭載するものがある．MPU は，メモリ保護を実現するためのハードウェアであるが，MMU と異なり，TLB のようなキャッシュを必要とせず，実行時間の変動要因は存在しない．そのため，本研究では，厳しい時間制約が求められるリアルタイムシステムでメモリ保護を実現するために，MPU を用いることとした．

4.2.1 MMU

汎用システムでは，メモリ保護を実現するために，MMU を用いることが一般的である．MMU は，仮想記憶のためのアドレス変換，および，メモリアクセスの制限を行うためのハードウェアである．MMU は，アドレス変換を行う際に，指定されたページテーブルに従って，仮想アドレスを物理アドレスに変換する．ページテーブル

の各エントリには，そのページに対してどのようなアクセスが許可されているかを指定することができ，メモリ保護機能を実現できる．ここで，メモリアクセスが起こるたびに，ページテーブルから該当するエントリをサーチして読込む（ページテーブルウォークを行う）と，実行時間のオーバヘッドが非常に大きくなるため，TLB という，ページテーブルウォークの結果を保持しておくキャッシュを使用することが一般的である．TLB に該当するエントリがあれば，その結果を基にアドレス変換やアクセス権の確認を行う．TLB に該当するエントリがなかった場合は，TLB ミスとなり，ページテーブルウォークが行われる．コンテキストスイッチの際には，ページテーブルの各エントリに対するアクセス権の設定が変わるため，ページテーブルを更新し，TLB の内容をフラッシュする必要がある．コンテキストスイッチ時に TLB フラッシュの必要をなくすために，ASID という，コンテキストを識別する値を導入した MMU も存在する [2].

しかし，厳しいリソース制約やハードリアルタイム性の求められる組込みリアルタイムシステムでは，ページテーブルを管理するためのメモリ使用量のオーバヘッドが大きい，TLB ミスの影響によるリアルタイム性の保証が難しい，といった理由により，MMU を用いることは難しい [13]．また，組込みリアルタイムシステムでは，オブジェクトモジュール全体を一つのメモリ領域に配置するため，本来アドレス変換が不要という特徴もある．

4.2.2 MPU

MPU は，メモリ保護機能を実現するためのハードウェアであるが，MMU とは異なり，アドレス変換は行わない．MPU は，有限個の領域レジスタを持ち，各領域レジスタに対して，アドレス空間の範囲と許可するアクセスの種別，その領域レジスタの設定を有効とするかを，それぞれ設定する．ここで，MPU に設定するメモリ領域の開始番地と終了番地には，アライメント制約があり，その制約を満たす番地でなければ設定はできない．そして，MPU は，領域レジスタに設定された情報に従って，メ

Table 4.1: MPU の領域指定

	TMS570LS	LM3S6965	V850E2/MN4	SH72AW
コア	ARM Cortex-R4F	ARM Cortex-M3	V850E2M	SH2A
領域の設定	ビットマスク	ビットマスク	上下限, 一部ビットマスク	上下限
領域数	12	8	命令，定数用（上下限）5 データ用（上下限）5 データ用（ビットマスク）1	16
アライメント制約	2 のべき乗	2 のべき乗	16 の倍数（上下限） 2 のべき乗（ビットマスク）	4 の倍数
重複領域	優先度	優先度	論理和	論理和
ROM サイズ	2MB	256KB	1MB	768KB
RAM サイズ	160KB	64KB	64KB	32KB

モリアクセスを監視する．MPU によって，許可されていないメモリアクセスが検知されると，メモリアクセス違反の CPU 例外をプロセッサに対して発生させる．MPU を用いる場合，アドレス変換が行われないため，TLB を必要とせず，実行時間の変動要因となる TLB ミスは発生しない．また，コンテキストスイッチの際には，MMU 同様，MPU の設定を切り替えなければならないが，この切替えは，有限個の領域レジスタの設定を更新するだけでよく，設定の切替えにかかる時間は容易に予測できるため，リアルタイム性の保証がしやすい．

既存のプロセッサで，MPU を搭載するものとして，TMS570LS[57][4]，LM3S6965 [56] [3], V850E2M[50]，SH72AW[49] などが挙げられる．各プロセッサと MPU の特徴を，表 4.1 にまとめる．領域レジスタに対してメモリ領域を設定する方法には，上下限方式とビットマスク方式の，2 つがある．上下限方式では，各領域レジスタに対して，対象メモリ領域の開始番地と終了番地を設定する．ビットマスク方式では，各領域レジスタに対して，対象メモリ領域の開始番地と，開始番地から終了番地までの間に変化しない上位ビットのマスク（ビットマスク）を設定する．例えば，0x20000000 番地以上 0x20000200 番地未満のメモリ領域を領域レジスタに設定する場合，ビットマスク方式では，0x20000000 の番地と，0xFFFFFE00 のビットマスクを領域レジスタに設定する．そのため，ビットマスク方式では，2 のべき乗のサイズのメモリ領域しか設定できず，さらに，そのメモリ領域の開始番地は，メモリ領域のサイズの倍数と

Table 4.2: MMU と MPU の比較

	MMU	MPU
アドレス変換機能	○	×
メモリ使用量	×	○
実行時間の予測可能性	×	○
アクセス権の設定粒度	○	×
静的メモリ配置の必要性	なし	あり

する必要がある．一方，上下限方式では，設定できるメモリ領域のサイズや開始番地について，ビットマスク方式の制約ほど厳しいものではないが，ある整数の倍数となるように設定する必要がある．

また，あるメモリ領域が複数の領域レジスタに重複して設定されていた場合，その重複領域が扱われる方式には，優先度方式と論理和方式の 2 つがある．優先度方式では，重複領域が設定された領域レジスタの内で，ID の値が最も大きいレジスタの設定に従ってメモリアクセスを制限する．一方，論理和方式では，重複領域が設定された領域レジスタの内で，いずれかのレジスタで許可設定されているメモリアクセスを許可する．例えば，ある重複領域に対して，領域レジスタ 0 で書込みアクセスが許可され，領域レジスタ 1 で読出しと実行アクセスが許可されている場合，その重複領域に対するメモリアクセスは，優先度方式では読出しと実行アクセスのみ許可され，論理和方式では読出しと書込みと実行アクセスが許可される．

MMU と MPU とを比較した表を表 4.2 に示す．メモリ使用量について，MMU はページテーブルを保持する必要があり，メモリ使用量が大きくなる可能性があるが，MPU は，デバイスレジスタに設定する値だけを保持すればよく，メモリ使用量は小さい．実行時間の予測可能性については，MMU では TLB ミスの影響により，実行時間の予測が困難であるが，MPU では，実行時間の変動要因は存在しない．アクセス権の設定粒度について，MMU はページ単位でのアクセス権の設定が可能であり，細かい粒度でアクセス権を設定できるが，MPU では，有限個の領域レジスタに対してメモリ領域とアクセス権を設定することしかできず，細かいアクセス権の設定は困難

である．静的メモリ配置の必要性について，MMU の場合はページテーブルをシステム初期化時に生成すれば，静的にメモリ配置を行う必要性はないが，MPU では，領域レジスタの数が不足することないように同じアクセス権のコードやデータを連続して配置し，また，領域レジスタのアドレス制約を満たす必要があり，静的にメモリ配置を行う必要がある．

4.2.3 静的なメモリ配置の必要性

MPU は，領域レジスタの数が有限であるため，設定できるメモリ領域の数に制限がある．そのため，MPU の領域レジスタが不足しないように，同じアクセス権を設定する必要のあるコードやデータを，連続した番地に配置するように，静的にメモリ配置を行い，MPU に設定するメモリ領域の数を，MPU の領域レジスタ数以下にする必要がある．加えて，MPU では，設定するメモリ領域の開始番地やサイズに制限がある．そのため，MPU の制限に合わせて，設定するメモリ領域のアライメントを静的に適切に行う必要がある．

4.2.4 リアルタイム OS による MPU のサポート

組込みリアルタイムシステムの開発では，生産性向上のためにリアルタイム OS が使われている．リアルタイム OS は，ハードウェアを隠蔽，抽象化して，アプリケーションソフトウェアに提供する．表 4.1 に記載したように，MPU には様々な種類があり，領域レジスタの設定方式やメモリ領域のアラインメント制約がそれぞれ異なる．さらに，MPU のデバイスレジスタのデータ形式や操作方法も，MPU によって様々である．そこで，メモリ保護機能が必要なシステムで用いるリアルタイム OS では，MPU を隠蔽して，アプリケーションに提供することが求められる．具体的には，次のような要件を満たす必要があると考える．

- MPU に関する操作（初期化や設定の書換え，有効化・無効化）は，リアルタイ

ム OS の内部処理で行う．

- 同じアクセス権を設定する必要のあるコードやデータを，連続した番地に配置するように，リアルタイム OS が静的にメモリ配置を行う．このとき，MPU に設定するメモリ領域を，MPU の制限に合わせて適切にアライメントする．

- アプリケーション開発者は，MPU に設定する具体的な番地やサイズを知る必要がない．アプリケーション開発者は，セクションやオブジェクトファイル単位でメモリ保護属性の設定ができる．ここで，セクションとは，リンカスクリプトが扱う単位であり，同じメモリ配置の属性を持ったコードやデータのまとまりである．

4.3 HRP2 カーネルの仕様

HRP2 カーネルは，μITRON4.0 仕様の保護機能拡張である，μITRON4.0/PX 仕様 [61] をベースとして，拡張，および，改良を加えた，TOPPERS 新世代カーネル仕様 [60] の保護機能対応カーネルである．本節では，TOPPERS 新世代カーネル仕様のうち，本論文の主張であるメモリ保護機能に関する部分のみを述べるが，さらに詳細な説明やそれ以外の機能については，TOPPERS 新世代カーネル統合仕様書に記載されている．

4.3.1 基本概念

保護機能対応カーネルは，タスクが，許可されたメモリ領域に対して，許可されたアクセス種別でのみアクセスすることを許可し，それ以外のアクセスを防ぐ，メモリ領域のアクセス保護機能を持つ．ここで，メモリ領域とは，ある番地からある番地までの，連続したメモリの範囲である．保護機能対応カーネルのメモリ保護機能は，システムの実行中に発生する，不正なメモリアクセスを検出する．カーネルがアクセ

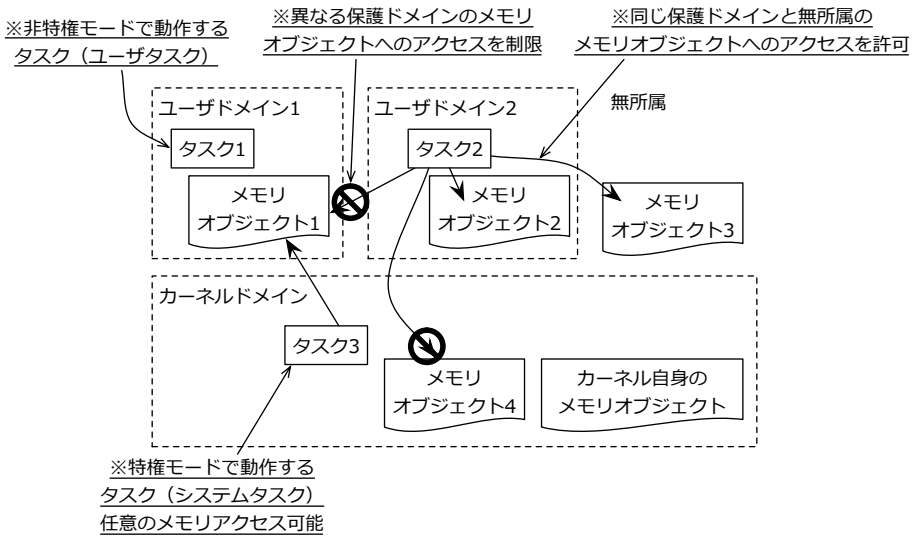

Figure 4.1: メモリ保護機能の概念図

ス保護の対象とするメモリ領域を，メモリオブジェクトと呼び，先頭番地，サイズ，アクセス保護属性（アクセス保護属性については後述する）によって特徴付けられる．メモリオブジェクトは互いに重なることはなく，あるメモリ領域に設定できるアクセス保護属性は単一である．

保護機能対応カーネルの提供するメモリ保護機能では，メモリオブジェクトをアクセス対象，保護ドメインという単位をアクセス主体とし，そして，メモリオブジェクトに対する，読出し，書込み，実行の操作を制御する．つまり，各メモリオブジェクトについて，読出し，書込み，実行の操作が，どの保護ドメインから許可されているのかを，それぞれ設定することとなる．ここで，例外として，タスクをアクセス主体として扱う，ユーザスタック領域というメモリオブジェクトも存在する．ユーザスタック領域については，4.3.4 節で詳しく述べる．

4.3.2 保護ドメイン

保護機能対応カーネルの提供するメモリ保護機能の概念図を，図 4.1 に示す．

保護機能を提供するために用いるカーネルオブジェクトの集合を，保護ドメイン

と呼ぶ．カーネルオブジェクトとは，カーネルが管理対象とするソフトウェア資源であり，タスクやメモリオブジェクトが，カーネルオブジェクトの例として挙げられる．カーネルオブジェクトは，高々一つの保護ドメインに属するものとする．ただし，タスクなどの処理単位は，必ずいずれかの保護ドメインに属するものとする．いずれの保護ドメインにも属さないカーネルオブジェクトを，無所属のカーネルオブジェクトと呼ぶ．

カーネルオブジェクトに対するアクセス保護属性は，保護ドメイン単位で設定し，タスクがカーネルオブジェクトにアクセスできるかどうかは，タスクの属する保護ドメインによって決まる．メモリオブジェクトに対するアクセス保護属性は，そのメモリオブジェクトに対する読出し，書込み，実行を許可する保護ドメインをそれぞれ設定する．ただし，タスクが，非特権モード（メモリ保護が有効な状態）で使用するスタック領域（ユーザスタック領域）は，そのタスクのみがアクセスできる．

システムには，カーネルドメインと呼ばれる保護ドメインが一つ存在し，ユーザドメインと呼ばれる保護ドメインが 0 個以上存在する．カーネルドメインに属するタスク（システムタスク）は，信頼性が高いプログラムを実行するものとみなされ，特権モード（メモリ保護が無効な状態）で実行され，すべてのメモリ領域に対して，任意のアクセスが可能である．ユーザドメインに属するタスク（ユーザタスク）は，カーネルドメインに属するプログラムに比べて信頼性が低いとみなされ，あるいは，予期せぬ障害の発生を検知するために，非特権モードで実行され，メモリ領域に対するアクセスを制限できる．

4.3.3 サービスコールと拡張サービスコール

サービスコールとは，アプリケーションソフトウェアからカーネルのソフトウェアを呼び出すインタフェースである．サービスコールによって呼び出されるカーネルのソフトウェアは，特権モードで実行される．つまり，ユーザタスクからサービスコールを呼び出した場合，カーネルのソフトウェアを実行する前に，非特権モードから特

権モードへの変換プロセスが実行され，カーネルのソフトウェアからユーザタスクに戻るときは，特権モードから非特権モードへの変換プロセスが実行される．また，HRP2 カーネルは，カーネルドメインに配置された関数を呼び出すための API である，拡張サービスコールを提供する．拡張サービスコールは，サービスコールと同様に，ユーザタスクから，特権モードで実行される処理を呼び出す．拡張サービスコールで呼び出すことのできる関数は，アプリケーション開発者によって静的に定義され，プログラム中では，cal_svc という API 記述で，拡張サービスコールの番号と引数を指定することによって呼び出される．ここで，拡張サービスコールに渡すことができる引数は，32 ビットのデータ 5 つである．

サービスコールの中には，ポインタを引数で受け取り，指定されたメモリ領域に対して，サービスコール内で読込み，または，書込みアクセスするものがある．このようなサービスコールを用いると，あるユーザタスクから，そのタスクではアクセスできない領域に対して，間接的にアクセスされる可能性がある．これは，サービスコールの処理が特権モードで実行されることによる．特権モードでは，MPU が無効となっており，不正なメモリアクセスをハードウェアで検知することができないためである．上述の不正なメモリアクセスを防ぐため，このようなサービスコールでは，サービスコールを呼び出した処理単位が，引数として受け取ったポインタの指すメモリ領域に対する，指定されたアクセスが許可されているかどうかを，サービスコール内で明示的に確かめる．指定されたメモリ領域へのアクセスが許可されていない場合には，メモリアクセスは実施せず，エラーコードをサービスコールの呼出し元に返す．

また，メモリ領域に対するアクセス権を確かめるためのサービスコールとして，prb_mem がある．prb_mem は，対象とするメモリ領域の開始番地とサイズ，アクセス種別，そして，アクセス主体となるタスクの ID を引数として受け取り，指定されたタスクが，そのメモリ領域に対してアクセス可能かどうかを確認し，その結果を呼び出し元に返す．

4.3.4 アクセス保護属性

メモリオブジェクトに対するアクセス保護属性として，少なくとも，次の 6 つの設定が可能である．

1. 専有リード属性
   メモリオブジェクトが属する保護ドメインのみに，読出しと実行アクセスのみを許可する．

2. 専有リードライト属性
   メモリオブジェクトが属する保護ドメインのみに，読出しと書込みと実行アクセスを許可する．

3. 共有リード属性
   すべての保護ドメインに，読出しと実行アクセスのみを許可する．

4. 共有リードライト属性
   すべての保護ドメインに，読出しと書込みと実行アクセスを許可する．

5. 共有リード専有ライト属性
   メモリオブジェクトが属する保護ドメインのみに，読出しと書込みと実行アクセスを許可し，他の保護ドメインには，読出しと実行アクセスのみを許可する．

6. ユーザスタック領域
   このメモリオブジェクトが，ユーザスタック領域として割り当てられたユーザタスクのみに，読出しと書込みと実行アクセスを許可し，他のユーザタスクは，いかなるアクセスも許可されない．

原則として，メモリオブジェクトに対するアクセス保護属性は，上記 1 から 5 に示したとおり，保護ドメインをアクセス主体として設定する．しかし，上記 6 に示したとおり，ユーザスタック領域だけは例外であり，タスクをアクセス主体として設定

Table 4.3: メモリ保護に関する静的 API

名称と引数	機能の概要
ATT_REG(region, {regatr, base, size})	番地 base から始まり，サイズ size 分のメモリ領域を，メモリリージョン名 region，メモリリージョン属性 regatr のメモリリージョンとして登録する．
ATT_SEC(section, {mematr, region}) ATA_SEC(section, {mematr, region}, apatr)	オブジェクトファイルに含まれ，セクション名が section であるセクションをメモリリージョン region に配置し，section が配置されたメモリ領域を，メモリオブジェクト属性 mematr のメモリオブジェクトとして登録する． ATA_SEC の場合には，登録するメモリオブジェクトのアクセス保護属性 apatr を指定する．
ATT_MOD(module) ATA_MOD(module, apatr)	オブジェクトファイル名が module であるオブジェクトファイルに含まれる，標準のセクションについて，それらのセクションが配置されたメモリ領域を，それぞれメモリオブジェクトとして登録する． ATA_MOD の場合には，登録するメモリオブジェクトのアクセス保護属性 apatr を指定する．
ATT_MEM({mematr, base, size}) ATA_MEM({mematr, base, size}, apatr)	番地 base から始まり，サイズ size 分のメモリ領域を，メモリオブジェクト属性 mematr のメモリオブジェクトとして登録する． ATA_MEM の場合には，登録するメモリオブジェクトのアクセス保護属性 apatr を指定する．
CRE_TSK(tskid, {tskatr, exinf, task, itskpri, stksz, stk})	タスクを生成する静的 API であり，タスクの名称 tskid，タスク属性 tskatr，タスクの起動時に引数として渡される情報 exinf，タスクの優先度 itskpri，スタック領域のサイズ stksz，スタック領域の開始番地 stk を指定する． ユーザタスクを生成する場合には，指定されたスタック領域を，メモリオブジェクト（ユーザスタック領域）として登録する．
DEF_SVC(fncd, {svcatr, extsvc, stksz})	拡張サービスコールを登録する静的 API であり，開始番地 extsvc の関数を，fncd の番号の拡張サービスコールとして登録する． また，拡張サービスコール属性 svcatr，拡張サービスコールで使用するスタックサイズ stksz を指定する．

する．MPU の領域レジスタ数が多いターゲットでは，上記以外のアクセス保護属性を設定させることも可能である．

4.3.5 コンフィギュレーションファイル

カーネルオブジェクトの登録は，ASP カーネルと同様に，オブジェクトの生成情報や初期状態などを定義するインタフェースである，静的 API によってのみ行われ，カーネルオブジェクトの所属する保護ドメインとアクセス保護属性は，すべて静的に設定する．メモリ保護機能に関する静的 API の名称と引数，および，機能の概要を，表 4.3 に示す．静的 API についての詳細な説明は，TOPPERS 新世代カーネル統合仕様書 [60] に記載されている．メモリオブジェクトを登録するための静的 API として，

ATT_SEC（ATTach SECtion）や ATT_MOD（ATTach MODule），ATT_MEM（ATTach MEMory）が存在する．ATT_SEC は，リンク対象のオブジェクトファイルに含まれる，指定されたセクションを，指定されたメモリリージョンに配置し，そのセクションが配置されたメモリ領域を，メモリオブジェクトとして登録する．ここで，メモリリージョンとは，セクションの配置対象となる，同じ性質を持った，連続したメモリ領域であり，具体例としては，SRAM や FLASH を表すものである．メモリリージョンにどのようなものがあるかは，ターゲットシステムごとに，コンフィギュレーションファイルに，静的 API である ATT_REG（ATTach REGion）を用いて記述する．メモリリージョンは，セクションが配置されるメモリの範囲を定義するものであり，アクセス保護属性には影響しない．アクセス保護属性を指定する方法については，本節で後述する．ATT_MOD は，指定されたオブジェクトファイルに含まれる標準のセクションについて，それらのセクションが配置されたメモリ領域を，それぞれメモリオブジェクトとして登録する．ここで，標準のセクションとは，コンパイラに特別な指定をしない場合に出力されるセクションである．標準のセクションの例としては，GNU プロジェクトの開発した C コンパイラである GCC における，.text セクションや.bss セクションが挙げられる．標準のセクションの名称や配置するメモリリージョンは，カーネルの開発者が指定し，コンフィギュレーションファイルには記述しない．ATT_MEM は，指定された開始番地とサイズで表されるメモリ領域を，メモリオブジェクトとして登録する．

メモリオブジェクトの登録例について述べる．プログラムのコード（機械語テキスト）や静的変数の配置されるメモリ領域をメモリオブジェクトとして登録するためには，そのコードや静的変数が含まれるセクション，または，オブジェクトファイルを，ATT_SEC，または，ATT_MOD を用いて指定する．また，局所変数が格納されるスタック領域をメモリオブジェクトとして登録するためには，タスクを生成するための静的 API である，CRE_TSK（CREate TaSK）を用いて指定する．ヒープ上の変数については，高信頼システムにおいて使用されない場合が多い．もし，ヒープが必要な

場合には，ヒープ用のメモリ領域を静的に確保し，そのメモリ領域をメモリオブジェクトとして登録しておき，システムの実行時に，そのメモリ領域から，ヒープ上の変数を格納する領域を割り当てるようにすればよいと考えられる．

カーネルオブジェクトに，それが属する保護ドメインを設定するためには，対象とするカーネルオブジェクトを生成，登録するための静的 API を，所属する保護ドメインの囲みの中に記述する．保護ドメインの囲みの記述は，カーネルドメインの場合には，“KERNEL_DOMAIN” キーワードに続く括弧であり，ユーザドメインの場合には，“DOMAIN(%DOMID%)” キーワードに続く括弧である．ここで，%DOMID%には，ユーザドメインの名称を指定する．保護ドメインの囲みの中に記述しなかったカーネルオブジェクトは，無所属のカーネルオブジェクトとなる．保護ドメインの設定例として，図 4.2 を用いて説明する．図 4.2 中の，TA_ACT という属性は，システム起動時にタスクを起動することを示すタスク属性，TA_NOWRITE という属性は，メモリオブジェクトへの書込みアクセスが禁止であることを示すメモリオブジェクト属性，TA_NULL という属性は，特別な属性を持たないことを示すタスク属性，および，メモリオブジェクト属性である．また，2-3 行目は，メモリリージョンを定義する静的 API の記述であり，0 番地から，768KByte 分のメモリ領域を，“FLASH” というメモリリージョンとして，0xfff80000 番地から，32KByte 分のメモリ領域を，“SRAM” というメモリリージョンとして，それぞれ定義している．4-9 行目のように，KERNEL_DOMAIN という囲みの中に，カーネルオブジェクトを生成，登録する静的 API を記述すると，そのカーネルオブジェクトは，カーネルドメインに属する．図 4.2 の例では，タスク MAIN_TASK と，セクション.sample_kernel に対応するメモリオブジェクトがカーネルドメインに属する．10-22 行目のように，DOMAIN(%DOMID%) という囲みの中に，カーネルオブジェクトを生成，登録する静的 API を記述すると，そのカーネルオブジェクトは，%DOMID%というユーザドメインに属する．図 4.2 の例では，タスク TASK1 と，セクション.sample1 に対応するメモリオブジェクトがユーザドメイン DOM1 に，タスク TASK2 と，セクション.sample2_1，.sample2_2 に対応するメモリ

```
 1:/* メモリリージョンの定義 */
 2:ATT_REG("FLASH",{TA_NULL,0,768*1024});
 3:ATT_REG("SRAM",{TA_NULL,0xfff80000,
                   32*1024});
 4:KERNEL_DOMAIN{/*カーネルドメイン*/
 5:  /*タスクの生成*/
 6:  CRE_TSK(MAIN_TASK,{TA_ACT,0,main_task,
                        6,1024,NULL});
 7:  /*メモリオブジェクトの登録*/
 8:  ATT_SEC(".sample_kernel",{TA_NULL,
                               "SRAM"});
 9:}
10:DOMAIN(DOM1){/*ユーザドメイン DOM1*/
11:  /*タスクの生成*/
12:  CRE_TSK(TASK1,{TA_NULL,0,task1,7,1024,
                    NULL});
13:  /*メモリオブジェクトの登録*/
14:  ATT_SEC(".sample1",{TA_NOWRITE,"FLASH"});
15:}
16:DOMAIN(DOM2){/*ユーザドメイン DOM2*/
17:  /*タスクの生成*/
18:  CRE_TSK(TASK2,{TA_NULL,0,task2,8,1024,
                    NULL});
19:  /*メモリオブジェクトの登録*/
20:  ATT_SEC(".sample2_1",{TA_NULL,"SRAM"});
21:  ATT_SEC(".sample2_2",{TA_NULL,"SRAM"});
22:}
23:/*無所属*/
24:/*メモリオブジェクトの登録*/
25:ATT_SEC(".sample_shared",{TA_NOWRITE,
                             "FLASH"});
```

Figure 4.2: 保護ドメインの設定例

オブジェクトがユーザドメイン DOM2 に，それぞれ属する．25 行目のように，保護ドメインの囲みの中以外の場所に，カーネルオブジェクトを生成，登録する静的 API を記述すると，そのカーネルオブジェクトは，無所属となる．図 4.2 の例では，セクション.sample_shared に対応するメモリオブジェクトが無所属となる．

次に，4.3.4 節で述べた，メモリオブジェクトのアクセス保護属性を，コンフィギュレーションファイルで指定する方法について述べる．メモリオブジェクトのアクセ

ス保護属性を指定する方法として，保護ドメインを用いる方法と，静的 API である，ATA_SEC，ATA_MOD，ATA_MEM （ATA：ATtach with Access control）を用いる方法の 2 通りがある．

まず，保護ドメインを用いて，メモリオブジェクトのアクセス保護属性を指定する方法について述べる．専有リード属性，または，専有リードライト属性を持ったメモリオブジェクトを登録するためには，保護ドメインの囲みの中に，ATT_SEC，ATT_MOD，ATT_MEM を記述する（図 4.2 の 8，14，20，21 行目）．このとき，メモリオブジェクト属性として，書込みアクセス禁止を指定すると，専有リード属性，そうでなければ，専有リードライト属性が，メモリオブジェクトのアクセス保護属性として設定される．また，共有リード属性，または，共有リードライト属性を持ったメモリオブジェクトを登録するためには，保護ドメインの囲みの中以外に，ATT_SEC，ATT_MOD，ATT_MEM を記述する（図 4.2 の 25 行目）．このとき，メモリオブジェクト属性として，書込みアクセス禁止を指定すると，共有リード属性，そうでなければ，共有リードライト属性が，メモリオブジェクトのアクセス保護属性として設定される．そして，ユーザスタック領域としてメモリオブジェクトを登録するためには，保護ドメイン（ユーザドメイン）の囲みの中に，CRE_TSK を記述する（図 4.2 の 12，18 行目）．このとき，CRE_TSK で指定されたスタック領域が，ユーザスタック領域のメモリオブジェクトとして登録される．なお，CRE_TSK で，スタック領域の開始番地に NULL を指定した場合には，コンフィギュレータが，指定されたサイズ分の配列を，スタック領域として確保し，そのスタック領域を，ユーザスタック領域のメモリオブジェクトとして登録する．ここで，共有リード専有ライト属性を持ったメモリオブジェクトについては，保護ドメインを用いる方法では登録できない．

次に，ATA_SEC，ATA_MOD，ATA_MEM を用いて，メモリオブジェクトのアクセス保護属性を指定する方法について述べる．ATA_SEC，ATA_MOD，ATA_MEM は，それぞれ，ATT_SEC，ATT_MOD，ATT_MEM の機能に加えて，登録したメモリオブジェクトのアクセス保護属性 apatr を直接指定する静的 API である．apatr は，4 つの

パラメータによって構成される情報であり，書込みアクセスな保護ドメインの集合，読出しアクセスと実行アクセスが可能な保護ドメインの集合，メモリオブジェクトの管理操作が可能な保護ドメインの集合，メモリオブジェクトの参照操作が可能な保護ドメインの集合を，それぞれ指定する．ここで，メモリオブジェクトの管理操作とは，メモリオブジェクトの登録や登録解除をするための操作であり，メモリオブジェクトの参照操作とは，メモリオブジェクトの設定情報を参照するための操作である．HRP2 カーネルでは，メモリオブジェクトの管理操作に対応しておらず，管理操作に関するアクセス保護属性の指定は無視する．ATA_SEC，ATA_MOD，ATA_MEM を用い，アクセス保護属性を直接指定することで，共有リード専有ライト属性を含む，任意のアクセス保護属性が設定可能である．ただし，ユーザスタック領域については例外であり，ATA_SEC，ATA_MOD，ATA_MEM では登録できない．また，保護ドメインの囲みの中に，ATA_SEC，ATA_MOD，ATA_MEM を記述した場合には，ATA_SEC，ATA_MOD，ATA_MEM で指定したアクセス保護属性が，メモリオブジェクトのアクセス保護属性として設定される．

4.4 関連研究

メモリ保護機能を持ったリアルタイム OS として，T-Kernel や RTEMS，L4 マイクロカーネルが挙げられるが，これらのリアルタイム OS では，MMU のみをサポートしており，MPU はサポートしていないため，4.2 節で述べた要求を満たさない．

MPU をサポートするリアルタイム OS として，FreeRTOS-MPU[23] と AUTOSAR OS[7] が挙げられる．

FreeRTOS-MPU は，カーネルオブジェクトの生成を動的に行うが，MPU を用いたメモリ保護を実現している．しかし，FreeRTOS-MPU は，次に挙げる理由により，MPU の隠蔽が十分にできていない．FreeRTOS-MPU では，アクセス保護属性の指定は，個々のタスクを生成する API に対して，タスクの実行中に，MPU の領域レジス

タに設定するメモリ領域の開始番地，サイズ，および，許可するアクセスの情報を指定することで行う．なお，この API で，タスクごとに独自に使用できる領域レジスタの数は 3 つまでであり，他の領域レジスタはカーネルが使用する（ユーザスタック領域，カーネルやデバイスドライバの使用するメモリ領域）．さらに，カーネルは，静的メモリ配置を実施しない．このような方法では，アプリケーション開発者は，同じアクセス保護属性のメモリ領域が連続するように，そして，MPU のアライメント制約を満たすように意識して，アプリケーションのメモリ配置を行う必要があり，さらに，MPU に設定する番地とサイズも取得する必要がある．

AUTOSAR OS では，MPU を用いたメモリ保護を実現するものとしており，カーネルオブジェクトの生成を静的に行うとしている．しかし，メモリ保護に関する静的コンフィギュレーションについては，仕様では未定義となっており，静的メモリ配置をカーネルでサポートできてない．また，AUTOSAR OS 仕様に準拠した，メモリ保護機能を持ったリアルタイム OS として，MICROSAR OS といった商用 OS は存在するが，OS の実装やそれに関連する文献が公表されているものは発見できなかった．

以上のように，MPU をサポートするリアルタイム OS はいくつか存在するが，MPU の隠蔽が十分であり，静的メモリ配置をサポートするものについては発見できなかった．

4.5 HRP2 カーネルにおけるメモリ保護機能の実現

本節では，HRP2 カーネルの実装の詳細について述べる．ただし，本論文の主張である，メモリ保護機能についてのみ述べる．HRP2 カーネルは，TOPPERS 新世代カーネル準拠のリアルタイム OS である，ASP カーネルに対して，保護機能を追加する形で実装する．ASP カーネルからの拡張点は，次のとおりである．

- 静的コンフィギュレーションにおいて，メモリ保護のための情報を生成する．さらに，メモリ配置をリンカに対して指定するリンカスクリプトを生成して，そのリンカスクリプトを基にオブジェクトモジュールの生成を行う．

- 保護機能対応カーネル仕様で追加された機能をサポートする．具体的には，4.3.3 節で述べた，サービスコール内で，引数として受け取ったポインタの指すメモリ領域に対するアクセス権を確認する機能と，サービスコール prb_mem をサポートする．

- MPU の操作（初期化，有効化と無効化，タスク切替えに伴う設定の書換え）をカーネル内部で行う．

4.5.1 設計目標

- MPU を用いたメモリ保護機能の実現

  4.2 節で述べたとおり，組込みリアルタイムシステムでは，MPU を用いたメモリ保護を実現することが重要である．そこで，静的なカーネルオブジェクトの生成情報から，MPU を用いたメモリ保護を実現する．MPU を用いることで，システムの実行中に発生する，不正なメモリアクセスを検出する．ただし，MPU の領域レジスタ数は隠蔽しない．MPU を用いたメモリ保護機能を実現するために，静的コンフィギュレーションにおいて，リンカスクリプトを生成し，MPU の制約を満たすメモリ配置を決定する．さらに，システムの実行中に，カーネルの内部処理で，MPU を操作する．今回の実装では，4.2.2 節で挙げた，SH2A，および，ARM Cortex-M3 の MPU を使用する．

- MPU の機能の活用

  4.3.3 節で述べたように，サービスコールの中には，指定されたメモリ領域に対するアクセス権を，ソフトウェアで明示的に確かめるものがある．このアクセス権を確かめる処理によって生じる，サービスコールの実行時間のオーバヘッド増加を抑えるようにする．ここで，MPU には，領域サーチ機能の持つものが存在する．領域サーチ機能とは，MPU の有効，無効に関わらず，MPU の領域レジスタに設定されている情報において，特定の番地に対して，どのようなアクセ

スが許可されているかを確かめる機能である [74]．サービスコールを呼び出したタスクからのアクセスが可能かどうかを確かめる場合，この領域サーチ機能を使用することで，アクセス権の確認にかかる時間を削減し，サービスコールの実行時間オーバヘッド増加をさらに抑えることが可能である．SH2A の MPU は，領域サーチ機能を持つため，アクセス権の確認をするときに領域サーチ機能を利用する．

- RAM 使用量の削減

  組込みリアルタイムシステムでは，RAM 領域のサイズが小さいことが多いため，メモリ保護機能を持つことによる，RAM 使用量の増加を抑えるようにする．そこで，メモリ保護のための情報は，可能な限り静的コンフィギュレーション時に生成するようにし，ROM 領域に配置するようにすることで，RAM の使用量を極力増加させないようにする．

- 既存ツールの使用

  汎用性の観点から，コンパイラやリンカなどの開発環境は，容易に入手可能な，既存のツールを使用可能なことが求められると考えられる．そこで，HRP2 カーネルを用いたソフトウェア開発を，既存のコンパイラやリンカの機能で実現できるようにする．今回の実装では，GNU のツールチェーンを用いる．

上記の設計目標と，HRP2 カーネル実現のための拡張点との対応関係を，図 4.3 に示す．

4.5.2 静的コンフィギュレーションにおける課題

HRP2 カーネルの静的コンフィギュレーションを実現するにあたり，以下の課題がある．

- メモリ配置の情報をリンカに対して与えるために，リンカスクリプトを生成する．

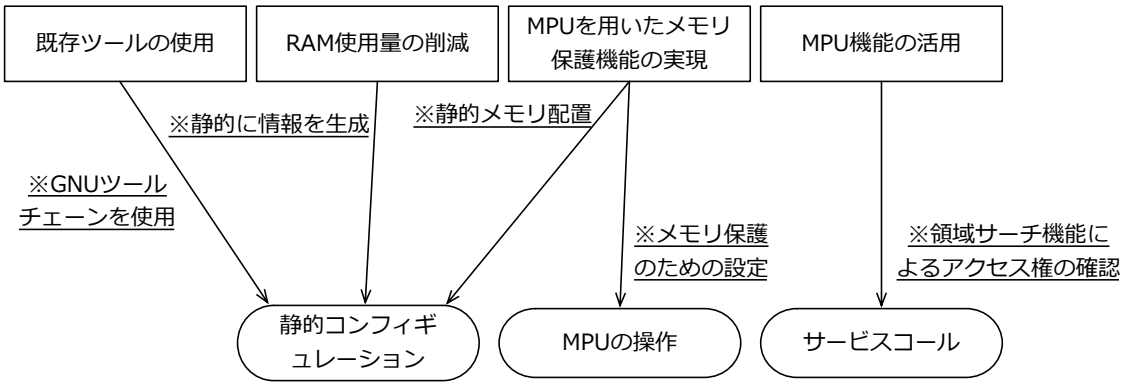

Figure 4.3: HRP2 カーネルの設計目標と拡張点の対応

- MPU の制約を満たすようにメモリ配置番地を決定する．

- RAM 使用量を削減するために，メモリ保護のための情報を静的に生成する．

前述のとおり，HRP2 カーネルは，メモリ配置を静的コンフィギュレーションで行う．メモリ配置を制御するためには，オブジェクトファイル群をリンクする際に，指定した順序でオブジェクトファイルやセクションを配置することを，リンカに対して命令する，リンカスクリプトが必要となる．そのため，HRP2 カーネルにおける静的コンフィギュレーションでは，このリンカスクリプトを，コンフィギュレーションファイルの情報からコンフィギュレータによって生成し，オブジェクトモジュールの生成時に使用する必要がある．

また，ARM Cortex-M3 の MPU では，SH2A の MPU と比べ，領域レジスタに設定するメモリ領域の開始番地に対する制約が厳しい．SH2A の MPU では，メモリ領域の開始番地を 4 の倍数にすればよく，その制約はリンカスクリプトによる命令で容易に実現できる（4.5.4 節で詳しく述べる）が，ARM Cortex-M3 の MPU では，メモリ領域の開始番地をそのメモリ領域のサイズの倍数にアラインメントする必要があるため，最終的なオブジェクトモジュールにおけるメモリ配置番地を決定するためには，セクションのサイズが必要となる．

さらに，HRP2 カーネルで扱う，メモリ保護のための情報の中には，最終的なオブジェクトモジュールにおいて，セクションが配置される具体的な番地やサイズを必要とするものがある．その情報の例として，MPU の領域レジスタを有効とするかどうかを設定するための情報が挙げられる．MPU の各領域レジスタは，4.5.5 節で述べるように，設定すべきアクセス保護属性を持ったメモリ領域が存在する場合にのみ有効とする．そのため，MPU の領域レジスタを有効とするかどうかを設定するための情報を生成する際には，それぞれのアクセス保護属性に対応するメモリ領域のサイズが必要となる．また，4.5.6 節で述べる，メモリ領域に対するアクセス権をソフトウェアで確認するときに用いる，メモリオブジェクトの管理情報を，開始番地の小さいものから順に並べたテーブルを生成するためには，セクションが配置される具体的な番地が必要となる．領域レジスタの値やメモリオブジェクトのテーブルは，必ずしも静的に生成しなければならないものではなく，システムの初期化時にも生成可能であるが，システムの初期化時に生成する場合，生成した情報は RAM 領域に置かなければならない．一方，これらの情報を，静的コンフィギュレーションで生成すると，その情報は ROM 領域に置くことが可能となるため，RAM 使用量を抑えることが可能となる．HRP2 カーネルでは，RAM 使用量を抑えるために，上記のような情報を可能な限り静的に生成することとする．

4.5.3 HRP2 カーネルにおける静的コンフィギュレーションの方針

前節の考察に基づき，HRP2 カーネルの静的コンフィギュレーションでは，コンフィギュレーションファイルの情報からリンカスクリプトを生成し，さらに，メモリ保護のための情報を，可能な限り静的に生成する．メモリ保護のための情報の中には，その生成時に，最終的なオブジェクトモジュールにおいて，セクションが配置される番地やサイズを必要とするものがあると述べた．この問題への対策として，オブジェクトモジュールを 2 回生成することとした．1 回目に生成するオブジェクトモジュールは，仮のデータ構造を用いて生成し，セクションが配置される番地やサイズ

を取得するために用いる．そして，その仮のオブジェクトモジュールから情報を抽出して，メモリ保護のための情報を生成した後，最終的なオブジェクトモジュールを生成する．ここで，ARM Cortex-M3 の場合には，仮のオブジェクトモジュールから得たセクションのサイズを基に，セクションの開始番地を決定し，再度リンカスクリプトを生成する．

上記の方法以外に，リンク対象のすべてのオブジェクトファイルから，セクションの名前とサイズを取得し，それらの情報とリンカスクリプトから，最終的なオブジェクトモジュールにおいて，セクションが配置される番地とサイズを計算する方法も考えられる．この方法では，オブジェクトモジュールの生成は 1 回で済むが，情報を抽出する対象のファイル数が多く，前者の方法よりも非効率的であると考える．前者の方法では，情報を抽出する対象のファイルは，仮のオブジェクトモジュール 1 つであり，また，2 回オブジェクトモジュールを生成することによって生じる不利益はないと考える．そのため，2 回オブジェクトモジュールを生成する方法を用いることとした．

4.5.4 HRP2 カーネルの静的コンフィギュレーション

HRP2 カーネルを用いたソフトウェア開発における，オブジェクトモジュールの生成手順の概略を，図 4.4 に示す．

まず，アプリケーション開発者は，コンフィギュレーションファイルにタスクなどのカーネルオブジェクトの生成情報や初期状態を記述する．さらに，カーネルオブジェクトの属する保護ドメインの設定，および，アクセス保護属性の設定を記述する．

次に，コンフィギュレーションファイルを，コンフィギュレータに入力し，カーネル構成初期化ファイルに加えて，メモリ構成初期化ファイル，リンカスクリプトを生成する．ここで，メモリ構成初期化ファイルとは，4.5.2 節で述べた，最終的なセクションの配置番地やサイズを必要とする情報を出力するファイルである．この段階では，最終的なセクションの配置番地やサイズを取得できないため，メモリ構成初期化

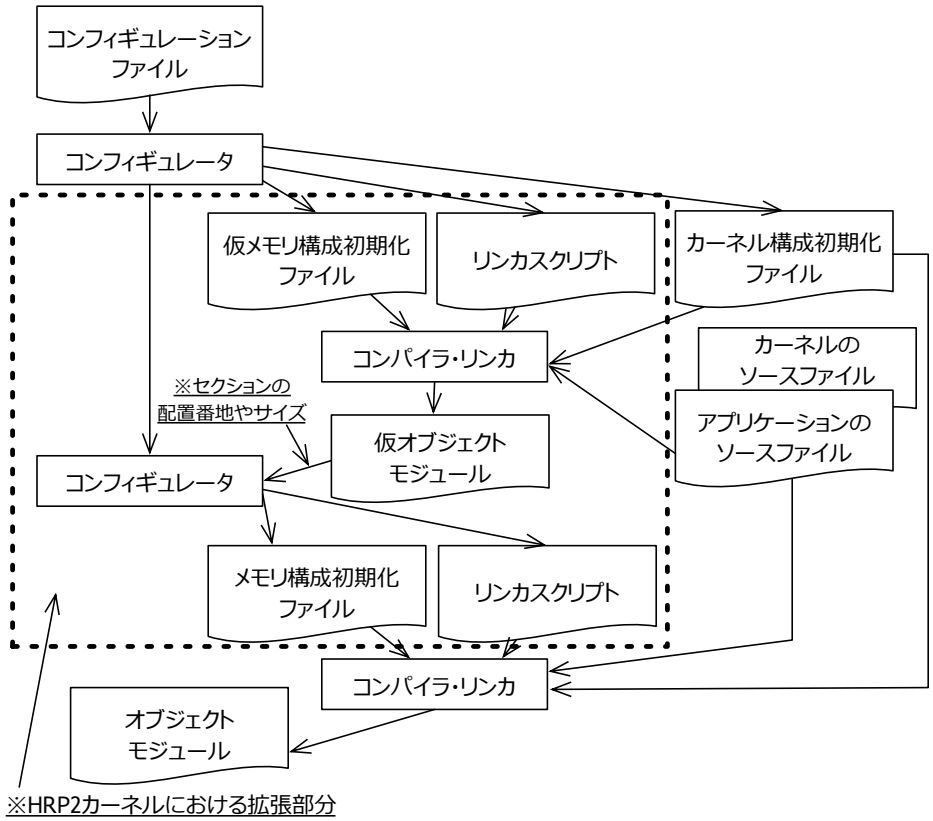

Figure 4.4: HRP2 カーネルにおける静的コンフィギュレーション

ファイルに出力すべき真の情報は生成できない．真の情報を出力する代わりに，最終的なメモリ構成初期化ファイルに出力する情報を格納するためのデータ構造を，仮情報として出力する．これは，仮のオブジェクトモジュールと最終的なオブジェクトモジュールで，メモリ配置が変化することを防ぐためである．

コンフィギュレータの生成するリンカスクリプトでは，静的 API によって登録されたセクションを，オブジェクトモジュールに出力する順序を制御する．このとき，使用する MPU の領域レジスタの数が不足することを避けるため，同じアクセス保護属性を持つセクションが連続して配置されるように出力順序を制御する．さらに，MPU のアライメント制約に合わせて，各アクセス保護属性に対応するメモリ領域の開始番地と終了番地をアライメントする．例えば，4.3.5 節の図 4.2 のコンフィギュレーションファイルを用いた場合に生成されるリンカスクリプトは，図 4.5 のようになり，こ

```
 1: MEMORY{/*セクションを配置するメモリ領域*/
 2:   FLASH:ORIGIN=0,LENGTH=786432
 3:   SRAM:ORIGIN=0xfff80000,LENGTH=32768
 4: }
 5: SECTIONS{/*セクションの配置指定*/
 6:   /*DOM1 の専有リード属性領域*/
 7:   .rx_DOM1 : {
 8:     ___start_rx_DOM1 = .;
 9:     *(.sample1)
10:   } > FLASH
11:   . = ALIGN(4); /*アライメント*/
12:   ___limit_rx_DOM1 = .;
13:   /*共有リード属性領域*/
14:   .rx_shared ALIGN(4) : {
15:     ___start_rx_shared = .;
16:     *(.sample_shared)
17:   } > FLASH
18:   . = ALIGN(4); /*アライメント*/
19:   ___limit_rx_shared = .;
20:   /*カーネルドメインの専有リードライト領域*/
21:   .rwx_kernel : {
22:     ___start_rwx_kernel = .;
23:     *(.sample_kernel)
24:     /*省略：特権モード時のスタック領域*/
25:   } > SRAM
26:   . = ALIGN(4); /*アライメント*/
27:   ___limit_rwx_kernel = .;
28:   /*省略：TASK1，TASK2 のユーザスタック領域*/
29:   /*DOM2 の専有リードライト属性領域*/
30:   .rwx_DOM2 ALIGN(4) : {
31:     ___start_rwx_DOM2 = .;
32:     *(.sample2_1)
33:     *(.sample2_2)
34:   } > SRAM
35:   . = ALIGN(4); /*アライメント*/
36:   ___limit_rwx_DOM2 = .;
37: }
```

Figure 4.5: リンカスクリプトの例

のリンカスクリプトを用いた場合のメモリ配置は，図 4.6 のようになる． 図 4.5 は，GNU のリンカスクリプト [25] の生成例である．GNU のリンカスクリプトでは，まず，“MEMORY” キーワードに続く括弧の中に，セクションの配置対象となるメモリ領域

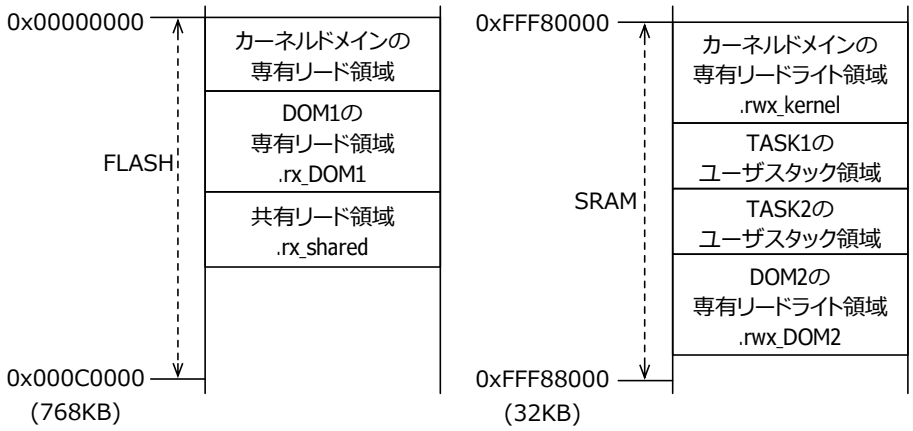

Figure 4.6: メモリ配置の例

を記述する．例えば，2 行目では，0 番地から，786432byte（768KB）分のメモリ領域に，FLASH というラベルを付けており，3 行目では，0xfff80000 番地から，32768byte（32KB）分のメモリ領域に，SRAM というラベルを付けている．これは，図 4.2 の 2-3 行目の情報から生成する．次に，“SECTIONS” キーワードに続く括弧の中に，出力セクションの配置する順序と配置するメモリリージョン，また，リンク対象のオブジェクトファイル中のセクションをどの出力セクションに割り当てるかといった情報を記述する．例えば，7-10 行目は，ユーザドメイン DOM1 の専有リード属性を持ったメモリ領域を表しており，オブジェクトモジュールの “.rx_DOM1” という出力セクションには，リンク対象のオブジェクトファイルの，“.sample1” というセクションが配置され，FLASH 領域に配置されることを表している．これは，図 4.2 の，14 行目の情報から生成する．30-34 行目は，ユーザドメイン DOM2 の専有リードライト属性を持ったメモリ領域を表しており，オブジェクトモジュールの “.rwx_DOM2” というセクションには，“.sample2_1” と “.sample2_2” というセクションが配置され，SRAM 領域に配置されることを表している．これは，図 4.2 の，20-21 行目の情報から生成する．ここで，32 行目，33 行目の “.sample2_1” と “.sample2_2” のように，同じアクセス保護属性を持つセクションを，連続した番地に配置し，同じアクセス権が設定されたコードやデータを，連続した番地に配置することを実現する．また，11 行目や 14 行目で

は，ALIGN(4) という命令により，メモリ領域の開始番地と終了番地を，4 の倍数の番地にアライメントしている．SH2A の MPU を用いる場合は，このアライメントにより，MPU のアライメント制約を満たすことを実現できる．一方，ARM Cortex-M3 の MPU を用いる場合は，メモリ領域のサイズを 2 のべき乗にする必要があり，さらに，メモリ領域の開始番地をそのサイズの倍数に整列する必要があるため，この段階における ALIGN 命令では，MPU のアライメント制約を満たすことが難しい．そのため，ARM Cortex-M3 の MPU を用いる場合には，次の段階でセクションの配置番地やアラインメントを計算し，リンカスクリプトを再度生成することで対応する．

その後の段階では，カーネル構成初期化ファイルとメモリ構成初期化ファイル，そして，カーネルのソースコードやアプリケーションのソースコードをコンパイルし，それらのオブジェクトファイル群を，コンフィギュレータの生成したリンカスクリプトを使用してリンクして，オブジェクトモジュールを生成する．ここで生成されるオブジェクトモジュールは，真のメモリ構成初期化ファイルを生成するために必要となる情報を抽出するための，仮のオブジェクトモジュールである．この仮のオブジェクトモジュールから，セクションの配置された具体的な番地やサイズを抽出する．そして，それらの情報から，メモリ保護に必要な情報を生成し，メモリ構成初期化ファイルに出力する．さらに，ARM Cortex-M3 の MPU を用いる場合は，セクションのサイズが 2 のべき乗となり，セクションの開始番地がそのサイズの倍数となるように，最終的なオブジェクトモジュールにおけるメモリ配置を決定し，再度リンカスクリプトを生成する．

オブジェクトモジュールから，具体的な番地やサイズを得る方法の一例を述べる．まず，コンフィギュレータの生成するリンカスクリプトで，図 4.5 にあるように，あるアクセス保護属性を持ったメモリ領域の開始番地と終了番地を，あるシンボルに代入し，そのシンボルをオブジェクトモジュールに含める．例えば，8 行目では，DOM1 の専有リード属性を持ったメモリ領域の開始番地を，___start_rx_DOM1 というシンボルに代入しており，12 行目では，DOM1 の専有リード属性を持ったメモリ領域の終

Table 4.4: MPU に設定する領域

MPU の領域	設定するメモリ領域	許可するアクセス
領域 0	共有リード属性を持つメモリ領域	読出し，実行
領域 1	共有リードライト属性を持つメモリ領域	読出し，書込み，実行
領域 2	共有リード専有ライト属性を持つメモリ領域の全体	読出し，実行
領域 3	ユーザスタック領域	読出し，書込み，実行
領域 4	専有リード属性を持つメモリ領域	読出し，実行
領域 5	専有リードライト属性を持つメモリ領域	読出し，書込み，実行
領域 6	共有リード専有ライト属性を持つメモリ領域	読出し，書込み，実行

了番地を，___limit_rx_DOM1 というシンボルに代入している．そして，GNU ツールチェーンの nm コマンドを使用し，オブジェクトモジュールの持つシンボルの情報を表示させ，その中から，上記のようなシンボルの情報を抽出する．その結果，あるアクセス保護属性を持ったメモリ領域の開始番地と終了番地を取得することができる．

最後に，真のメモリ構成初期化ファイルをコンパイルし，カーネル構成初期化ファイル，カーネル，アプリケーションのオブジェクトファイル群と再度リンクすることによって，最終的なオブジェクトモジュールを生成する．

4.5.5 MPU の操作

まず，MPU の使用方針について述べる．使用できる MPU の領域レジスタの数は 8 個までを想定する．そして，4.3.4 節で述べた，HRP2 カーネルが最低限サポートする，6 つのアクセス保護属性が実現可能となるように，領域レジスタを設定する．今回の実装では，MPU の各領域レジスタに対して，表 4.4 に示す対応関係で，アクセスを許可するメモリ領域を設定する．ここで，MPU の領域レジスタのうち，領域 3 には，実行中のタスクのユーザスタック領域を設定し，領域 4 から領域 6 までには，それぞれ，実行中のタスクが属する保護ドメインの専有リード属性，専有リードライト属性，共有リード専有ライト属性のメモリ領域を設定する．共有リード専有ライト領域は，領域 2 と領域 6 を組み合わせて実現する．領域 2 と領域 6 では，共有リード専有ライト領域のうち，実行中のタスクが属するユーザドメインの共有リード専有ライ

ト領域が重複して設定される．SH2A の MPU では，重複したメモリ領域が複数の領域レジスタに設定されている場合，その領域に対して許可するアクセスは，対象メモリ領域が設定されている領域レジスタのいずれかで許可されているアクセスとなる．一方，ARM Cortex-M3 の MPU では，重複したメモリ領域が複数の領域レジスタに設定されている場合，その領域に対して許可するアクセスは，対象メモリ領域が設定されている領域レジスタの中で，領域番号の最も大きいもので許可されているアクセスとなる．よって，表 4.4 の設定では，SH2A，ARM Cortex-M3 ともに，共有リード専有ライト領域のうち，実行中のタスクが属するユーザドメインの共有リード専有ライト領域には，読出し，書込み，実行が許可されており，それ以外の共有リード専有ライト領域には，読出し，実行のみが許可される．また，MPU の領域レジスタに設定されていないメモリ領域へのアクセスはすべて禁止する．

MPU に設定するための情報は，静的コンフィギュレーションで生成する．具体的には，MPU の各領域レジスタに設定する開始番地と終了番地，そして，MPU の各領域レジスタに対して，有効，無効を設定する値を生成する．これらの情報について，ユーザドメインの専有領域（領域 4 から領域 6 まで）に対してはユーザドメインごとに，ユーザタスクの専有領域（領域 3）に対してはユーザタスクごとに，共有領域（領域 0 から 2 まで）に対しては全体で 1 つ，それぞれ生成する．ここで，MPU の各領域レジスタを有効とするかどうかは，設定するメモリ領域のサイズで判断する．設定するメモリ領域のサイズが 0 より大きい場合は，対応する MPU の設定領域を有効に設定する．

カーネルが MPU に対して行う操作は，初期化，有効化と無効化，タスク切替えに伴う設定の書換えの 3 つに分類される．ここで，各処理について述べる．

4.5.5.1 初期化

MPU の初期化は，システムの初期化処理で行う．共有領域（領域 0 から領域 2 まで）の開始番地と終了番地を設定し，領域 0 から領域 6 までの各設定領域に対して，

許可設定をするアクセスの種別をそれぞれ設定する．例えば，領域 0 に対しては，読出しと実行アクセスを許可する設定をする．

4.5.5.2 有効化と無効化

MPU の有効化は，ユーザタスクの起動時，カーネルドメインの処理からユーザタスクの処理に戻るときに行う．そして，MPU の無効化は，カーネルドメインの処理を呼び出すときに行う．カーネルドメインの処理としては，サービスコールや割込みハンドラ，CPU 例外ハンドラなどが挙げられる．

4.5.5.3 タスク切替えに伴う設定の書換え

MPU 設定の書換えは，実行するタスクが切り替わるとき（ディスパッチ時）に行う．ここで，書換えの対象となる可能性のある領域は，ユーザドメインの専有領域（領域 4 から領域 6 まで），および，ユーザタスクの専有領域（領域 3）である（図 4.7）．

MPU 設定を書き換える処理は，ディスパッチ前後のタスクがそれぞれ属する保護ドメインによって異なる．図 4.8 は，ディスパッチ処理の流れを表したものである．図 4.8 のうち，HRP2 カーネルで拡張した部分について説明する．

まず，ディスパッチ後のタスクが属する保護ドメインを確認する．ディスパッチ後のタスクがカーネルドメインに属する場合には，MPU 設定は書き換えない．

次に，MPU のユーザドメイン専有領域に対して，どのユーザドメインの情報が設定されているかを確認する．MPU のユーザドメイン専有領域に設定されている情報が，ディスパッチ後のタスクが属するユーザドメインの情報である場合には，ユーザタスクの専有領域のみを，ディスパッチ後のタスクのユーザスタック領域で書き換える．

そして，ディスパッチ後のタスクがユーザドメインに属し，かつ，MPU のユーザドメイン専有領域に設定されている情報が，ディスパッチ後のタスクが属するユーザドメインの情報でない場合には，ユーザタスクの専有領域を，ディスパッチ後のタス

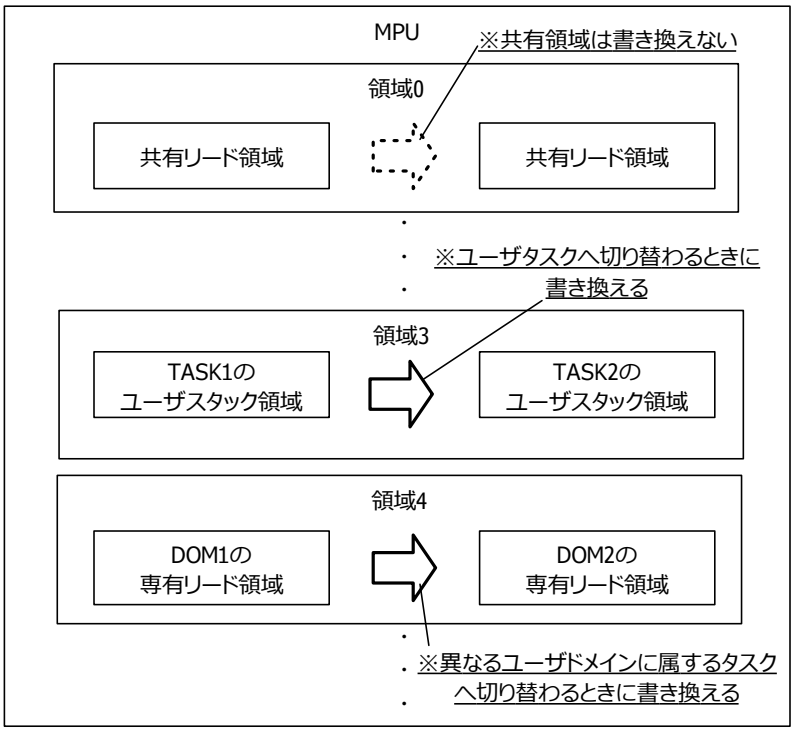

Figure 4.7: MPU 設定の書換え

クのユーザスタック領域で，そして，ユーザドメインの専有領域を，ディスパッチ後のタスクが属するユーザドメインの情報で書き換える．

4.5.6 サービスコール内でのアクセス権の確認

サービスコールの中には，ポインタを引数で受け取り，指定されたメモリ領域に対して，サービスコール内でリード，または，ライトアクセスするものがある．例えば，ref_tsk というサービスコールは，指定されたタスクの現在の実行状態や優先度などの情報を取得するサービスコールであるが，この ref_tsk は，タスクの情報を入れるパケットへのポインタを引数として受取り，そのポインタが指すメモリ領域へタスクの情報を書き込む．このようなサービスコールでは，4.3.3 節で述べたように，引数として受け取ったポインタの指すメモリ領域に対して，指定されたアクセスが許可されているかどうかを，ソフトウェアで明示的に確認する必要がある．

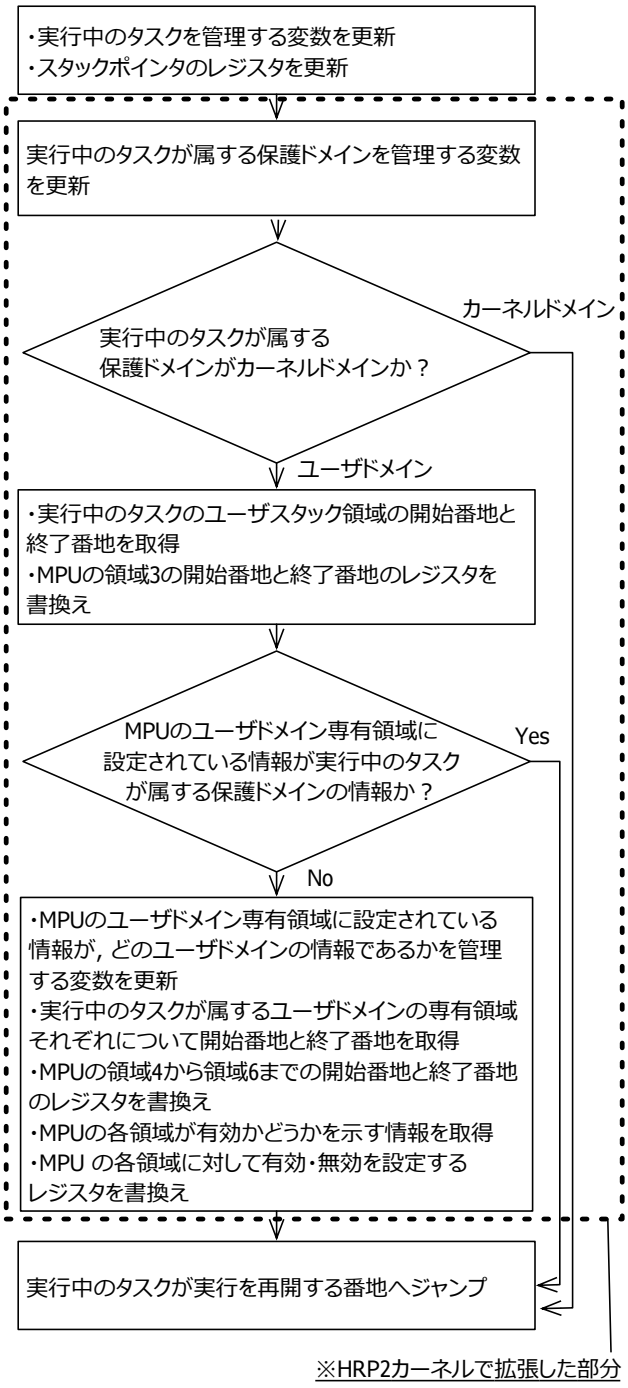

Figure 4.8: ディスパッチ処理の流れ

また，4.3.3 節で述べた，メモリ領域へのアクセス権を確認するサービスコールである，prb_mem においても同様に，指定されたメモリ領域に対して指定されたアクセ

スが許可されているかどうかを，ソフトウェアで明示的に確認する必要がある．

このアクセス権の確認を行うために，静的コンフィギュレーションにおいて，メモリオブジェクトごとに，開始番地とサイズ，アクセス保護属性をまとめた管理情報を生成する．そして，その管理情報をサーチし，対象となるメモリ領域のアクセス保護属性を取得し，指定されたアクセスが許可されているかどうかを確認する．ここで，メモリオブジェクトの管理情報のテーブルを単純に生成すると，メモリオブジェクトの数 N に対して，O(N) の時間が管理情報のサーチに必要となり，非効率的であることに加え，実行時間の予測が難しい．そこで，静的コンフィギュレーション時に，メモリオブジェクトの管理情報を，開始番地の小さいものから順に並べて，テーブルを生成する．開始番地の順にメモリオブジェクトの管理情報を並べることで，管理情報のサーチを，メモリオブジェクトの開始番地を用いたバイナリサーチで処理することができる．その結果，管理情報のサーチにかかる時間は，O(logN) に短縮でき，さらに，実行時間の予測も容易である．

一方，SH2A の MPU には，4.5.1 節で述べたような領域サーチ機能がある．ここで，サービスコールの処理を実行している間の MPU の設定は，サービスコールを呼び出したときの設定，つまり，サービスコールを呼び出したユーザタスクの処理を実行している間の MPU の設定と同一である．よって，この領域サーチ機能を用いることで，指定されたメモリ領域に対する，指定されたアクセスが，サービスコールを呼び出したユーザタスクに許可されているかどうかを確認することができる．アクセス権の確認を，MPU の機能を用いて実現したソースコードを図 4.9 に示す．図 4.9 のように，アクセス権の確認は，対象番地の設定，領域サーチの実行，領域サーチ結果の取得，アクセス権の確認，という処理を，開始番地と終了番地のそれぞれについて実行すればよい．さらに，アクセス権の確認にかかる時間は，メモリオブジェクトの数に依存せず，一定の値，つまり O(1) である．

以上のように，サービスコールを呼び出したタスクからのアクセスが可能かどうかを確かめる場合，MPU の領域サーチ機能を用いて実現することで，アクセス権の

```
 1:/* 指定されたメモリ領域がユーザタスクから
 2:    書込みアクセス可能ならばtrueを返す. */
 3:bool_t
 4:probe_mem_write(void *base, SIZE size)
 5:{
 6:  uint32_t res;
 7:  {/* 開始番地をサーチ */
 8:    /* サーチする番地をレジスタに設定 */
 9:    mpu_write_base( base );
10:    /* 領域サーチ開始レジスタに値を書込み */
11:    mpu_write_control( DO_SEARCH );
12:  }
13:  /* 開始番地のサーチ結果をレジスタから取得 */
14:  res = mpu_read_flag();
15:  /* 開始番地に書込みアクセス可能か確認 */
16:  if((res & WRITE_ACCESS_BIT) == 0){
17:   return false;
18:  }
19:  {/* 終了番地をサーチ */
20:    /* サーチする番地をレジスタに設定 */
21:    mpu_write_base( base + size - 1 );
22:    /* 領域サーチ開始レジスタに値を書込み */
23:    mpu_write_control( DO_SEARCH );
24:  }
25:  /* 開始番地のサーチ結果と終了番地のサーチ
26:     結果が一致するかどうかを確認する. */
27:  if(mpu_read_flag() != res){
28:    return false;
29:  }
30:  return true;
31:}
```

Figure 4.9: 領域サーチ機能を用いた書込みアクセス権の確認

確認による，サービスコールの実行時間オーバヘッドをさらに抑えることができる．

Table 4.5: ターゲットプロセッサの仕様

	SH72AW	LM3S6965
CPU コア	SH2A	ARM Cortex-M3
CPU 動作クロック	160MHz	50MHz
ROM サイズ	768KB	256KB
RAM サイズ	32KB	64KB

4.6 評価実験

4.6.1 評価環境

ターゲットプロセッサとして，SH2A の MPU を搭載した，ルネサスエレクトロニクス社製の SH72AW[49]，および，ARM Cortex-M3 の MPU を搭載した，Texas Instruments 社製の LM3S6965 [56] を用いた．SH72AW と LM3S6965 の仕様を表 4.5 に示す．ここで，実行時間の計測には，計測時間の分解能が 0.1μ 秒のタイマを使用し，キャッシュを有効にして計測した．また，評価結果で述べる実行時間は，10,000 回試行を繰り返して得られた最大実行時間とした．

4.6.2 評価項目

まず，カーネルの基本性能として，サービスコールの実行時間を評価する．メモリ保護を実現する上で，実行時間オーバヘッドが生じる処理として，4.5.5.3 節で述べた，実行タスクを切り替える処理がある．そこで，実行タスクの切り替えが生じるサービスコールである，act_tsk の実行時間オーバヘッドについて評価する．act_tsk は，タスクを起動するためのサービスコールであり，act_tsk を呼び出したタスクよりも，act_tsk によって起動されたタスクの方が優先度が高い場合に，実行タスクの切り替えが生じる．評価においては，ASP カーネルにおける act_tsk の実行時間を，HRP2 カーネルにおける act_tsk の実行時間と比較した．

また，4.5.1 節で，組込みリアルタイムシステムでは，RAM 領域のサイズが小さいことが多く，RAM 使用量のオーバヘッドを抑えることが重要であると述べた．そこで，HRP2 カーネルで，RAM 使用量のオーバヘッドが抑えられているかを評価する．評価においては，HRP2 カーネルを用いてアプリケーションを開発したときの，カーネルのオブジェクトサイズを，同じアプリケーションについて，保護ドメインとメモリオブジェクトを取り除き，ASP カーネルを用いて開発したときの，カーネルのオブジェクトサイズと比較した．評価に用いたアプリケーションの違いについては，4.6.4 節で詳しく述べる．ここで，カーネルのオブジェクトサイズとは，カーネルのソースコード，および，コンフィギュレータによって生成された，カーネル構成初期化ファイルとメモリ構成初期化ファイルをコンパイルしたオブジェクトファイルのオブジェクトサイズを合計したものである．

4.3.3 節で，サービスコールの中には，指定されたメモリ領域に対するアクセス権を，ソフトウェアで確かめるものがあると述べた．このアクセス権の確認を内部で行うサービスコールについて，実行時間を評価した．評価対象としたサービスコールは，4.5.6 節で述べた，ref_tsk とした．ref_tsk は，指定されたタスクの現在の実行状態や優先度などの情報を取得するサービスコールであり，ref_tsk の引数には，書込み可能なメモリ領域へのポインタが含まれる．アクセス権の確認方法としては，4.5.6 節で述べた，ソフトウェアのみで実現する方法と，MPU の領域サーチ機能を用いる方法の，2 通りの方法について評価した．MPU の領域サーチ機能は，ARM Cortex-M3 の MPU では存在しないため，SH72AW でのみ比較評価した．

4.2 節で，組込みリアルタイムシステムでは，ハードリアルタイム性を保証するために，MPU を用いたメモリ保護を実現することが重要であると述べた．そこで，HRP2 カーネルで，MPU を用いたメモリ保護により，リアルタイム性の保証が容易となっているかどうかについて考察する．

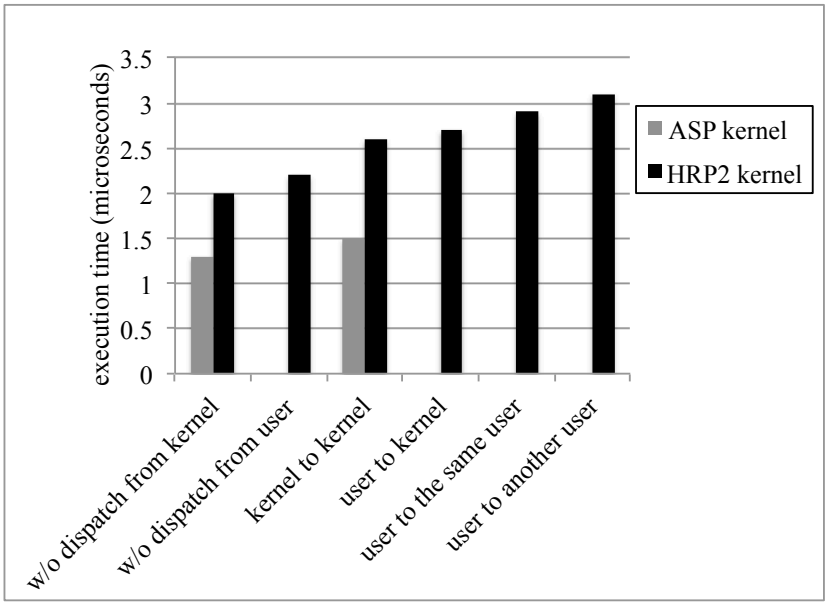

Figure 4.10: SH72AW における act_tsk の実行時間（μ 秒）

4.6.3 サービスコールの実行時間

サービスコールの実行時間のオーバヘッドを評価するため，act_tsk の実行時間を，ASP カーネルと HRP2 カーネルを用いた場合のそれぞれで計測した．act_tsk を呼び出したときに，実行タスクが切り替わらない（ディスパッチしない）場合と，実行タスクが切り替わる（ディスパッチする）場合のそれぞれについて，実行時間を計測した．ディスパッチしない場合の実行時間は，act_tsk を呼び出してから，呼び出し元に戻るまでの時間とし，ディスパッチする場合の実行時間は，act_tsk を呼び出してから，ディスパッチ後のタスクの起動番地に到達するまでの時間とした．ここで，HRP2 カーネルでの act_tsk の呼出しでは，ディスパッチしない場合には，サービスコール呼出しに伴う実行時間オーバヘッドが生じ，ディスパッチする場合には，サービスコール呼出しに伴うオーバヘッドに加えて，ディスパッチ処理に伴うオーバヘッドが生じる．

act_tsk の実行時間を計測したパターンと計測結果を，SH72AW と LM3S6965 のそれぞれについて，図 4.10 と図 4.11 に示す． ASP カーネルでは，保護ドメインの概念が存在しないため，ディスパッチしない場合とディスパッチする場合の 2 通り計測し，

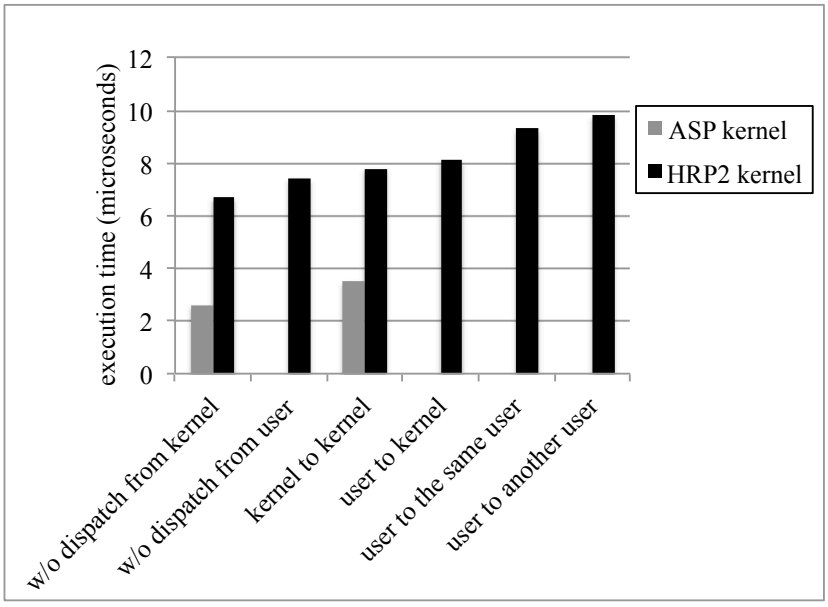

Figure 4.11: LM3S6965 における act_tsk の実行時間（μ 秒）

HRP2 カーネルでは，act_tsk を呼び出したタスクが属する保護ドメインと，ディスパッチ処理後のタスクが属する保護ドメインの組合せを替えて計測をし，合計 6 通り計測した．図 4.10，および，図 4.11 のグラフは左から順に，システムタスクからディスパッチしない act_tsk を呼び出した場合（ASP カーネルでディスパッチしない場合），ユーザタスクからディスパッチしない act_tsk を呼び出した場合，システムタスクから別のシステムタスクにディスパッチする場合（ASP カーネルでディスパッチする場合），ユーザタスクからシステムタスクにディスパッチする場合，ユーザタスクから同じユーザドメインのタスクにディスパッチする場合，ユーザタスクから別のユーザドメインのタスクにディスパッチする場合の実行時間をそれぞれ表している．

まず，ディスパッチしない場合の実行時間をそれぞれ比較すると，HRP2 カーネルを用いた場合，システムタスクから呼び出すと，SH72AW では 0.7μ 秒，LM3S6965 では 4.1μ 秒の増加がみられ，ユーザタスクから呼び出すと，SH72AW では 0.9μ 秒，LM3S6965 では 4.8μ 秒の増加がみられる．これは，サービスコールを呼び出すときに，MPU を無効にする処理，そして，サービスコールから戻るときに，MPU を，サー

ビスコール呼出し時の状態に戻す処理がそれぞれ実行されるオーバヘッドであると考えられる．ここで，SH72AW と LM3S6965 で CPU の動作クロックに違いはあるものの，実行時間の増加量が大きく異なっている．これは，プロセッサの仕様の違いにより，MPU を無効にする処理を，SH72AW では MPU の制御レジスタを直接書き換えることで実現しているが，LM3S6965 ではソフトウェア割込みを利用して実現しているためである．MPU の制御レジスタを直接書き換える方法では，実行時間は短いが，ソフトウェアで MPU の制御レジスタを書き換える必要があり，かつ，その制御レジスタにアクセスできるようにするため，4.5.5 節で述べた領域レジスタの設定に加えて，1 つの MPU 領域を，MPU の制御レジスタの I/O 領域のために使用しなければならない．一方，ソフトウェア割込みを利用する方法では，実行時間は長いが，MPU の無効化はハードウェアで自動的に行われる．

次に，システムタスクから act_tsk を呼び出し，別のシステムタスクに切り替わる場合の実行時間を比較すると，HRP2 カーネルを用いた場合，SH72AW では 1.1μ 秒，LM3S6965 では 4.3μ 秒の増加がみられる．これは，前述のサービスコールを呼び出す処理に加え，ディスパッチ処理において，ディスパッチ後のタスクが属する保護ドメインを確かめる処理が実行されるオーバヘッドであると考えられる．ただし，ディスパッチしない場合のサービスコール呼出しでは，サービスコールの出口処理が実行されるが，ディスパッチする場合のサービスコールでは，サービスコールの出口処理でなく，ディスパッチ先のタスクの起動処理が実行されるため，その差分は存在する．

システムタスクから別のシステムタスクに切り替わる場合と，ユーザタスクからシステムタスクに切り替わる場合を比較すると，後者の方が，SH72AW では 0.1μ 秒，LM3S6965 では 0.3μ 秒だけ実行時間が長くなっている．これは，ユーザタスクからサービスコールを呼び出した場合，システムタスクからサービスコールを呼び出した場合よりも，MPU の状態を管理する変数の処理など，実行される処理が増えるためであると考えられる．

ユーザタスクからシステムタスクに切り替わる場合と，ユーザタスクから同じ保

護ドメインに属するタスクに切り替わる場合を比較すると，後者の方が，SH72AW では 0.2μ 秒，LM3S6965 では 1.2μ 秒だけ実行時間が長くなっている．これは，後者の場合には，MPU の領域レジスタのうち，ユーザタスク専有の領域を書き換える処理，および，MPU のユーザドメイン専有領域に設定されている情報が，ディスパッチ後のタスクが属する保護ドメインの情報かどうかを確かめる処理が増えるためであると考えられる．また，LM3S6965 では，実装の都合上，ユーザタスクの起動処理においてソフトウェア割込みを使用する必要があったため，SH72AW よりも実行時間の増加量が大きくなっている．

ユーザタスクから同じ保護ドメインに属するタスクに切り替わる場合と，ユーザタスクから別の保護ドメインに属するユーザタスクに切り替わる場合を比較すると，後者の方が，SH72AW では 0.2μ 秒，LM3S6965 では 0.5μ 秒だけ実行時間が長くなっている．これは，後者の場合には，MPU の領域レジスタのうち，ユーザドメイン専有の領域を書き換える処理が増えるためであると考えられる．

そして，HRP2 カーネルで act_tsk を呼び出したときの実行時間は，ASP カーネルで act_tsk を呼び出したときの実行時間と比較して，最大で約 3 倍大きくなることが分かった．

4.6.4 オブジェクトサイズ

評価の対象とした，HRP2 カーネルを用いて開発するアプリケーション構成は，カーネルドメインが 1 つ，ユーザドメインが 3 つ存在し，それぞれの保護ドメインには，タスクとメモリオブジェクトがそれぞれ 1 つずつ属しており，そして，無所属のメモリオブジェクトが 1 つ存在するものとした．一方，比較対象とした，ASP カーネルを用いて開発するアプリケーション構成は，HRP2 カーネルを用いて開発するアプリケーションから，保護ドメインとメモリオブジェクトを取り除いたもの，つまり，タスクが 4 つ存在し，かつ，アプリケーションのソースコードが同じものとした．

以上のアプリケーションのオブジェクトモジュールを生成し，それらに含まれる

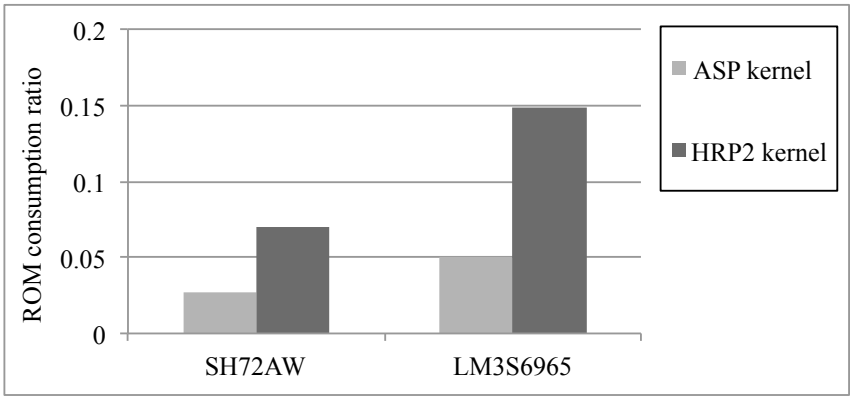

Figure 4.12: ROM 使用量の比較

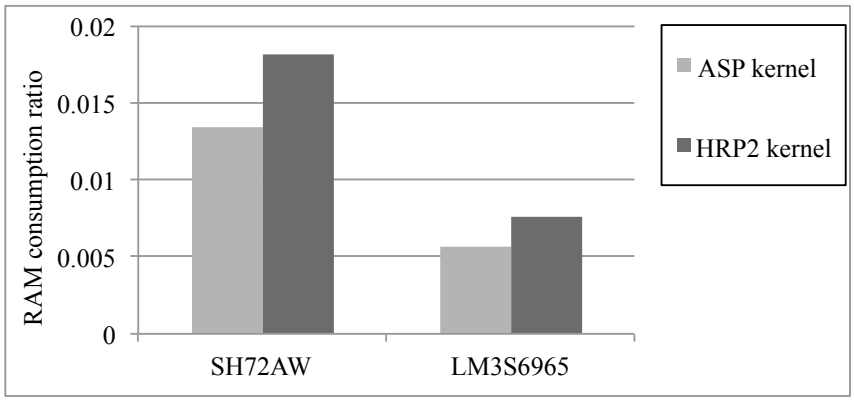

Figure 4.13: RAM 使用量の比較

カーネルのオブジェクトサイズを計測した結果を，ROM 使用量については図 4.12 に，RAM 使用量については図 4.13 にそれぞれ示す．図 4.12，および，図 4.13 では，メモリ使用量を，チップに搭載された ROM，および，RAM のサイズに対する割合で表している．

ROM 使用量をみると，HRP2 カーネルを用いた場合，ASP カーネルを用いた場合と比較して，約 2.5 倍から 3 倍の使用量となっている．この増加量の主な要因は，メモリ保護機能を実現するためのプログラムコードであった．しかし，表 4.5 で示したターゲットプロセッサの ROM サイズと比較すると，ROM 使用量は最大で約 15%程度であり，それほど多くないと考える．

一方，RAM 使用量をみると，HRP2 カーネルを用いた場合，ASP カーネルを用いた場合と比較して，約 150byte 程度の増加量で抑えられている．この増加量の要因を調べたところ，まず，MPU の状態や実行中のタスクが属する保護ドメインを管理するデータのように，システムの実行中に書き換える必要があり，かつ，メモリ保護を実現するために新たに導入したデータがあった．これらの要因による RAM 使用量のオーバヘッドは，メモリ保護を実現する上で避けられないものであるといえる．

ここで，HRP2 カーネルでは，静的コンフィギュレーション時に，一度仮のオブジェクトモジュールを生成した後で，メモリ保護のための情報を生成し，その情報を ROM に配置することで，RAM 使用量を抑えると述べた．この効果を確かめるため，メモリ構成初期化ファイルに出力された情報の ROM 使用量を確認した．メモリ構成初期化ファイルに出力される情報は，4.5.2 節で述べたように，静的に生成しなければ，RAM に配置されてしまう情報である．メモリ構成初期化ファイルに出力された情報の ROM 使用量を確認した結果，そのサイズは，SH72AW で 1,936byte，LM3S6965 で 1,004byte であった．このサイズは，RAM 使用量のオーバヘッドである，150byte を大きく上回るものである．このような結果から，メモリ保護のための情報を，可能な限り静的に生成することで，RAM 使用量のオーバヘッドを抑えることができたといえる．

4.6.5 アクセス権の確認を含むサービスコールの実行時間

SH72AW 上の HRP2 カーネルにおいて，ref_tsk を呼び出したときの実行時間を，(i) ソフトウェアのみでアクセス権の確認をした場合，(ii)MPU の領域サーチ機能を用いてアクセス権の確認をした場合の 2 通りの場合でそれぞれ計測した．ここで，実行時間は，ref_tsk を呼び出してから，呼出し元に戻るまでの時間とした．また，メモリオブジェクトの数については，32 個，64 個，128 個の 3 通りを計測対象とした．

以上のように，ref_tsk の実行時間を計測した結果を，図 4.14 に示す．まず，(i) の場合の計測結果をみると，メモリオブジェクトの数が 2 倍になるにつれて，実行時間

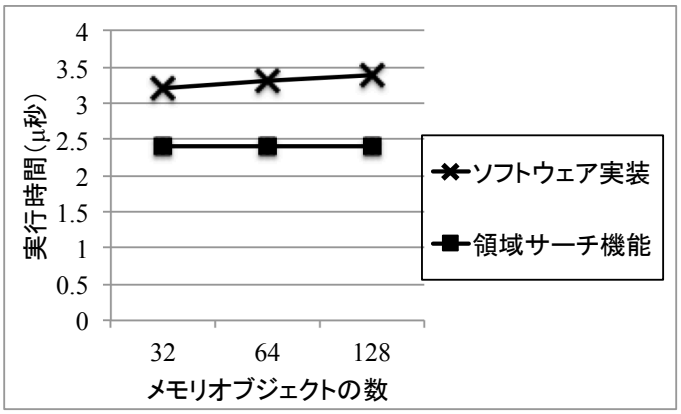

Figure 4.14: ref_tsk の実行時間（μ 秒）

は 0.1μ 秒増加している．この結果から，4.5.6 節で述べたように，N 個のメモリオブジェクトの管理情報を，ソフトウェアでサーチする処理の実行時間が O(logN) に抑えられていることを確認した．そして，(ii) の場合の計測結果をみると，メモリオブジェクトの数に関わらず，一定の時間でアクセス権の確認ができており，さらに，(i) の場合よりも，短時間でアクセス権の確認ができている．この結果から，MPU の領域サーチ機能を用いることで，メモリオブジェクトの数に関わらず一定時間でアクセス権の確認が可能であること，および，ソフトウェアのみでアクセス権の確認を行うよりも，短時間で処理が可能であることを確認した．

4.6.6 MPU を用いたメモリ保護の実現とリアルタイム性の保証

メモリ保護が実現できること，および，リアルタイム性の保証が容易であることは，定量的な評価が困難であるため，定性的に評価（考察）した結果を述べる．

HRP2 カーネルでは，4.5.4 節で述べたように，静的コンフィギュレーション時に，同じアクセス保護属性を持ったセクションが連続して配置されるように，かつ，MPU のアライメント制約に合った番地にメモリ領域が配置されるように，オブジェクトモジュールのセクションを並べること，および，4.5.5 節で述べたように，MPU の操作をカーネル内部で行うことにより，MPU を用いたメモリ保護を実現し，MPU の隠蔽

を実現した．ただし，MPU の領域レジスタ数は隠蔽しておらず，アプリケーション開発者が設定することのできるアクセス保護属性は，MPU の領域レジスタ数に依存する．

HRP2 カーネルでは，MPU を用いてメモリ保護を実現したため，4.2.2 節で述べたように，TLB ミスのようにリアルタイム性を阻害するような事象は発生しない．そして，HRP2 カーネルにおいて，MPU の操作をする箇所は，ループ処理のように，実行時間の予測が困難な処理を用いず，すべて一定の命令数の処理で実現が可能であった．

以上のことから，HRP2 カーネルでは，MPU を用いたメモリ保護を実現しており，リアルタイム性の保証が容易であると考える．

4.7 まとめ

本章では，メモリ保護機能を持ったリアルタイム OS として開発した，HRP2 カーネルについて述べた．HRP2 カーネルは，TOPPERS 新世代カーネル仕様に準拠した，保護機能対応カーネルである．HRP2 カーネルでは，静的コンフィギュレーションにおいて，メモリ配置を静的に行い，また，メモリ保護のための情報を静的に生成するようにした．そして，SH2A と ARM Cortex-M3 の MPU を用いたメモリ保護機能を実現し，アプリケーション開発者が，MPU の制約を意識せずに開発を行えるようにした．また，メモリ保護のための情報を，可能な限り静的に生成することで，RAM 使用量のオーバヘッドを抑えるようにした．HRP2 カーネルの性能評価を行い，実行時間のオーバヘッドを示した．さらに，RAM 使用量のオーバヘッドが抑えられていることを示した．

CHAPTER 5

メモリ保護を考慮したコンポーネント技術

5.1 概要

本章では，TECS に基づいて開発した，高信頼性が求められる組込みシステム向けのコンポーネント技術，HR-TECS について述べる．本論文では，高信頼のための機能のうち，メモリ保護機能の実現について述べる．メモリ保護以外の，時間保護や機能安全，セキュリティなどの機能の実現については今後の課題である．本章では，まず，HR-TECS の要件を定義し，HR-TECS の仕様を設計する．そして，HRP2 カーネルを基盤として，HR-TECS の実装方法を示したうえで，実行時間とメモリ使用量について評価した結果を述べる．

本章の研究内容による貢献は，以下のとおりである．

(3-1) メモリ保護を考慮したコンポーネント記述方法を提案する．

(3-2) コンポーネントの所属するパーティションに依存しない，コンポーネント間通信を実現する方法を示す．

(3-3) 提案するコンポーネント技術の実現方法を示し，メモリ使用量と実行時間のオーバヘッドを明らかにする．

HR-TECS では，メモリ保護を考慮したコンポーネント記述が可能であり，このコンポーネント記述から，リアルタイム OS の設定ファイルを自動生成する．コンポーネント間の通信について，通信処理本体をコンポーネント記述から自動生成することで，個々のコンポーネントにおける通信 API の記述は，配置されるパーティション（保護ドメイン）に依存せず，共通の API で開発することができる．また，HR-TECS は，非特権モードで動作するアプリケーションだけでなく，特権モードで動作するミドルウェアやデバイスドライバなどのソフトウェアプラットフォームのコンポーネントベース開発もできるようにする．ここで，HR-TECS は，TECS に基づいており，TECS は，第 3 章で示したようにプラットフォーム開発に有用であることから，HR-TECS で開発したプラットフォームは，ハードウェア構成やターゲットアプリケーションの要求に対して，柔軟にプラットフォームの構成を変更，カスタマイズできると考えられる．

本章の構成は次のとおりである．まず，HR-TECS の対象システムと要件を定義し，要件を満たすための開発方針について述べる．次に，HR-TECS の仕様と，HRP2 カーネルを用いた実装方法について述べ，HR-TECS を用いて開発したマイクロベンチマークソフトにおける，実行時間とメモリ使用量のオーバヘッドを評価する．

5.2 メモリ保護を考慮したコンポーネント技術

メモリ保護を考慮した組込みシステムのソフトウェア開発を支援するコンポーネント技術 HR-TECS を提案するにあたり，HR-TECS が対象とするシステムの前提条件，および，HR-TECS の要件を定義する．

5.2.1 対象とするシステム

HR-TECS は，単一プロセッサ上で動作し，メモリ領域のパーティショニングが必要なソフトウェアを対象とした，ソフトウェアのコンポーネントベース開発を支援するためのソフトウェアプラットフォームである．HR-TECS では，メモリ領域のパーティショニングを実現するために，メモリ保護用のハードウェアである，MMU や MPU を搭載したプロセッサを用い，そして，MMU や MPU を抽象化するリアルタイム OS の機能を用いる．

また，HR-TECS では，使用されるコンポーネントやその結合関係，および，使用されるリアルタイム OS のカーネルオブジェクト（タスクやセマフォなど）のソフトウェアの構成要素が，すべて静的に決定されるものを対象とする．つまり，HR-TECS では，システムの実行中に，コンポーネントの生成や削除，結合関係の変更などを行わないソフトウェア開発を対象とする．ここで，HR-TECS の仕様は，上記の性質を満たすものであれば，特定のリアルタイム OS に依存しないものとする．

5.2.2 HR-TECS の要件

HR-TECS を提案するにあたり，その要件を以下のように定義する．

要件 1. ソフトウェアをコンポーネント単位でいくつかのパーティションに，静的に分割できること

要件 2. 特権モードで動作するコンポーネントが属するパーティションと，非特権モードで動作するコンポーネントが属するパーティションが存在できること

要件 3. コンポーネントの処理を実行する主体となる処理単位（タスクなど）を指定でき，その処理単位が所属するパーティションを指定できること

要件 4. 個々のコンポーネントが，異なるパーティションに配置されたコンポーネント固有のメモリ領域にアクセスできないこと

要件 5. 個々のコンポーネントを開発するにあたり，コンポーネント間の通信を，コンポーネントが配置されるパーティションに依存せず，共通の API を使用して開発できること

要件 6. RAM の使用量が小さいこと

要件 7. コンポーネント間通信の実行時間が小さく，かつ，実行時間の予測が可能であること

要件 1，4，および，5 に関しては，既存のメモリ保護を考慮したコンポーネント技術と同様に，メモリ保護を考慮したソフトウェアのコンポーネントベース開発を支援するために必要であると考え，定めた．

要件 2 に関しては，組込みシステムにおいては，デバイスドライバやミドルウェアを含むソフトウェアプラットフォーム部分をカスタマイズする場合があり，プラットフォームを含めたコンポーネント開発を可能とするために必要であると考え，定めた．デバイスドライバなどのソフトウェアは，特権モードでなければアクセスすることができない場合がある．

要件 3 に関して，組込みリアルタイムシステムにおいては，ソフトウェアの処理単位が複数存在し，処理単位を切り替えながら実行を進める，マルチタスクとなっていることが多く，さらに，それらの処理単位が，定められた時刻までに処理を終えることができるかどうかを解析することが重要である．よって，ソフトウェア開発をするうえで，ソフトウェアを処理する処理単位，および，その処理単位が実行するコンポーネントが明確になっている方が適切であると考え，この要件を定めた．また，処理単位と実行されるコンポーネントの関係を明確化することにより，処理単位をリアルタイム OS で構成するための設定情報も自動生成することができ，ソフトウェア開発効率を向上できる可能性がある．

要件 6 に関しては，組込みシステムで用いられるハードウェアは，RAM 領域のサイズが小さいことが多いため，RAM 使用量の増加を抑えるようにする必要があると

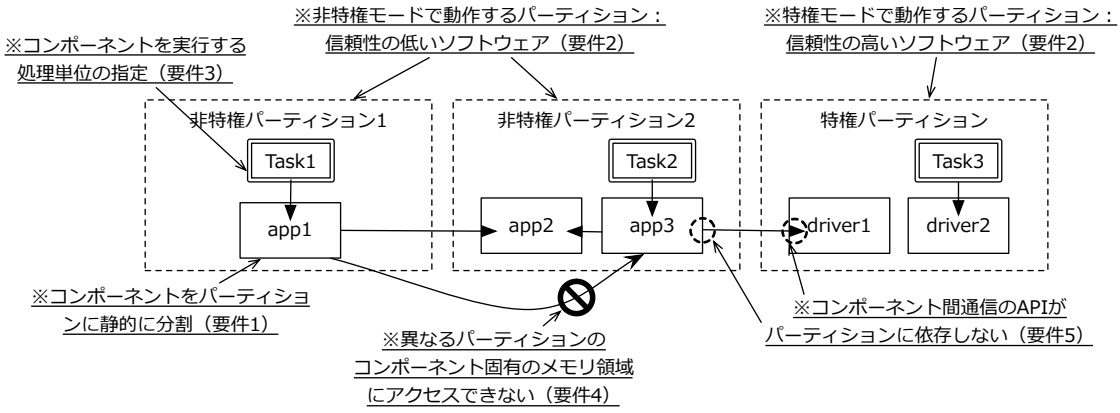

Figure 5.1: HR-TECS の要件

Table 5.1: HR-TECS の要件対応

	要件 1	要件 2	要件 3	要件 4	要件 5	要件 6	要件 7
TECS	×	×	△	×	△	○	○
HR-TECS	○	○	○	○	○	○	○

考え，定めた．

要件 7 に関しては，組込みシステムでは，実行時間のオーバヘッドが小さいことが求められ，かつ，実行時間が予測でき，リアルタイム性を保証できることが重要であるため，定めた．

HR-TECS の要件を表すイメージ図を，図 5.1 に示す．

5.2.3 HR-TECS の開発方針

HR-TECS は，組込みシステム向けのソフトウェアコンポーネント技術である，TECS に基いて開発することとした．TECS で解決できる要件と，HR-TECS で解決する要件について，表 5.1 にまとめる．

TECS では，静的なコンポーネントモデルを採用しており，コンポーネントのインスタンス化や結合にかかる実行時オーバヘッドをなくすことができる．さらに，コンポーネント結合の最適化を行うことで RAM の使用量や実行時オーバヘッド削減で

きる [9]．これにより要件 6，7 を満たすことができると考えた．

要件 1 に関して，TECS では，静的なコンポーネントモデルを採用しているため，個々のコンポーネントをパーティションに静的に配置することは容易にできると考えた．しかしながら，従来の TECS 仕様では，個々のコンポーネントをパーティションに配置することは考慮されていないため，仕様を拡張する必要がある．このとき，要件 2 を満たすことができるように，コンポーネントを配置するパーティションに対して，特権モードで動作するか，非特権モードで動作するかの情報を指定できるようにする．

要件 3 に関しては，処理単位をコンポーネントとして扱うことによって実現する．これは，文献 [10] で述べられている，リアルタイム OS のリソースをコンポーネントとして扱う方法と同様の方法である．まず，処理単位に相当するセルタイプを定義し，HR-TECS から提供する．そして，アプリケーション開発者が，そのセルタイプから，処理単位のセルを生成し，処理単位から呼ばれるべきセルの受け口と結合することによって，コンポーネントの処理を実行する処理単位を明確化できるようにする．さらに，処理単位のセルタイプには celltype プラグインを指定し，生成された処理単位のセルに応じて，リアルタイム OS が処理単位を構成するために必要な情報を出力させ，リアルタイム OS によってコンポーネントの処理の実行を制御させる．ここで，文献 [10] の方法では，処理単位が所属するパーティションを指定できないが，HR-TECS では，その指定を可能とし，さらに，celltype プラグインにより，リアルタイム OS で処理単位が動作するパーティション（ドメイン）を指定するための情報を出力させる．

要件 4 に関しては，既存のコンポーネント技術と同様に，メモリ保護機能を持ったリアルタイム OS を用いる．ここで，コンポーネントの設定記述から，そのリアルタイム OS におけるメモリ保護の設定ファイルを自動生成するようにする．TECS においては，そのコンポーネント記述から，celltype プラグインを用いて独自の出力をすることができるため，この celltype プラグインにより，メモリ保護の設定を出力する．しかしながら，すべてのセルタイプに対して，メモリ保護の設定を出力させる

celltype プラグインを指定することは非効率的であり，アプリケーションソフトウェアのセルタイプを定義する開発者にとって不便であると考える．そのため，本論文における実装では，処理単位のセルタイプに対する celltype プラグインによって，すべてのセルタイプとセルの情報を読込み，メモリ保護のためのリアルタイム OS の設定情報を出力することとした．

要件 5 に関して，TECS では，個々のコンポーネントにおいて，自身のセルの持つ呼び口を介して，結合先の受け口の関数を呼び出すため，あるコンポーネントにおいて，別のコンポーネントの処理（受け口関数）を呼び出す処理を実装するとき，結合先のコンポーネントに依存せず，共通の API（呼び口関数）で実装できる．さらに，文献 [8] では，結合されたコンポーネント間に，コンポーネントが配置された環境に応じた通信処理（Remote Procedure Call: RPC）を行うコンポーネントを，through プラグインで挿入することにより，共通の呼び口関数で，コンポーネントの配置される環境に依存しない通信を実現する手法を述べている．この手法を応用することで，コンポーネントの配置されたパーティションに依存せず，個々のコンポーネントを開発できるようにする．

また，これまでに我々は，ソフトウェアプラットフォームを含めて TECS を用いて開発した実績があり [71]，要件 2 に関して，ソフトウェアプラットフォームを含めてコンポーネントベース開発できるようにするために，TECS が適していると考えた．

5.3 HR-TECS の仕様設計

5.3.1 HR-TECS の概要

HR-TECS の構成を図 5.2 に示す．HR-TECS は，MMU や MPU といったメモリ保護専用ハードウェアを搭載したプロセッサ上で動作するソフトウェアを対象とし，メモリ保護機能を持ったリアルタイム OS の機能を利用して，アプリケーションソフト

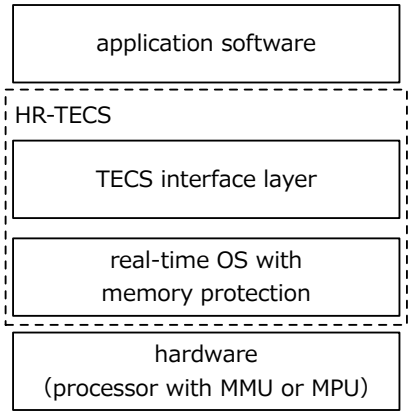

Figure 5.2: HR-TECS の構成

ウェアのメモリ保護を実現する．TECS インタフェース層は，セルの定数や変数，および，セル間の結合を実現するためのソフトウェア部分であり，TECS ジェネレータによって自動生成される．

次に，HR-TECS を用いたアプリケーション開発の流れを図 5.3 に示す．まず，アプリケーションソフトウェアのインタフェースとコンポーネントを定義するために，シグニチャ記述とセルタイプ記述を作成する．(図 5.3 の手順 1)．そして，定義したコンポーネントを生成，および，結合してソフトウェアを構築するために，組上げ記述を作成する．(図 5.3 の手順 2)．ここで，組上げ記述において，セルの所属するパーティションを設定する．これらのコンポーネント記述を TECS ジェネレータに入力することで，コンポーネントの生成と結合部分（インタフェースコード）とコンポーネントのソースコードのテンプレートが生成される．ここで生成されるテンプレートを編集することによって，アプリケーションのソースコードを作成する（図 5.3 の手順 3）．また，TECS ジェネレータは，リアルタイム OS において，処理単位の生成や，メモリ保護の設定をするための設定ファイルを生成する．この設定ファイルから，リアルタイム OS のソースコードが，リアルタイム OS の持つ設定ツールにより生成される．最後に，アプリケーションのソースコードとインタフェースコード，リアルタイム OS のソースコードをコンパイル，リンクすることで，対象ハードウェア上で実

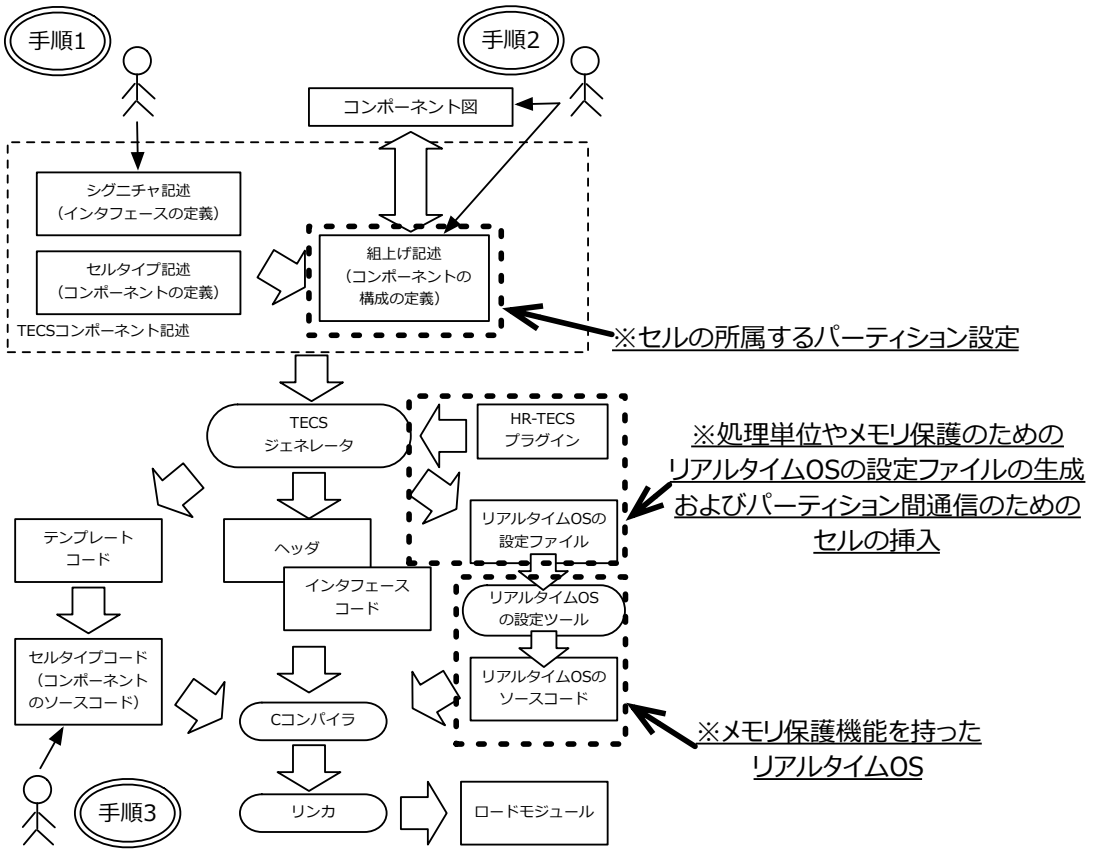

Figure 5.3: HR-TECS を用いた開発の流れ

行可能なロードモジュールを生成する．

5.3.2 コンポーネントのパーティショニング

HR-TECS では，個々のコンポーネントをパーティションに配置できるようにする（要件 1）．従来の TECS では，コンポーネントをパーティションに配置することはできないため，コンポーネント記述の仕様を拡張する．

まず，パーティションに相当する概念を TECS に導入する．TECS において，コンポーネントのインスタンスであるセルを配置するためのパーティションをリージョンと呼ぶ．図 5.4 は，リージョンにセルを配置するための TECS コンポーネント記述の例である．2 行目のように，region キーワードに続けてリージョン名（rKernelDomain,

```
 1: [domain("KERNEL_DOMAIN","trusted")]
 2: region rKernelDomain{
 3:   cell tTrustedCallee TrustedCallee{
 4:   };
 5: };
 6: [domain("DOMAIN_B","nontrusted")]
 7: region rB{
 8:   cell tCallee Callee{
 9:     cCall =
        rKernelDomain::TrustedCallee.eEntry;
10:   };
11: };
12: [domain("DOMAIN_A","nontrusted")]
13: region rA{
14:   cell tCaller Caller{
15:     cCall = rB::Callee.eEntry;
16:   };
17: };
```

Figure 5.4: リージョン記述

rB，rA）を記述し，それに続く括弧内に，そのリージョンに配置するセルの組上げ記述を列挙する．ここで，リージョンに対しては，1 行目のように指定子を付け，リージョンの使用目的を区別する[1]．domain は，そのリージョンが，メモリ保護のためのパーティションであることを示す指定子である．domain 指定子には，引数として，ID（KERNEL_DOMAIN，DOMAIN_B，DOMAIN_A）と，そのリージョンに属するコンポーネントが特権モードで動作するかどうか（trusted，nontrusted）を指定する．domain 指定子の引数は，リアルタイム OS におけるメモリ保護の設定情報を生成するために必要となり，また，要件 2 を満たすために必要となる．セルとセルを結合するための記述は，それらのセルが同じリージョンに所属する場合には，従来の TECS と同様の記述である．一方，互いに異なるリージョンに所属する 2 つのセルを結合するときには，9 行目のように，結合先のセル名の前に，結合先のセルが所属するリージョン名を，::キーワードで連接して記述する．

[1] メモリ保護のパーティション以外の，セルをグループ化させる目的で使えることを考慮した．

5.3.3 コンポーネントの処理単位

コンポーネントの処理を実行する処理単位を指定する（要件 3）ために，処理単位をコンポーネントとして扱う方法を用いる [10]．処理単位のセルタイプを HR-TECS で提供し，処理単位のセルと，その処理単位から呼び出されるセルを結合することで，アプリケーション開発者が，処理単位から実行されるセルを指定できるようにする．さらに，処理単位のセルを生成するコンポーネント記述から，リアルタイム OS において，処理単位を生成するための設定情報（静的 API の呼出し）を，TECS ジェネレータにより自動生成する．ここで，リアルタイム OS のための設定情報は，celltype プラグインによって生成し，処理単位のセルタイプに対して，この celltype プラグインを指定する．

5.3.4 コンポーネントのメモリ保護

コンポーネントのメモリ保護（要件 4）を実現するためには，コンポーネントがどのようなプログラム（コード，データ）で構成されているかを考慮し，そのプログラムに対してメモリ保護の設定を行う必要がある．従来の TECS において，コンポーネントを実現するために扱われる C 言語プログラムには，大きく分けて，セル固有のデータと，同じセルタイプから生成されたセルで共有されるコードとデータの 2 種類が含まれる．セル固有のデータには，セルの定数や結合情報を持つセル初期化ブロック，セルの変数を持つセル管理ブロック，セルの受け口に関する情報を持つ受け口ディスクリプタがある．また，同じセルタイプから生成されたセルで共有されるコードとデータには，セルタイプの受け口関数，受け口関数を呼び出すための受け口スケルトン関数，受け口スケルトン関数の関数テーブルがある．これらのコード，データについての詳細は，文献 [65] で述べられているが，ここでは省略する．

セル固有のデータについては，要件 4 を満たすために，対象のセルが所属するリージョンからのみアクセスできるよう，メモリ保護の設定をする．一方，同じセルタイ

プから生成されたセルで共有されるコードとデータについては，次のように扱うこととする．まず，あるセルタイプから生成されたセルが，すべて同じリージョンに所属する場合，これらのセルで共有されるコードとデータは，その単一のリージョンからのみアクセスできるように設定すればよいと考えられる．次に，あるセルタイプから生成されたセルが，すべて同じリージョンに所属していない場合，これらのセルで共有されるコードとデータは，複数のリージョンからアクセスできるように設定する必要がある．ある特定の複数のリージョンからアクセスできるように設定できるターゲットであれば，そのように設定すればよいと考えられるが，メモリ保護ユニットを用いる場合のように，設定できるアクセス権の種類が限られており，そのような設定ができない場合には，すべてのリージョンからアクセスできるように設定する（セル固有のコードやデータではないため，要件 4 には違反しない）．もし，設定できるアクセス権の種類が限られているターゲットで，セルタイプコードへのアクセスを制限したい場合には，セルタイプコードファイルを必要なリージョンの数だけ複製して，個別にメモリ保護の設定をする方法が考えられるが，本論文では，考慮の対象外とする．

5.3.5 コンポーネント間通信の抽象化

HR-TECS では，メモリ保護機能により，互いに異なるリージョンに所属する 2 つのセル間で，結合先の受け口関数を直接呼び出すことは制限される．これらのセル間での通信を，要件 5 を満たし，従来の TECS におけるセル間通信の記述で実現するために，TECS の through プラグインを利用する．through プラグインにより，結合されたセルとセルの間に，異なるパーティション間での通信を行うためのセルを挿入することで，コンポーネント開発者は，セルの配置されるリージョンに依存せず，共通の呼び口関数で，結合先セルの受け口関数を呼び出す機能を実装できる．

5.4 HR-TECS の実装

本節では，実際のメモリ保護機能を持ったリアルタイム OS を用いて，HR-TECS を実現するための方法について述べる．本論文では，メモリ保護機能を持ったリアルタイム OS として，HRP2 カーネルを用いる．

5.4.1 環境

本論文では，HRP2 カーネルのメモリ保護機能により，HR-TECS を実現する．また，HR-TECS の実装，および，評価をするための対象ハードウェアとして，ARM Cortex-M3 コアを持つプロセッサ，LM3S6965[56] を使用する．LM3S6965 プロセッサは，ARM Cortex-M3 の仕様で定義される MPU を持ち，HRP2 カーネルのメモリ保護機能を実現できる．また，開発環境には，GNU のツールチェーンを用いる．

5.4.2 処理単位とメモリ保護の設定

5.3.3 節，および，5.3.4 節で述べた，コンポーネントを実行する処理単位，および，コンポーネントのメモリ保護は，HRP2 カーネルの提供するタスクと，メモリ保護機能によって実現する．そのために，HR-TECS におけるコンポーネント記述から，HRP2 カーネルの静的 API の呼出し（コンフィギュレーションファイル）を，TECS ジェネレータによって自動生成し，HRP2 カーネルにおけるタスク生成や，メモリ保護の設定を行う．

まず，HR-TECS における，domain 指定されたリージョンを，HRP2 カーネルにおける保護ドメインに対応させる．そして，HR-TECS において，“trusted”，“nontrusted” が指定されたリージョンは，それぞれ，HRP2 カーネルにおける，カーネルドメイン，ユーザドメインに対応させる．

次に，HR-TECS における処理単位（タスク）のセルの生成記述から，HRP2 カーネルにおけるタスクを生成するための静的 API（CRE_TSK）を，TECS ジェネレータ

により自動生成する．これらの静的 API は，生成対象となるタスクセルが所属するリージョンに対応した保護ドメインの括弧内に出力する．ここで，静的 API の出力は，タスクのセルタイプに指定した，celltype プラグインによって行い，この celltype プラグインに，静的 API を出力する処理を実装する．

そして，コンポーネントのメモリ保護設定は，HRP2 カーネルにおける，ATT_MOD（静的 API 関数）によって行う．ATT_MOD により，セル固有のセル初期化ブロック，および，受け口ディスクリプタが，そのセルが所属するリージョンに対応する保護ドメインから読出し，および，実行できるように設定し，セル管理ブロックが，そのセルが所属するリージョンに対応する保護ドメインから読出し，書込み，および，実行できるように設定する．また，同じセルタイプから生成されたセルで共有される，受け口関数，受け口スケルトン関数，および，受け口スケルトン関数テーブルについては，そのセルタイプから生成されたセルが，すべて同じリージョンに所属する場合には，そのリージョンに対応する保護ドメインから読出し，および，実行できるように設定し，そうでない場合には，すべての保護ドメインから読出し，および，実行できるように設定する．ATT_MOD は，タスクのセルタイプに指定した，celltype プラグインの内部処理で出力する．

5.4.3 コンポーネント間通信

5.3.5 節で述べた，コンポーネント間通信の抽象化を実現するために，異なるリージョンに所属する 2 つのセル間で通信が必要な場合には，through プラグインにより，リージョンを越えた通信処理を行うセルを挿入する．

本実装では，HR-TECS におけるセル間の通信としては，(a) 同じリージョンに属するセル間の通信，(b) あるリージョンに所属するセルから，特権モードで動作するリージョンに所属するセルへの通信，(c) あるリージョンに所属するセルから，非特権モードで動作するリージョンに所属するセルへの通信，の 3 種類がある．ここで，(a) については，従来の TECS におけるセル間通信で実現できる．

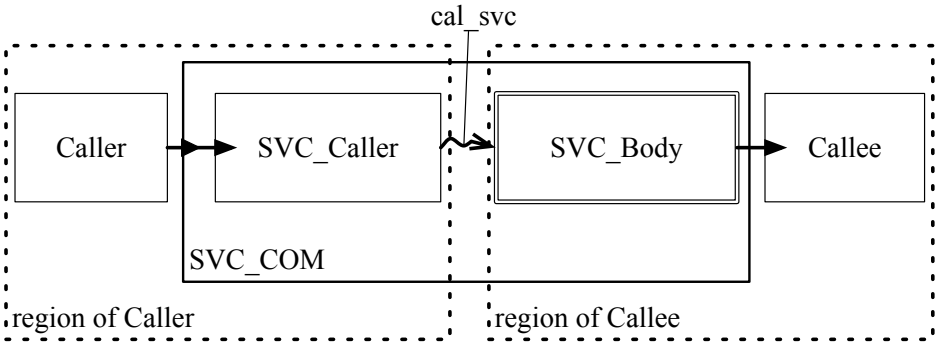

Figure 5.5: SVC_COM コンポーネントの構成

(b) の通信は，HRP2 カーネルにおける拡張サービスコール機能を利用して実現する．拡張サービスコール機能を利用して通信を行うセル（SVC_COM）を，図 5.5 に示す．図 5.5 は，セル Caller から，セル Callee の受け口関数を呼び出す場合を表している．まず，Caller は，自身の呼び口関数により，Callee の受け口関数を呼びだそうとする．このとき，この呼び口関数は，through プラグインにより，SVC_Caller セルの受け口関数に割り当てられている．SVC_Caller は，対応する Callee の受け口関数を呼び出す主体となる，SVC_Body セルの main 関数を，cal_svc によって呼び出す．SVC_Body セルの main 関数は，拡張サービスコールであり，through プラグインによって，静的 API，DEF_SVC を生成し，登録される．ここで，main 関数とは，受け口に含まれる関数ではなく，リアルタイム OS の機能などにより呼び出される関数であり，他のセルからは直接呼び出さない（タスクの main 関数などと同様）．SVC_Body の呼び口は，through プラグインによって，Callee の受け口に結合されており，SVC_Body の main 関数から，Callee の受け口関数を呼び出す．そして，Callee の受け口関数を実行した結果の返り値は，cal_svc の返り値として SVC_Caller に渡され，さらにその返り値として，Caller に渡される．ここで，SVC_Caller は，Caller と同じリージョンに所属させ，SVC_Body は，Callee と同じリージョン（カーネルドメインに対応するリージョン）に所属させる．

(c) の通信は，HRP2 カーネルではその通信機能を提供していないため，遠隔手続き

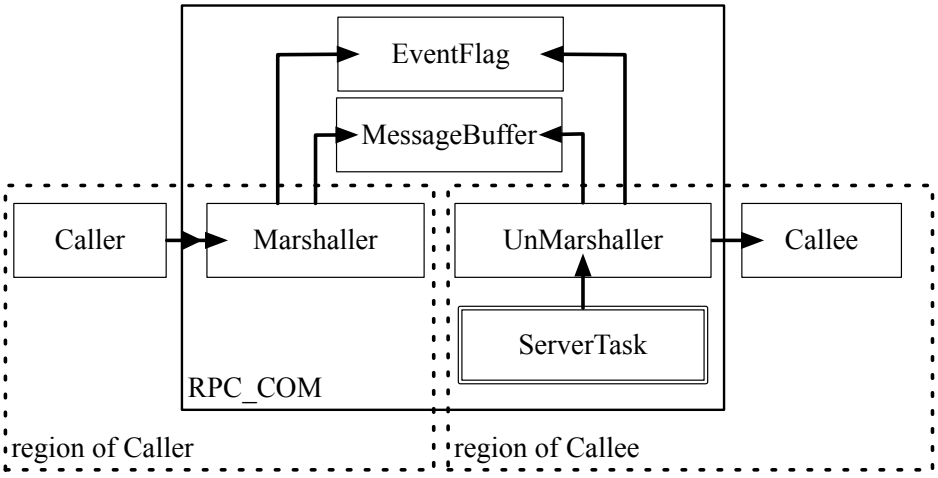

Figure 5.6: RPC_COM コンポーネントの構成

呼出し（RPC）によって実現する．遠隔手続き呼出しにより通信を行うセル（RPC_COM）を，図 5.6 に示す．図 5.6 は，図 5.5 と同様に，セル Caller から，セル Callee の受け口関数を呼び出す場合を表している．まず，Caller は，自身の呼び口関数により，Callee の受け口関数を呼びだそうとする．このとき，この呼び口関数は，through プラグインにより，Marshaller セルの受け口関数に割り当てられている．Marshaller は，呼び出す受け口関数の ID と引数を 1 つのメッセージにまとめ，MessageBuffer セルに送信した後，EventFlag セルに対してイベントが通知されるのを待つ．ここで，MessageBuffer は，データ通信用のセルであり，μITRON4.0/PX 仕様 [61] で定義されるメッセージバッファ機能を用いて実現する．メッセージバッファは，可変長のメッセージを送受信するためのカーネルオブジェクトであり，メッセージバッファ機能を HRP2 カーネルに実装して利用した．また，EventFlag は，イベント通知用のセルであり，HRP2 カーネルの提供するイベントフラグ機能を用いて実現する．イベントフラグは，ある特定の値を介して，イベント通知を行うためのカーネルオブジェクトであり，ここでは，Callee の受け口関数の実行終了の通知を受け取るために用いる．次に，ServerTask セルは，Callee の受け口関数を呼び出す主体のタスクセルであり，リアルタイム OS の機能によって呼び出される main 関数で，UnMarshaller セルを呼び出す．UnMarshaller

は，MessageBuffer にメッセージが送信されるのを待ち，送信されたメッセージを受信する．そして，受信した関数 ID と引数の情報から，対応する Callee の受け口関数を呼び出す．そして，Callee の受け口関数を実行した結果の返り値は，EventFlag に設定され，受け口関数の終了通知とともに，Marshaller に渡された後，さらにその返り値として，Caller に渡される．ここで，Marshaller は，Caller と同じリージョンに所属させ，ServerTask と UnMarshaller は，Callee と同じリージョンに所属させる．MessageBuffer と EventFlag については，リージョンには所属させず，対応するカーネルオブジェクトに対し，Marshaller と UnMarshaller から適切なアクセスができるように設定する．また，RPC_COM で使用するメッセージバッファ，イベントフラグ，および，タスクを生成するための静的 API は，through プラグインによって自動生成する．

5.5 評価実験

HR-TECS におけるメモリ使用量と，コンポーネント間通信にかかる実行時間について評価した結果について述べる．本評価実験では，5.4.1 節で述べた環境を用い，(A) 同じリージョンに所属するセル間通信，(B) 非特権モードで動作するリージョンに所属するセルから特権モードで動作するリージョンに所属するセルへの通信，および，(C) 非特権モードで動作するリージョンに所属するセルから非特権モードで動作する別のリージョンに所属するセルへの通信，の 3 種類の処理を行うソフトウェアを，HR-TECS を用いてそれぞれ開発し，評価を行った（図 5.7）．(A) の通信は，通常の TECS と同じ通信処理（呼び口から受け口関数の関数を呼び出す処理）で実現される．(B) の通信においては SVC_COM セルを，(C) の通信においては RPC_COM セルを，それぞれ使用した．また，評価対象のコンポーネント間通信の処理として，32 ビットの引数 5 つを渡す，関数呼出しを実行する．

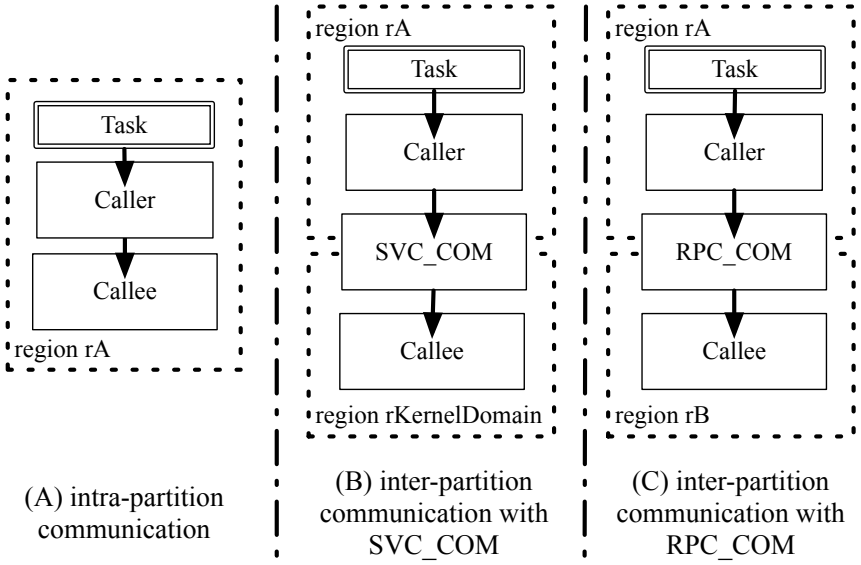

Figure 5.7: 評価アプリケーションのコンポーネント図

5.5.1 コンポーネント記述

図 5.8 に，本評価実験で作成したソフトウェアの，TECS コンポーネント記述（組上げ記述）を示す．ここで，図 5.8 では，Caller セルは，各リージョンに属するすべての Callee セルに結合されているが，実際には，(A) の評価実験では，リージョン rA に所属する Callee のみ，(B) の評価実験では，リージョン rKernelDomain に所属する Callee のみ，(C) の評価実験では，リージョン rB に所属する Callee のみを，それぞれインスタンス化し，Caller の呼び口 cCall と，Callee の受け口 eEntry とを結合している．16 行目は，(A) の場合におけるセルの結合をするための記述であり，従来の TECS と同様に記述する．18-19 行目は，(B) の場合における，rA リージョンに所属するセルから，rA とは別の rKernelDomain リージョンに所属するセルに対して結合するための記述であり，through プラグインによって，SVC_COM セルを間に挿入することを表している．21-22 行目は，(C) の場合における，rA リージョンに所属するセルから，rA とは別の rB リージョンに所属するセルに対して結合するための記述であり，through プラグインによって，RPC_COM セルを間に挿入することを表している．

```
 1: [domain("KERNEL_DOMAIN","trusted")]
 2: region rKernelDoamin{
 3:   cell tCallee Callee{
 4:   };
 5: };
 6: [domain("DOMAIN_B","nontrusted")]
 6: region rB{
 7:   cell tCallee Callee{
 8:   };
 9: };
10: [domain("DOMAIN_A","nontrusted")]
11: region rA{
12:   cell tCallee Callee{
13:   };
14:   cell tCaller Caller{
15:     /*(A)*/
16:     cCall = Callee.eEntry;
17:     /*(B)*/
18:     [through(HRP2SvcPlugin,"")]
19:     cCall =
          rKernelDoamin::Callee.eEntry;
20:     /*(C)*/
21:     [through(HRP2RpcPlugin,"")]
22:     cCall = rB::Callee.eEntry;
23:   };
24:   cell tTask Task{
25:     cBody = Caller.eBody;
26:     /*omit the task attributes*/
27:   };
28: };
```

Figure 5.8: 評価アプリケーションの組上げ記述

本評価実験では，HR-TECS におけるコンポーネント記述から，TECS ジェネレータにより，HRP2 カーネルの静的 API を自動生成する．ここで生成される静的 API は，タスクなどの処理単位を生成するもの，セルのメモリ保護を設定するもの，および，セル間の通信処理に必要なカーネルオブジェクト（拡張サービスコールやメッセージバッファなど）を生成するものである．

また，本評価実験では，Caller と Callee について，同じセルタイプコード（受け口関数）を，(A)，(B)，(C) のすべての場合で使用することができた．これは，セル間

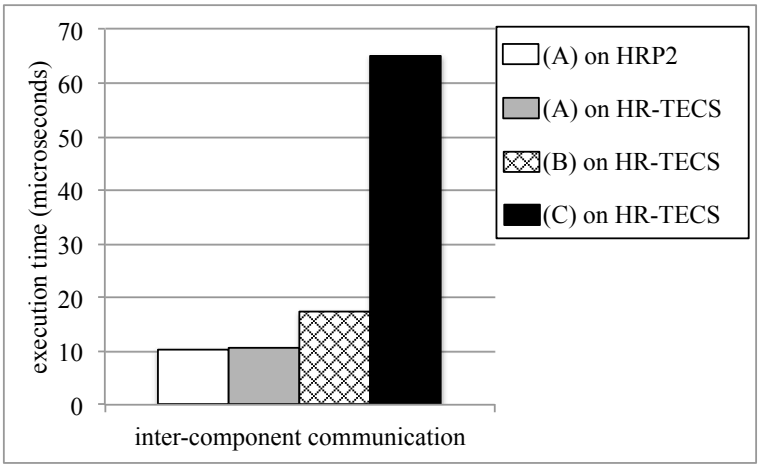

Figure 5.9: コンポーネント間通信の実行時間

の通信が呼び口関数によって抽象化され，その通信処理が，TECS ジェネレータ，および，through プラグインにより自動生成されたためであり，HR-TECS において，コンポーネントの再利用性があることを示している．

5.5.2 実行時間

図 5.9 は，セル間における通信時間，つまり，Caller セルから Callee セルの受け口関数関数を呼出し，呼び口関数の呼出し元に戻るまでの実行時間を計測した結果を，HRP2 カーネルにおいて通常の関数呼出しを行なってから呼出し元に戻るまでの実行時間と比較した結果を示している．図 5.9 で示される実行時間は，セル間の通信時間を，10,000 回計測した内の最大実行時間である．なお，実行時間の計測には，0.1μ 秒単位で時間を計測できるタイマを用いた．また，図 5.9 のグラフは，左から順に，通常の関数呼出し，(A) の場合のセル間通信（通常の TECS におけるセル間通信と同じ処理），(B) の場合のセル間通信，(C) の場合のセル間通信にかかる実行時間を，それぞれ示している．

ここで，(A)，(B)，(C) のそれぞれの場合において，計測された実行時間の分布（最小実行時間と最大実行時間の差）は，すべて，0.2μ 秒以内に収まっていた．この結

果は，セル間の通信にかかる実行時間が一定であり，組込みリアルタイムシステムにとって重要な性質である，実行時間の予測性があることを示している．

(A)，および，(B) の場合，セル間通信の実行時間は，通常の関数呼出しと比較して，それぞれ，0.1μ 秒，7.0μ 秒増加した．この結果から，同じリージョン内，また，特権モードで動作するリージョンへのセル間通信では，HR-TECS で通信を抽象化したことによる実行時間のオーバヘッドは十分小さいと考える．

一方，(C) の場合，セル間通信の実行時間は，通常の関数呼出しと比較して，54.7μ 秒増加した．この実行時間の増加は，メッセージバッファやイベントフラグを用いて，データ通信する機能を呼び出すためのサービスコール呼出し，関数 ID や引数，返り値の送受信，タスク切替えにかかるオーバヘッドが原因であると考えられる．また，この実行時間は，RPC によって渡される引数の数によって，メッセージバッファを介して引数を送受信する時間が変わるため，必ずしも上記の実行時間にならないと考えられる．しかしながら，本評価実験から，引数の数が一定であれば，RPC による実行時間は一定になると考えられるため，実行時間の予測可能性は低下しないと考える．

5.5.3 メモリ使用量

図 5.10 は，HR-TECS を用いて，(A)，(B)，(C) におけるソフトウェアを開発した場合のメモリ使用量を，HRP2 カーネルのみを用いて，(A) に相当するソフトウェアを開発した場合のメモリ使用量と比較した結果を示している．図 5.10 では，FLASH と SRAM のそれぞれのメモリ領域に対する使用量を示しており，図中のグラフは，左から順に，HRP2 カーネルのみを用いて通常の関数呼出しを行う場合，(A) の場合，(B) の場合，(C) の場合におけるメモリ使用量を，それぞれ示している．

(A)，(B) の場合，HR-TECS を用いたときのメモリ使用量は，HRP2 カーネルのみを用いたときと比較して，FLASH 領域においては，それぞれ，92byte と 240byte 増加し，また，SRAM 領域においては，(A)，(B) ともに，16byte 増加した．この結果から，同じリージョン内のみで動作するソフトウェア，また，特権モードで動作する

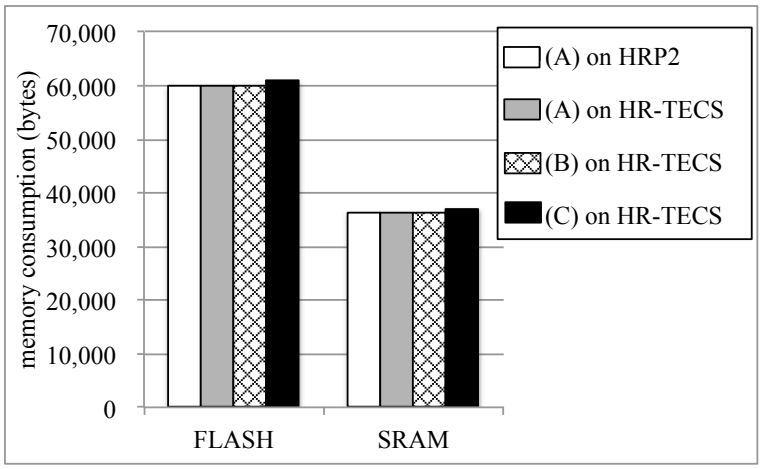

Figure 5.10: メモリ使用量の比較

リージョンへのセル間通信が含まれるソフトウェアでは，HR-TECS を用いたことによるメモリ使用量のオーバヘッドは十分小さいと考える．

一方，(C) の場合におけるメモリ使用量は，(A)，(B) の場合と比較して，FLASH 使用量は約 800byte，SRAM 使用量は約 600byte 増加した．ここで，この SRAM 使用量には，RPC_COM セルに含まれる ServerTask のスタック領域として 384byte，および，MessageBuffer のメッセージ送受信領域として 24byte が含まれている．ただし，タスクのスタックやメッセージバッファ領域に必要なメモリ使用量は，ターゲットアプリケーションによって異なるため，必ずしも上記の値となるということではない．よって，RPC_COM によってセル間通信を抽象化することによるメモリ使用量のオーバヘッドは，ServerTask や Messagebuffer で多くのメモリ領域を使用しなければ，十分に小さい値であると考える．

5.6 まとめ

本章では，メモリ保護を考慮した組込みシステム向けのコンポーネント技術，HR-TECS について述べた．HR-TECS では，TECS 仕様を拡張し，コンポーネント単位で

のソフトウェアのメモリ領域に対するパーティショニング設定を可能とし，そして，メモリ保護機能を持ったリアルタイム OS の機能により，コンポーネントのメモリ領域に対するアクセスを制限する．また，HR-TECS では，コンポーネントが特権モードと非特権モードのどちらで実行されるかを指定でき，特権モードで動作するミドルウェアやデバイスドライバのコンポーネント化ができる．さらに，TECS の through プラグインを用いて，パーティション間通信のためのコンポーネントを挿入することで，コンポーネント間通信を抽象化できる．HR-TECS を実現する方法として，HRP2 カーネルを用いて，メモリ保護の設定やパーティション間通信の抽象化をする方法を具体的に述べ，また，HR-TECS を用いて開発することによって生じる，実行時間やメモリ使用量のオーバヘッドが抑えられていることを，評価実験により示した．

CHAPTER 6

周期タスクにおける応答時間の確率的解析

6.1 概要

本章では，複数の周期タスクで構成されるシステムを対象とし，タスクの応答時間の確率分布を解析する手法を提案する．これまでに提案されたタスクの応答時間を解析する手法では，タスクの実行時間を確率変数として扱うが，周期タスクの初期位相が確率変数である場合が考慮されておらず，初期位相が定数であるタスクセットしか解析することができない．本論文で提案する解析手法では，タスクの実行時間に加えて，初期位相を確率変数として扱い，タスクの応答時間分布を解析する．タスクの初期位相を確率変数として扱う場合，各タスクの初期位相が取りうる値の任意の組合せに対して，応答時間分布を解析する必要があり，解析時間が長くなると考えられる．そこで，本論文で提案する手法では，次に示す 2 つの手法により，応答時間分布の解析時間を短縮する．

(1) タスクの実行時間分布と初期位相分布を離散化する．

(2) 応答時間分布の終端部分に着目して数学的に解析する．

(1) については，まず，タスクの実行時間分布を離散化することにより，応答時間分布を数値解析で近似的に求めることができる．そして，初期位相分布を離散化することにより，応答時間分布を解析する初期位相の組合せを有限とすることができる．タスクの実行時間分布と初期位相分布の離散化については，離散化した確率分布を用いて解析した応答時間分布が，正確な応答時間分布よりも悲観的に近似されるように離散化誤差を丸める．

(2) については，応答時間分布の終端部分に着目して解析することによって，解析する初期位相の組合せ，および，解析する応答時間分布の区間を限定する．リアルタイムシステムの開発では，応答時間がデッドラインを越えるかどうかを検証することが重要であり，応答時間が長い区間について，確率分布を求めることが重要であると考える．そして，応答時間分布の終端部分は，確率密度が低く，シミュレータなどにより分布を求める場合，解析時間が長くなってしまう．そこで，応答時間分布の終端部分に着目して，数学的に解析する手法を提案することを考えた．

本章の研究内容による貢献は，以下のとおりである．

(4-1) 周期タスクの初期位相と実行時間を悲観的に離散化する方法を示す．

(4-2) 周期タスクの応答時間分布の終端部分を，初期位相と実行時間の確率分布から高速に解析する方法を示す．

本章の構成は次のとおりである．まず，関連研究について述べたあと，本章で扱うシステムのモデルについて述べる．次に，タスクの実行時間分布と初期位相分布を離散化する手法，および，タスクの応答時間分布を解析する手法について述べ，提案手法を評価する．

6.2 関連研究

文献 [12] では，タスクの最大実行時間を用いて解析した結果，デッドラインミスが発生すると判定されたとしても，実際にシステムが稼働する際には，デッドラインミスが発生しない場合や，デッドラインミスが許容される場合があるとしている．そして，スケジューラビリティの判定を，デッドラインミスが発生する確率や回数を用いて行うようなリアルタイムシステムとして，weakly hard real-time systems を提案している．

文献 [19]，[28] では，単一のプロセッサにおいて，互いに依存関係のない周期タスクからなるタスクセットを対象として，プリエンプティブな固定優先度ベースのスケジューリングのもとで，タスクの応答時間を確率的に解析する手法を提案している．ここでは，タスクの実行時間のみを確率変数として扱っている．提案されている手法では，対象とするタスクについて，ハイパーピリオド内のすべてのインスタンスについて応答時間分布を数学的に解析し，それらの結果を平均化することによって，タスクの応答時間分布を解析している．

文献 [62]，[63] では，自動車の電子システムを対象として，分散システムにおける端点間処理の応答時間を確率的に解析する手法を提案している．分散システム内の各プロセッサにおけるタスクの応答時間分布の解析には，文献 [19]，[28] で提案されている手法をベースとして用いており，加えて，ノンプリエンプティブなスケジューリングのもとで応答時間分布を解析する手法を提案している．また，プロセッサ間通信として CAN プロトコルによる通信を想定し，CAN メッセージの応答時間を近似する手法を提案している．そして，タスクの応答時間分布と CAN メッセージの応答時間分布を組み合わせることで，分散システムにおける端点間処理の応答時間分布を解析する手法を提案している．

文献 [20]，[39] では，文献 [19]，[28] で提案されている手法は，単純なモデルを扱う場合でのみ適用可能であると述べている．そして，資源の共有やリリースジッタ

などを考慮した，複雑なモデルを扱う場合において，確率を悲観的に近似するための手法を提案している．さらに，文献 [19]，[28] のモデルを拡張し，資源共有によるブロッキング時間やリリースジッタを考慮したモデルを扱い，応答時間分布を悲観的に近似する手法を示している．

しかしながら，これまでに提案されている解析手法では，対象とするタスクセットのモデルにおいて，周期タスクの初期位相が確率変数である場合が考慮されておらず，初期位相が定数であるタスクセットしか解析することができない．

6.3 システムのモデル

本章で扱うリアルタイムシステムは，単一のプロセッサのみで構成されるものとし，プロセッサ内には複数のタスクが存在するものとする．そして，タスクのスケジューリング方式は，プリエンプティブな固定優先度ベースのスケジューリングとする．また，タスク間の依存関係や割込みについては考慮しないものとする．

タスク τ_i は，(P_i, T_i, E_i, O_i) というパラメータによって特徴づけられるものとする．P_i，T_i は定数とし，それぞれ，τ_i の優先度，τ_i の起動周期を示す．ここで，P_i は正の整数であり，優先度が高いタスクほど，P_i の値は小さいものとする．E_i，O_i は確率変数とし，それぞれ，τ_i の実行時間分布，τ_i の初期位相分布を示す．ここで，初期位相とは，システムの起動後に初めて τ_i が起動する時刻である．E_i，O_i は，それぞれ，区間 $[E_i^{min}, E_i^{max}]$，$[0, T_i)$ においてのみ確率密度が正の値をとるものとし，この区間以外では，確率密度が 0 になるものとする．そして，O_i は区間 $[0, T_i)$ の一様分布に従うものとする．また，タスク τ_i の最大応答時間 R_i^{max} は，τ_i の起動周期 T_i 以下であるとする．ここで，P_i，T_i，E_i，O_i は，すべて独立である．以降，確率変数 X について，X の確率密度関数，累積確率分布関数をそれぞれ，f_X，F_X と表すものとし，X がある区間 I に属する確率を，$P(X \in I)$ のように表すものとする．

対象システムにおけるタスクの例として，$\tau_1 = (1, 5, E_1, O_1)$，$\tau_2 = (2, 10, E_2, O_2)$

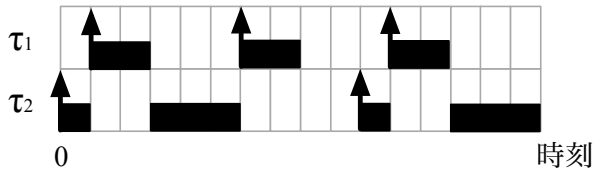

Figure 6.1: タスクの動作例

の 2 つのタスクで構成される場合の動作例を，図 6.1 に示す．ただし，図 6.1 では，各タスクの実行時間と初期位相を，$e_1 = 2$，$o_1 = 1$，$e_2 = 4$，$o_2 = 0$ に固定している．図中の横軸は，時刻を表しており，1 目盛りで 1 単位時間を表している．また，上向きの矢印は，タスクの起動を表しており，黒塗りのマスは，タスクが実行されていることを表している．

6.4 確率分布の離散化

リアルタイムシステムでは，タスクの応答時間が，デッドラインを満たすことが厳格に求められる．そのため，従来の決定的な応答時間解析では，タスクの最大実行時間などを用い，応答時間を悲観的に解析する．よって，応答時間を確率的に解析する場合においても，応答時間分布を悲観的に解析する必要がある [20][39]．そこで，タスクの実行時間分布と初期位相分布の離散化では，離散化した確率分布を用いて解析した応答時間分布が，正確な応答時間分布よりも悲観的な結果となるように離散化誤差を丸める．本節ではまず，確率分布における悲観論について説明する．その後，タスクの実行時間分布と初期位相分布を離散化する手法について，それぞれ述べる．

6.4.1 確率分布における悲観論

文献 [20]，[39] では，2 つの確率分布の間における悲観論を次のように定義している．

定義 6.1 ある 2 つの確率変数 X，Y について，任意の x に対し，$F_X(x) \leq F_Y(x)$ が成立するならば，X は Y よりも悲観的である．

ここで，正確な応答時間分布を R，R よりも悲観的な応答時間分布を R' とする．R' は，R よりも悲観的であるので，任意の t について，$F_{R'}(t) \leq F_R(t)$ が成立し，$1 - F_{R'}(t) \geq 1 - F_R(t)$ が成立する．t をデッドラインだと仮定すると，先述の式は，悲観的な応答時間分布から解析されるデッドラインミス率が，正確なデッドラインミス率よりも高いことを意味している．よって，定義 6.1 で定義されているように，悲観的に応答時間分布を解析すれば，デッドラインミス率を低く見積もることがなく，安全な解析結果が得られる．

6.4.2 実行時間分布の離散化

本節では，タスクの実行時間分布を離散化する手法について述べる．タスクの実行時間分布の離散化は，離散化後の確率分布を用いて解析した応答時間分布が，6.4.1 節で述べたように，正確な応答時間分布よりも悲観的になるように，離散化誤差を丸める．

文献 [20] において，実行時間分布と応答時間分布の関係について，次の定理が示されている．

定理 6.1 あるタスクの I つのインスタンスのみ，実行時間が確率変数 C であるとし，その他のパラメータはすべて定数であるとする．そして，C を用いて解析した応答時間分布を R とし，C よりも悲観的な確率分布 C' を用いて解析した応答時間分布を R' とする．このとき，R' は R よりも悲観的である．

定理 6.1 より，タスクの実行時間分布として，正確な実行時間分布よりも悲観的な確率分布を用いて応答時間分布を解析した場合，解析結果は正確な応答時間分布よりも悲観的になることが分かる．よって，タスクの実行時間分布の離散化では，基となる確率分布よりも悲観的な確率分布となるように離散化誤差を丸める．

そこで，τ_i の実行時間分布 E_i を離散化した確率分布 E_i^d について，次のように定義する．

定義 6.2 離散化の刻み幅を Δ_e とし，実際の τ_i の実行時間分布を E_i，E_i を離散化した実行時間分布を E_i^d とする．任意の整数 k について，$f_{E_i^d}(k\Delta_e)$ を次の式で定義する．

$$f_{E_i^d}(k\Delta_e) = P((k-1)\Delta_e < E_i \leq k\Delta_e) \tag{6.1}$$

ここで，定義 6.2 において，任意の時間 t に対する E_i^d の累積確率は，次の式を満たす．

$$F_{E_i^d}(t) = F_{E_i}(\left\lfloor \frac{t}{\Delta_e} \right\rfloor \Delta_e) \leq F_{E_i}(t) \tag{6.2}$$

定義 6.1 より，E_i^d は E_i よりも悲観的な分布である．よって，E_i^d を用いて解析した応答時間分布は，E_i を用いて解析した応答時間分布よりも悲観的になる．

6.4.3 初期位相分布の離散化

本節では，タスクの初期位相分布を離散化する手法について述べる．タスクの初期位相分布を離散化する際には，実行時間分布の離散化と同様に，離散化後の確率分布を用いて解析した応答時間分布が，正確な応答時間分布よりも悲観的になるように，離散化誤差を丸める．

本論文で提案する応答時間分布解析では，解析対象のタスク τ_i と，τ_i の実行を妨げるタスク τ_j の初期位相の差が必要となる．そのため，初期位相分布の離散化は，τ_i と τ_j の初期位相の差の離散確率分布を求めることによって行う．τ_i と τ_j の初期位相の差 $O_i - O_j$ の離散確率分布 O_{ij}^d について，次のように定義する．

定義 6.3 離散化の刻み幅を Δ_o とする．そして，O_i を離散化した確率分布として，O_i^U と O_i^L を，それぞれ次の式で定義する．

$$f_{O_i^U}(l\Delta_o) = P\left((l-1)\Delta_o < O_i \leq l\Delta_o\right) \tag{6.3}$$

$$f_{O_i^L}(l\Delta_o) = P\left(l\Delta_o \leq O_i < (l+1)\Delta_o\right) \tag{6.4}$$

ここで，l は任意の整数とする．さらに，O_j を離散化した確率分布として，O_j^U と O_j^L を，それぞれ O_i^U，O_i^L と同様に定義する．また，$O_{ij}^d = k\Delta_o$（k は任意の整数）のときの，τ_i の解析対象のインスタンスの起動時刻を a_i，a_i 以前で，どのタスクも実行中でない最も遅い時刻を a_0，a_0 以降で最も早い τ_j のインスタンスの起動時刻を a_j とする．

(1)$a_i \neq a_0$ のとき，$f_{O_{ij}^d}$ を次の式で定義する．

$$f_{O_{ij}^d}(k\Delta_o) = \begin{cases} \displaystyle\sum_{l\in Z} f_{O_i^L}(l\Delta_o) f_{O_j^U}\left((l-k)\Delta_o\right) & (a_j < a_i) \\ \displaystyle\sum_{l\in Z} f_{O_i^L}(l\Delta_o) f_{O_j^L}\left((l-k)\Delta_o\right) & (a_j > a_i) \\ \displaystyle\sum_{l\in Z} f_{O_i^L}(l\Delta_o) \Big(f_{O_j^U}((l-k)\Delta_o) & \\ +f_{O_j^L}((l-k)\Delta_o) \Big) & (a_j = a_i) \end{cases} \tag{6.5}$$

(2)$a_i = a_0$ のとき，$f_{O_{ij}^d}$ を次の式で定義する．

$$f_{O_{ij}^d}(k\Delta_o) = \begin{cases} \sum_{l \in Z} f_{O_i^U}(l\Delta_o) f_{O_j^L}((l-k)\Delta_o) & (a_j > a_i) \\ \sum_{l \in Z} \Big(f_{O_i^U}(l\Delta_o) f_{O_j^U}((l-k)\Delta_o) & \\ + f_{O_i^U}(l\Delta_o) f_{O_j^L}((l-k)\Delta_o) & (a_j = a_i) \\ + f_{O_i^L}(l\Delta_o) f_{O_j^U}((l-k)\Delta_o) \Big) & \end{cases} \tag{6.6}$$

ここで，定義 6.3 における，O_{ij}^d を用いて解析した応答時間分布が，正確な応答時間分布よりも悲観的であることを示す．

証明（式 6.5 について）まず，O_i の離散化について示す．式 6.5 では，O_i を離散化した分布として，O_i^L を用いている．これは，$a_i \leq a_i'$ を満たす a_i' を，離散化によって a_i に丸めることを示している．よって，τ_i の初期位相を小さくしたとき，つまり，a_i' を a_i に近づけたときの応答時間分布が，初期位相を小さくする前の応答時間分布よりも悲観的になることを示せばよい．

まず，τ_i の起動時刻が a_i である場合を考える．a_0 から a_i の間に起動したタスクによって，τ_i の実行が妨げられる時間の確率分布を W_i とする．そして，a_i 以降で，τ_i の実行を妨げうるタスクの起動時刻を a_k，実行時間分布を E_k とする．このときの，τ_i の応答時間分布 R_i について考える．R_i が，$(a_k - a_i) < t$ を満たす任意の時間 t 以上となる確率は，

$$\begin{aligned} &P(R_i \geq t) = \int_0^{a_k - a_i} \int_{(a_k - a_i) - x}^{\infty} f_{W_i}(x) f_{E_i}(y) P(t - y \leq E_k) dy dx \\ &+ \int_{a_k - a_i}^{\infty} f_{W_i}(x) P(t - x \leq E_i + E_k) dx \end{aligned} \tag{6.7}$$

となる．

次に，τ_i の起動時刻が a_i' である場合を考える．a_0 から a_i' の間に起動したタスクによって，τ_i の実行が妨げられる時間の確率分布を W_i' とする．このときの，τ_i の応答

時間分布 R'_i について考える．R'_i が，t 以上となる確率は，

$$\begin{aligned} &P(R'_i \geq t) = \int_0^{a_k - a'_i} \int_{(a_k - a'_i) - x}^{\infty} f_{W'_i}(x) f_{E_i}(y) P(t - y \leq E_k) dy dx \\ &+ \int_{a_k - a'_i}^{\infty} f_{W'_i}(x) P(t - x \leq E_i + E_k) dx \end{aligned} \tag{6.8}$$

となる．

ここで，$W_i > (a'_i - a_i)$ のとき，$f_{W_i}(x + a'_i - a_i) = f_{W'_i}(x)$ であること，および，6.5 節で述べるように，提案手法では，対象タスクが最大回数実行を妨げられる場合のみ解析することより，式 6.8 の第 1 項は，

$$\int_{a'_i - a_i}^{a_k - a_i} \int_{(a_k - a_i) - x}^{\infty} f_{W_i}(x) f_{E_i}(y) P(t - y \leq E_k) dy dx \tag{6.9}$$

となり，また，第 2 項も同様に，

$$\int_{a_k - a_i}^{\infty} f_{W_i}(x) P(t - (x - (a'_i - a_i)) \leq E_i + E_k) dx \tag{6.10}$$

となる．ここで，式 6.9 が式 6.7 の第 1 項よりも小さく，式 6.10 が式 6.7 の第 2 項よりも小さいことは明らかである．

よって，$P(R_i \geq t) \geq P(R'_i \geq t)$ が成立し，R_i は R'_i よりも悲観的であるといえるため，τ_i の初期位相を小さくしたときの応答時間分布が，初期位相を小さくする前の応答時間分布よりも悲観的になるといえる．

次に，O_j の離散化について，3 つの場合に分けて示す．

(a) $a_j < a_i$ の場合

O_j を離散化した分布として，O_j^U を用いている．これは，$a_j \geq a'_j$ を満たす a'_j を，離散化によって a_j に丸めることを示している．よって，τ_j の初期位相を大きくしたとき，つまり，a'_j を a_j に近づけたときの応答時間分布が，初期位相を大きくする前の応答時間分布よりも悲観的になることを示せばよい．

まず，τ_j の起動時刻が a_j である場合について考える．a_0 から a_j の間に起動したタスクによって，τ_j の実行が妨げられる時間の確率分布を W_j とする．このときの，τ_j が τ_i の実行を妨げる時間の確率分布 W_i について考える．W_i が，$0 < t$ を満たす任意の時間 t 以上となる確率は，

$$P(W_i \geq t) = \int_0^\infty f_{W_j}(x) P(t - (x - (a_i - a_j)) \leq E_j) dx \tag{6.11}$$

となる．

次に，τ_j の起動時刻が a'_j である場合について考える．a_0 から a'_j の間に起動したタスクによって，τ_j の実行が妨げられる時間の確率分布を W'_j とする．このときの，τ_j が τ_i の実行を妨げる時間の確率分布 W'_i について考える．W'_i が，t 以上となる確率は，

$$P(W'_i \geq t) = \int_0^\infty f_{W'_j}(x) P(t - (x - (a_i - a'_j)) \leq E_j) dx \tag{6.12}$$

となる．

ここで，$W_j > 0$ のとき，$f_{W_j}(x) = f_{W'_j}(x + (a_j - a'_j))$ であり，かつ，$W_j = 0$ のとき，$f_{W_j}(0) = \int_0^{a_j - a'_j} f_{W'_j}(x) dx$ であることより，式 6.11 は，

$$\begin{aligned} &\textstyle\int_0^{a_j - a'_j} f_{W'_j}(x) dx P(t + (a_i - a_j) \leq E_j) \\ &+ \textstyle\int_{a_j - a'_j}^\infty f_{W'_j}(x) P(t - (x - (a_i - a'_j)) \leq E_j) dx \end{aligned} \tag{6.13}$$

となる．そして，$0 \leq x \leq (a_j - a'_j)$ の範囲において，$t - (x - (a_i - a'_j)) \geq t + (a_i - a_j)$ であるため，式 6.12 は式 6.13（式 6.11）以下であることが分かる．

よって，$P(W'_i \geq t) \leq P(W_i \geq t)$ が成立し，W_i は W'_i よりも悲観的であるといえるため，W_i を用いて解析した応答時間分布は，W'_i を用いて解析した応答時間分布よりも悲観的であるといえる．そして，τ_j の初期位相を大きくしたときの応答時間分布

が，初期位相を大きくする前の応答時間分布よりも悲観的になるといえる．

(b) $a_j > a_i$ の場合

O_j を離散化した分布として，O_j^L を用いている．これは，$a_j \leq a'_j$ を満たす a'_j を，離散化によって a_j に丸めることを示している．よって，τ_j の初期位相を小さくしたとき，つまり，a'_j を a_j に近づけたときの応答時間分布が，初期位相を小さくする前の応答時間分布よりも悲観的になることを示せばよい．

まず，τ_j の起動時刻が a_j である場合について考える．τ_i の応答時間分布を R_i とし，τ_j に実行を妨げられない場合の，τ_i の応答時間分布を $\hat{R}_i$ とする．R_i が，$0 < t$ を満たす任意の時間 t 以下となる確率は，

$$P(R_i \leq t) = \begin{cases} \int_0^t f_{\hat{R}_i}(x)dx & (t \leq (a_j - a_i)) \\ \int_0^{a_j - a_i} f_{\hat{R}_i}(x)dx + & \\ \int_{a_j - a_i}^t \int_0^{x-(a_j-a_i)} f_{\hat{R}_i}(x-y) f_{E_j}(y)dydx & (t > (a_j - a_i)) \end{cases} \tag{6.14}$$

となる．

次に，τ_j の起動時刻が a'_j である場合について考える．このときの，τ_i の応答時間分布を R'_i とする．R'_i が，t 以下となる確率は，

$$P(R'_i \leq t) = \begin{cases} \int_0^t f_{\hat{R}_i}(x)dx & (t \leq (a'_j - a_i)) \\ \int_0^{a'_j - a_i} f_{\hat{R}_i}(x)dx + & \\ \int_{a'_j - a_i}^t \int_0^{x-(a'_j-a_i)} f_{\hat{R}_i}(x-y) f_{E_j}(y)dydx & (t > (a'_j - a_i)) \end{cases} \tag{6.15}$$

となる．

ここで，$a'_j > a_j$ より，$(a'_j - a_i) > (a_j - a_i)$ であるため，式 6.14 と式 6.15 より，$P(R'_i \leq t)$ は，$P(R_i \leq t)$ 以上であることが分かる．

よって，$P(R'_i \leq t) \geq P(R_i \leq t)$ が成立し，R_i は R'_i よりも悲観的であるといえるため，τ_j の初期位相を小さくしたときの応答時間分布が，初期位相を小さくする前の

応答時間分布よりも悲観的になるといえる．

(c) $a_j = a_i$ の場合

$a_j = a_i$ の場合は，$a_j < a_i$ の場合と $a_j > a_i$ の場合との境界となる場合であるため，$a_j < a_i$ である a_j に離散化誤差を丸めた確率と，$a_j > a_i$ である a_j に離散化誤差を丸めた確率の両者を含む．そのため，O_j を離散化した分布として，$O_j^U + O_j^L$ を用いており，そして，この分布を用いて解析した応答時間分布が悲観的であることは，先述の (a) と (b) の場合の証明より明らかである． □

証明（式 6.6 について）(d)$a_j > a_i$ の場合と，(e)$a_j = a_i$ の場合の 2 つに分けて示す．

(d) $a_j > a_i$ の場合

O_j を離散化した分布として，O_j^L を用いており，そして，O_i を離散化した分布として，O_i^U を用いている．これは，$a_j \leq a'_j$ を満たす a'_j を，離散化によって a_j に丸め，そして，$a_i \geq a'_i$ を満たす a'_i を，離散化によって a_i に丸めることを示している．よって，a'_j を a_j に近づけたとき，また，a'_i を a_i に近づけたときに，応答時間分布が悲観的になることを示せばよい．

a'_j を a_j に近づけたときについては，式 6.5 の証明における (b) の証明より，応答時間分布が悲観的になることが分かる．

a'_i を a_i に近づけたときについては，τ_i から見た τ_j の相対的な起動時刻を近づけると考えることができる．よって，式 6.5 の証明における (b) より，応答時間分布が悲観的になることが分かる．

(e) $a_j = a_i$ の場合

$a_j = a_i$ となる場合は次の 3 通りである．

(e-1) $a'_j \geq a_j$ を満たす a'_j を離散化によって a_j に丸めたとき，また，$a'_i \leq a_i$ を満たす a'_i を離散化によって a_i に丸めたとき

(e-2) $a'_j \leq a_j$ を満たす a'_j を離散化によって a_j に丸めたとき，また，$a'_i \geq a_i$ を満たす a'_i を離散化によって a_i に丸めたとき

(e-3) $a'_j \leq a_j$ を満たす a'_j を離散化によって a_j に丸めたとき，また，$a'_i \leq a_i$ を満たす a'_i を離散化によって a_i に丸めたとき

これらの場合のそれぞれについて，応答時間分布が悲観的になることを示す．

(e-1) の場合については，(d) の場合についての証明より，応答時間分布が悲観的になることが分かる．

(e-2) の場合については，式 6.5 の証明における O_i の離散化と，(a) の場合についての証明より，応答時間分布が悲観的になることが分かる．

(e-3) の場合について考える．まず，$a'_j < a'_i$ のときに，a'_j を a'_i に近づけると，式 6.5 の証明における (a) の場合の証明より，応答時間分布が悲観的になることが分かる．次に，$a'_j > a'_i$ のときに，a'_i を a'_j に近づけると，(d) の場合の証明より，応答時間分布が悲観的になることが分かる．そして，$a'_j = a'_i$ のときに，a'_j と a'_i を同時に $a_j (= a_i)$ に近づけると，a_j 以降に起動するタスクの相対的な起動時刻が a_j に近づくと見なすことができる．よって，(d) の証明より，応答時間分布が悲観的になることが分かる．
□

以上より，定義 6.3 に従って初期位相分布を離散化した O^d_{ij} を用い，6.5 節で述べる手法で応答時間分布を解析した場合，得られる応答時間分布は，正確な応答時間分布よりも悲観的になるといえる．

6.5 タスクの応答時間解析

本節では，まず，タスクの初期位相を定数とした場合における，応答時間の確率的な解析手法について述べる．次に，タスクの初期位相を確率変数とした場合における，応答時間の確率的な解析手法について述べる．

6.5.1 タスクの初期位相が定数の場合

タスクの初期位相を定数とした場合における，応答時間の解析は，文献 [28] で提案されている手法に基づいて行う．タスクの応答時間分布の解析は，タスクの実行時間分布の和の分布を繰り返し計算することによって行う．ここで，確率分布の和の分布の確率密度を計算するためには，畳込み積分が用いられる．

タスクの応答時間を解析するために，まず，対象とするタスク τ_i のインスタンス $\tau_{i,j}$ に対し，$\tau_{i,j}$ の起動時刻 $A_{i,j}$ における，優先度が P_i より高いタスクの残り実行時間の総和 $W^{P_i}_{A_{i,j}}$ を求める．ここで，優先度が P_i より高いタスクの集合を $hp(P_i)$ とする．

$W^{P_i}_{A_{i,j}}$ の解析は，次のような手順で行う．

(1) 時刻 $A_{i,j}$ 以前で，$hp(P_i)$ に属するどのタスクも実行中でない最も遅い時刻を t_0 とする．ここで，$W^{P_i}_{A_{i,j}}$ を 0 に初期化し，t を，t_0 以降で，$hp(P_i)$ に属するタスクが起動する最も早い時刻に初期化する．確率変数 $W^{P_i}_{A_{i,j}}$ を 0 に初期化するとは，$W^{P_i}_{A_{i,j}} = 0$ となる確率密度を 1，そうでない確率密度を 0 とすることを指す．

(2) $hp(P_i)$ に属するタスクの内，時刻 t で起動するタスクの集合を $At(t)$ とする．ここで，$At(t)$ に属するタスクの実行時間を，$W^{P_i}_{A_{i,j}}$ に加える．つまり，

$$W^{P_i}_{A_{i,j}} = W^{P_i}_{A_{i,j}} + \sum_{\forall \tau_k \in At(t)} E_k \tag{6.16}$$

となる．

(3) 時刻 t 以降で，$hp(P_i)$ に属するタスクが起動する最も早い時刻を t' とする．t' が $A_{i,j}$ 以上となるならば，t' を $A_{i,j}$ とする．ここで，$W^{P_i}_{A_{i,j}}$ の分布を，負の方向に，$t' - t$ だけ移動させる．このとき，$W^{P_i}_{A_{i,j}}$ が負の値となる分布については，0 の

分布に集約する．つまり，$W_{A_{i,j}}^{P_i}$ の確率密度関数 $f_{W_{A_{i,j}}^{P_i}}(w)$ は，

$$f_{W_{A_{i,j}}^{P_i}}(w) = \begin{cases} f_{W_{A_{i,j}}^{P_i}}(w + (t' - t)) & (w > 0) \\ \sum_{w'=-\infty}^{t'-t} f_{W_{A_{i,j}}^{P_i}}(w') & (w = 0) \\ 0 & (w < 0) \end{cases} \tag{6.17}$$

となる．

(4) t' が $A_{i,j}$ ならば解析を終了し，現時点での $W_{A_{i,j}}^{P_i}$ を解析結果とする．そうでなければ，t を t' に更新した後，(2) に戻る．

$W_{A_{i,j}}^{P_i}$ の解析が終了した後，その解析結果を用いて $\tau_{i,j}$ の応答時間 $R_{i,j}$ を求める．$R_{i,j}$ の解析は，次のような手順で行う．

(1) $R_{i,j}$ を $W_{A_{i,j}}^{P_i} + E_i$ に初期化する．$A_{i,j}$ において，$hp(P_i)$ に属するタスクが起動するならば，t を $A_{i,j}$ に初期化し，そうでなければ，t を，$A_{i,j}$ 以降で，$hp(P_i)$ に属するタスクが起動する最も早い時刻に初期化する．

(2) $R_{i,j}$ が τ_i の最大応答時間 R_i^{max} となる確率を確認し，その確率が 0 より大きいならば，解析を終了する．そうでなければ，(3) に進む．

(3) $hp(P_i)$ に属するタスクの内，時刻 t で起動するタスクの集合を $At(t)$ とする．ここで，$At(t)$ に属するタスクの実行時間を，$R_{i,j}$ の内，t を超える分布について加える．つまり，

$$R_{i,j} = \begin{cases} R_{i,j} + \sum_{\forall \tau_k \in At(t)} E_k & (R_{i,j} > t) \\ R_{i,j} & (R_{i,j} \leq t) \end{cases} \tag{6.18}$$

となる．

(4) 時刻 t 以降で，$hp(P_i)$ に属するタスクが起動する最も早い時刻を t' とする．t を t' に更新した後，(2) に戻る．

6.5.2 タスクの初期位相が確率変数の場合

タスクの初期位相を確率変数とした場合における応答時間の確率的な解析は，まず，各タスクの初期位相をある値に固定して，タスクの応答時間分布を求める．その後，固定した初期位相の組合せとなる確率を，求めたタスクの応答時間分布に掛け合わせ，その結果を足し合わせて平均化する．そのため，各タスクが取りうる初期位相の値すべての組合せに対し，タスクの応答時間分布を解析する必要があり，解析時間が長くなると考えられる．短時間で解析を行うための手法として，本論文では，タスクの応答時間分布の終端部分のみを解析する手法について述べる．

6.5.2.1 応答時間分布の解析

タスクの応答時間分布の終端部分を求めるために，対象とするタスクが，高優先度タスクに最大回数実行を妨げられる場合について解析を行う．まず，$hp(P_i)$ に属するタスク τ_j が，τ_i の実行を妨げる最大回数 C_j^{max} を求める．C_j^{max} は，τ_i の最大応答時間 R_i^{max} に対し，次の式で表される．

$$C_j^{max} = \left\lceil \frac{R_i^{max}}{T_j} \right\rceil \tag{6.19}$$

あるタスクが，高優先度タスクに最大回数実行を妨げられる場合の例を，表 6.1 に示すタスクセットを用いて示す．τ_3 について考えると，τ_3 の最大応答時間は 20 であり，τ_3 が τ_1，τ_2 に実行を妨げられる最大回数は，それぞれ，4 回，2 回である．よって，τ_3 が τ_1, τ_2 に最大回数実行を妨げられる場合の 1 つは，図 6.2 のように表すことができる．

次に，τ_i の応答時間分布の終端部分 R_i を求める．ここで，$hp(P_i)$ に属するタスク

Table 6.1: タスクセット 1

task	priority	period	E_i^{max}
τ_1	1	5	2
τ_2	2	10	4
τ_3	3	20	4

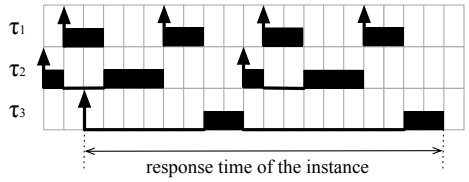

Figure 6.2: τ_3 が最大回数実行を妨げられる場合

$\tau_1, \tau_2, \cdots, \tau_{i-1}$ の初期位相の組 $(O_1, O_2, \cdots, O_{i-1})$ を，確率変数 Ψ_i とし，Ψ_i が取りうる値の集合を，Ω_i とする．R_i を解析する手順は次のとおりである．

(1) Ω_i に属する任意の ψ に対し，次の (2) と (3) の処理を行い，τ_i の応答時間分布 R_i^{ψ} を解析する．

(2) $hp(P_i)$ に属する任意の τ_j に対し，τ_j が，τ_i の実行を妨げる回数が C_j^{max} ならば，(3) へ進む．そうでなければ，$R_i^{\psi} = 0$ とする．

(3) τ_i の応答時間分布を 6.5.1 節で述べた手法により解析する．その解析結果を R_i^{ψ} とする．

(4) Ω_i に属する任意の ψ に対し，R_i^{ψ} の解析が終了したら，R_i を次の式により解析する．

$$f_{R_i}(r) = \sum_{\forall \psi \in \Omega_i} f_{\Psi_i}(\psi) f_{R_i^{\psi}}(r) \tag{6.20}$$

6.5.2.2 応答時間分布の信頼区間

提案手法により求めたタスクの応答時間分布の信頼区間について考える．ここで，ある確率変数 X に対し，X の確率密度が正しいことが保証されており，かつ，X の確率密度が 0 でない区間を，X の信頼区間と定義する．すなわち，実際のタスクの応答時間分布と，提案手法により求めたタスクの応答時間分布が一致する，確率密度が 0 でない区間である．本論文では，対象とするタスクが，高優先度タスクに最大回数実行を妨げられる場合について応答時間分布を解析するため，タスクの応答時間分布の信頼区間は，対象とするタスクが高優先度タスクによって最大回数実行を妨げられる場合の応答時間分布のみが存在する区間である．提案手法により求めた R_i の信頼区間 $(R_i^{from}, R_i^{to}]$ について，次の定理が成立する．

定理 6.2 提案手法により求めた R_i の信頼区間 $(R_i^{from}, R_i^{to}]$ について，R_i^{to} は，R_i^{max} であり，R_i^{from} は，$R_i^{max} - \min_{\tau_j \in hp(P_i)} E_j^{max}$ である．

証明 まず，R_i^{to} は，τ_i が高優先度タスクに最大回数実行を妨げられる場合における，τ_i の応答時間の最大値であり，それは，τ_i の最大応答時間 R_i^{max} である．

次に，R_i^{from} は，τ_i が高優先度タスクに最大回数実行を妨げられる場合以外における，τ_i の応答時間の最大値であり，それは，τ_i の最大応答時間から，$hp(P_i)$ に属するタスクを *1* 回実行したときの最大実行時間を引いた値の最大値である．よって，R_i^{from} は，$\max_{\tau_j \in hp(P_i)} \left(R_i^{max} - E_j^{max}\right)$ と表されるため，R_i^{from} は，$R_i^{max} - \min_{\tau_j \in hp(P_i)} E_j^{max}$ と表すことができる． □

6.6 評価実験

本節では，提案手法の有効性を評価するために行った評価実験について述べる．まず，実験内容の概要，および，実験環境について述べる．その後，実験結果と，実験結果に対する考察を述べる．

6.6.1 実験の概要

6.5 節では，応答時間分布を代数的に解析する手法について述べたが，本提案手法は，解析式が複雑であり，代数的に解析することはできない．そこで，提案手法の妥当性を評価するために，提案手法による解析において，離散化した確率分布を用い，数値解析によって応答時間分布を求め，その結果をモンテカルロシミュレーションと比較することとした．評価実験では，あるタスクセットを対象とし，そのタスクセット内で優先度が最低であるタスクの応答時間分布を解析した．タスクの応答時間分布解析は，モンテカルロシミュレーションによる解析と，提案手法による解析をそれぞれ行い，その結果得られた応答時間分布と，解析に要した時間を，それぞれ比較した．評価に用いたプログラムは，すべて C 言語で実装し，2.93GHz のプロセッサ，4GB のメモリを搭載したマシンを用いて実行した．解析に要した時間は，シミュレータ，提案手法による解析のそれぞれを実装したプログラムの処理に要した時間を，time コマンドを用いて取得した値とした．

応答時間分布の比較には，応答時間分布の累積確率分布を 1 から引いた値の常用対数を用いた．この値は，対応する時間 t を，応答時間が越える累積確率を表しており，t をデッドラインと仮定すると，デッドラインミス率を表す．

本論文で述べた提案手法は，既存手法 [28] を拡張したものであり，前提条件が既存手法と同じ（初期位相が定数として与えられる）タスクセットに関しては，既存手法を用いて応答時間分布を解析する．そのため，既存手法で解析可能なタスクセットを提案手法により解析した場合，解析時間は，既存手法と提案手法でほぼ同じになる．そこで，本評価実験では，タスクの初期位相が確率変数として与えられるタスクセットのみを対象とする．

また，提案手法について，タスクの実行時間分布と初期位相分布を離散化することによる影響を評価するため，実行時間分布の離散化の刻み幅（Δ_e）を変動させた場合，および，初期位相分布の離散化の刻み幅（Δ_o）を変動させた場合の解析結果をそ

Table 6.2: 評価対象のタスクセットの一つ

task	priority	period	E_i			
			E_i^{max}	E_i^{min}	μ_{E_i}	σ_{E_i}
τ_1	1	13	1.897	1.0	1.546	1.5
τ_2	2	38	6.355	1.0	4.689	1.5
τ_3	3	48	4.014	1.0	3.889	1.5
τ_4	4	49	3.439	1.0	2.694	1.5
τ_5	5	59	2.195	1.0	2.131	1.5
τ_6	6	71	10.42	1.0	8.205	1.5
τ_7	7	73	1.297	1.0	1.089	1.5

れぞれ比較した．

対象とするタスクセットには，タスクを 4 つから 7 つ含むタスクセットを，それぞれ 10 タスクセット（合計 40 タスクセット）用いた．各タスクの周期は，乱数を用いて生成した，10 から 100 の間の整数とした．そして，各タスクの実行時間分布は，区間 $[E_i^{min}, E_i^{max}]$ の切断正規分布に従うものとし，基となる正規分布の平均と分散をそれぞれ，μ_{E_i}，$\sigma_{E_i}^2$ とした．ここで，切断正規分布とは，確率変数の定義域が有限の確率分布である．基となる正規分布が $\hat{X}$ で，区間 $[a, b]$ の切断正規分布に従う確率変数 X の確率密度関数 f_X は，次の式で定義される．

$$f_X(x) = \begin{cases} \dfrac{f_{\hat{X}}(x)}{P(a \leq \hat{X} \leq b)} & (a \leq x \leq b) \\ 0 & otherwise \end{cases} \tag{6.21}$$

E_i^{max} と μ_{E_i} は，乱数を用いて生成した実数とし，E_i^{min} と σ_{E_i} は，それぞれ，1.0，1.5 とした．

本節では，評価実験を行ったタスクセットの内，表 6.2 に示すタスクセットについて，詳細に結果を述べる．表 6.2 のタスクセットについては，τ_4，τ_5，τ_6，τ_7 の応答時間分布を解析した．

シミュレータは，入力として，タスクセットの情報，応答時間計測の対象とする

Table 6.3: Δ_e を変化させた場合の解析時間の比較（$\Delta_o = 1.0$）

		タスク数			
		4	5	6	7
Δ_e	0.1	0.151	1.207	702.416	8950.128
	0.2	0.102	0.446	203.220	2605.243
	0.4	0.008	0.212	64.910	871.049
シミュレータ		3058.291	4046.404	5992.767	6993.226

Table 6.4: Δ_o を変化させた場合の解析時間の比較（$\Delta_e = 0.1$）

		タスク数			
		4	5	6	7
Δ_o	1.0	0.151	1.207	702.416	8950.128
	2.0	0.111	0.157	22.268	130.637
	4.0	0.093	0.086	1.042	3.747
シミュレータ		3058.291	4046.404	5992.767	6993.226

タスク，シミュレーションの試行回数，出力する離散応答時間分布の刻み幅を与える．そして，モンテカルロシミュレーションにより，計測対象タスクの離散応答時間分布を解析し，結果として，確率分布，および，累積確率分布を出力する．本実験では，シミュレーションの試行回数は，10^9 回とし，出力する離散応答時間分布の刻み幅は，0.1 とした．

6.6.2 解析時間の比較

表 6.2 のタスクセットについて，各タスクの応答時間分布解析に要した時間を，表 6.3 と表 6.4 に示す．

表 6.3 は，Δ_o を 1.0 に固定し，Δ_e を 0.1，0.2，0.4 とした場合の解析時間を，タスクセットに含まれるタスク数ごとにまとめたものである．そして，表 6.4 は，Δ_e を 0.1 に固定し，Δ_o を 1.0，2.0，4.0 とした場合の解析時間を，タスクセットに含まれるタスク数ごとにまとめたものである．ここで，表 6.3 と表 6.4 の一番下の行は，シ

ミュレータによる解析時間を示している．そして，それぞれの表中における解析時間の単位は秒である．

表 6.3 と表 6.4 をみると，提案手法を用いて解析した場合，シミュレータを用いて解析した場合よりも，解析時間を大きく短縮できていることがわかる．ここで，タスク数が 7 つであり，$\Delta_e = 0.1$，$\Delta_o = 1.0$ の場合の提案手法による解析時間は，シミュレータによる解析時間よりも長くなっているが，これは，シミュレータの試行回数が少ないことが原因である．タスク数が 7 つの場合の応答時間分布をみると，シミュレータによって得られた最大応答時間は，32.2 であり，そのときの累積確率は，$10^{-8.699}$ であった．一方，提案手法によって得られた最大応答時間は，33.5 であり，そのときの累積確率は，$10^{-12.741}$ であった．よって，タスク数が 7 つの場合に，シミュレータにより，応答時間分布の終端部分を解析するには，本実験の約 1000 倍の試行回数が必要であると考えられるため，提案手法を用いることで，解析時間を大きく短縮できているといえる．

表 6.3 をみると，Δ_e を大きくしたときの解析時間が短くなっていることが分かる．これは，Δ_e が畳込み積分の計算量に影響していることが原因であると考えられる．Δ_e を小さくすると，畳込み積分の対象となる 2 つの離散確率分布関数の刻み幅がそれぞれ小さくなり，計算する組合せが多くなる．具体的には，全体の計算量は，最大で，$1/\Delta_e^2$ に比例して多くなると考えられる．よって，提案手法では，Δ_e を大きくすることによって，解析に要する時間を短くできたと考えられる．

表 6.4 をみると，Δ_o を大きくしたときの解析時間が短くなっていることが分かる．これは，Δ_o が初期位相の組合せの数に影響していることが原因であると考えられる．Δ_o を小さくすると，初期位相の組合せが多くなり，応答時間分布を解析する回数が増える．具体的には，全体の計算量は，最大で，解析の対象となるタスクの実行を妨げるタスク数 N に対し，$1/\Delta_o^N$ に比例して多くなると考えられる．よって，提案手法では，タスク数が増えるにつれて解析に要する時間が急激に長くなるが，Δ_o を大きくすることによって，解析に要する時間を短くできたと考えられる．

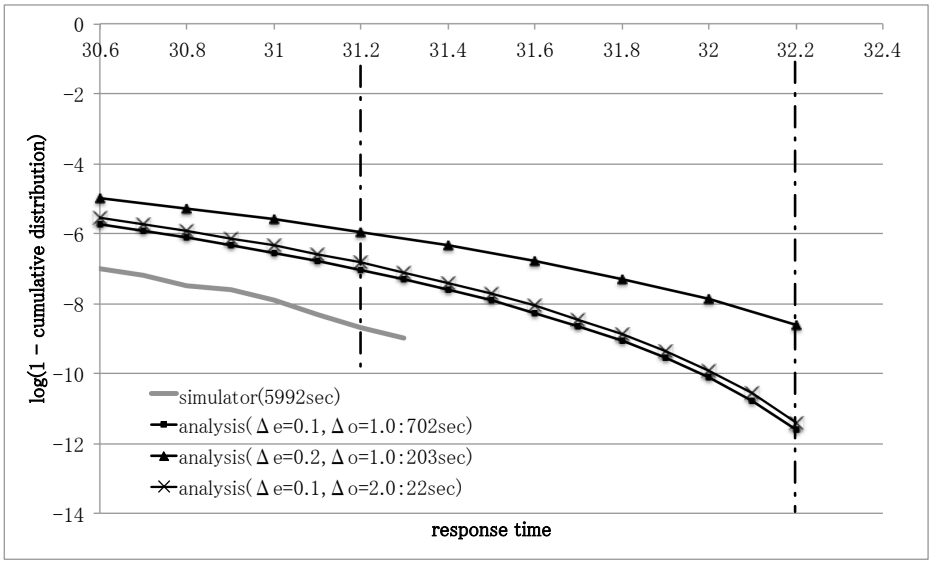

Figure 6.3: τ_6 の応答時間分布

すべての評価対象のタスクセットについて確認したところ，解析時間について，表 6.2 以外のタスクセットについても，表 6.2 のタスクセットの場合と同様の傾向がみられることがわかった．

6.6.3 応答時間分布の比較

図 6.3 と図 6.4 は，表 6.2 の τ_6 について，得られた応答時間分布をまとめ，その終端部分を拡大したものである．ここで，図 6.4 は，提案手法によって得られる応答時間分布の信頼区間内における，応答時間分布の比較結果を示している．図 6.3，図 6.4 のグラフの横軸はタスクの応答時間を，縦軸は応答時間分布の累積確率分布を 1 から引いた値の常用対数をそれぞれ表している．また，図 6.3 における 2 つの鎖線は，提案手法によって得られる応答時間分布の信頼区間 $(31.2, 32.2]$ の上限と下限を表している．

提案手法によって得られた応答時間分布（analysis）と，シミュレータによって得られた応答時間分布（simulator）を比較すると，両者の分布が近似していることが分

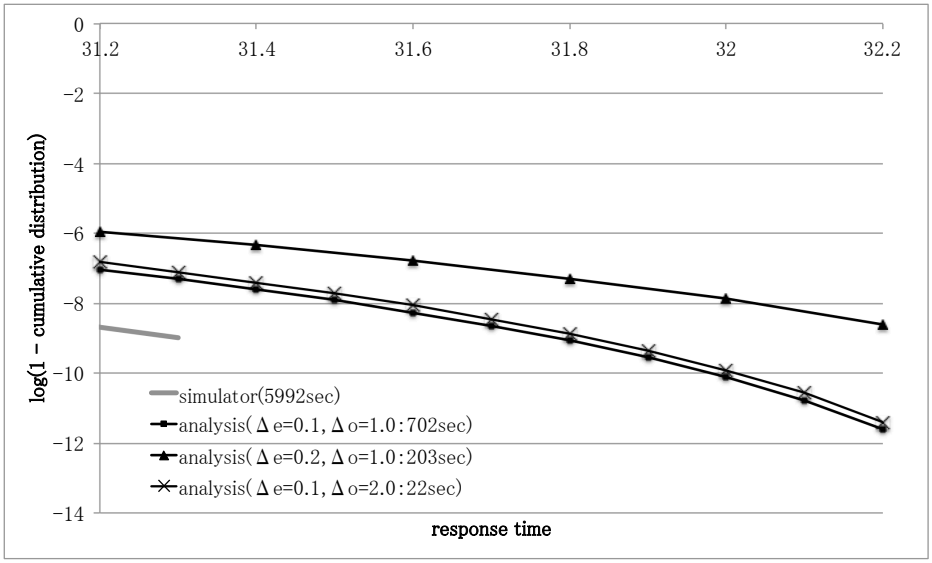

Figure 6.4: 信頼区間における τ_6 の応答時間分布

かる．ここで，信頼区間の範囲内を見ると，analysis が simulator を上回っていることが分かる．これは，提案手法において，タスクの実行時間分布と初期位相分布を離散化した際に，得られる応答時間分布が悲観的になるように離散化誤差を丸めたためである．そして，信頼区間の範囲外を見ると，信頼区間から遠ざかるにつれ，analysis と simulator が乖離していることが分かる．これは，信頼区間の範囲外では，τ_6 が他のタスクに最大回数実行を妨げられる場合以外の応答時間分布が含まれるためである．また，応答時間 31.3 付近を見ると，simulator の分布が途切れていることが分かる．これは，simulator によって得られる確率の限界（10^{-9}）を越えてしまい，正確な応答時間分布を出力できなかったためである．

analysis について，Δ_e と Δ_o を変動させた場合をそれぞれ比較する．比較項目としては，応答時間分布に関する誤差の最大値と平均値を用いた．

表 6.2 のタスクセットについて，解析によって得られた各タスクの応答時間分布に関する誤差の最大値と平均値を，表 6.5 と表 6.6 に示す．

表 6.5 は，Δ_o を 1.0 に固定し，Δ_e を 0.1，0.2，0.4 とした場合の実験結果を，タスクセットに含まれるタスク数ごとにまとめたものである．そして，表 6.6 は，Δ_e を

Table 6.5: Δ_e を変化させた場合における応答時間分布の誤差の比較（$\Delta_o = 1.0$）

		タスク数			
		4	5	6	7
Δ_e	0.1	2.105 / 1.283	1.535 / 1.333	1.691 / 1.675	-
	0.2	1.667 / 1.080	1.926 / 1.103	2.994 / 1.807	3.076 / 2.209
	0.4	3.062 / 2.317	3.425 / 2.477	5.162 / 3.848	5.977 / 4.646

Table 6.6: Δ_o を変化させた場合における応答時間分布の誤差の比較（$\Delta_e = 0.1$）

		タスク数			
		4	5	6	7
Δ_o	1.0	2.105 / 1.283	1.535 / 1.333	1.691 / 1.675	-
	2.0	0.294 / 0.294	0.307 / 0.307	0.201 / 0.201	0.361 / 0.361
	4.0	0.793 / 0.793	0.826 / 0.826	0.628 / 0.628	1.008 / 1.008

0.1 に固定し，Δ_o を 1.0，2.0，4.0 とした場合の実験結果を，タスクセットに含まれるタスク数ごとにまとめたものである．ここで，表 6.5 と表 6.6 において，応答時間分布に関する誤差は，常用対数軸上での差分を記載している．そして，それぞれの表中では，応答時間分布に関する誤差を，(最大値) / (平均値) と表記しており，誤差を求めることができなかった場合には，“-”と表記している．また，それぞれの表中において，$\Delta_e = 0.1$，かつ，$\Delta_o = 1.0$ のときの値は，シミュレータの出力した応答時間分布との誤差を示しており，それ以外のときの値は，提案手法を用いて，$\Delta_e = 0.1$，かつ，$\Delta_o = 1.0$ とした場合に解析された応答時間分布との誤差を示している．すべての提案手法について，シミュレータの出力した応答時間分布と比較しなかった理由は，シミュレータの出力した応答時間分布の終端部分について，分布が得られなかった部分があったためである．

Δ_e が小さい場合と大きい場合とを比較すると，Δ_e が小さい場合よりも，Δ_e が大きい場合の方が，応答時間分布が大きくなっている．また，すべてのタスクセットにおいて，信頼区間内では，提案手法による解析結果の応答時間分布が，シミュレー

ション結果の応答時間分布を上回っており，Δ_e が小さい場合よりも，Δ_e が大きい場合の方が，応答時間分布が大きくなっていることを確認した．この結果から，提案手法では，解析結果の応答時間分布が悲観的になるように，実行時間分布を離散化できていることを確認でき，さらに，実行時間分布の離散化の刻み幅を大きくすることによって，解析結果の応答時間分布がより悲観的になることが分かった．

Δ_o が小さい場合と大きい場合とを比較すると，Δ_o が小さい場合よりも，Δ_o が大きい場合の方が，応答時間分布が大きくなっている．また，すべてのタスクセットにおいて，信頼区間内では，提案手法による解析結果の応答時間分布が，シミュレーション結果の応答時間分布を上回っており，Δ_o が小さい場合よりも，Δ_o が大きい場合の方が，応答時間分布が大きくなっていることを確認した．この結果から，提案手法では，解析結果の応答時間分布が悲観的になるように，初期位相分布を離散化できていることを確認でき，さらに，初期位相分布の離散化の刻み幅を大きくすることによって，解析結果の応答時間分布がより悲観的になることが分かった．

6.6.4 終端部分のみを解析することの効果

提案手法において，応答時間分布の終端部分のみを解析することによる，解析の高速化の有効性を評価するために，比較実験を行った．表 6.2 の τ_6 について，提案手法によって，応答時間分布の終端部分のみを解析した結果と，提案手法から，6.5.2.1 節で述べた，解析を行う場合の限定条件を取り除き，応答時間分布全体を解析した場合の結果とを比較した．ここで，離散化の刻み幅については，$\Delta_e = 0.2$，かつ，$\Delta_o = 1.0$ として解析した．

その結果，解析時間については，応答時間分布の終端部分のみを解析した場合は表 6.3 のとおり，203.220 秒であった．一方，応答時間分布全体を解析した場合は，2553.309 秒であり，終端部分のみを解析した場合の約 13 倍であった．このように，応答時間分布の終端部分のみを解析することによって，解析時間を大きく短縮できていることが確認できた．また，2 つの解析手法によって得られた応答時間分布を比較し

た結果，信頼区間内において，両者の応答時間分布が一致していることを確認した．

以上の結果から，提案手法を用いることで，タスクの応答時間分布の終端部分について，短時間で，悲観的な解析ができることを示した．そして，提案手法では，タスク数が増えるにつれて解析に要する時間が急激に長くなるが，タスクの実行時間分布と初期位相分布の離散化の刻み幅を大きくし，悲観的に応答時間分布を解析することによって，解析に要する時間を短くできることを示した．

6.7 まとめ

本章では，タスクの実行時間に加えて，初期位相を確率変数として扱い，タスクの応答時間分布を解析する手法を提案した．提案手法では，タスクの実行時間分布と初期位相分布を離散化して扱い，さらに，応答時間分布の終端部分に着目して数学的に解析することによって，解析時間の短縮を実現した．タスクの実行時間分布と初期位相分布の離散化について，離散化した確率分布を用いて解析した結果の応答時間分布が，正確な応答時間分布よりも悲観的に近似されるように離散化誤差を丸める手法を提案し，その手法により，応答時間分布が悲観的に近似されることを証明した．応答時間分布の終端部分に着目した解析について，解析を行う条件を提案し，応答時間分布の終端部分を解析する手法を提案した．そして，提案した解析手法によって得られる応答時間分布の信頼区間を示した．提案手法の有効性を評価するための実験を実施し，その結果，提案手法を用いることで，タスクの応答時間分布の終端部分を短時間で，悲観的に解析できることを示した．そして，提案手法では，タスク数が増えるにつれて解析に要する時間が長くなるが，タスクの実行時間分布と初期位相分布の離散化の刻み幅を大きくし，解析結果の応答時間分布をより悲観的にすることによって，解析に要する時間を短くできることを示した．

CHAPTER 7

結論

7.1 まとめ

本論文では，組込みシステム向けのソフトウェアコンポーネント技術である，TECS コンポーネントモデルに基づき，メモリ保護を考慮した組込みシステム向けのソフトウェアのコンポーネントベース開発が可能なコンポーネント技術，HR-TECS を提案した．HR-TECS では，アプリケーションソフトウェアだけでなく，ミドルウェアやデバイスドライバのようなソフトウェアプラットフォームのコンポーネント化もできることを考慮した．

まず，TECS を用いて，NXT のソフトウェアプラットフォームをコンポーネントベース開発した事例について述べ，ソフトウェアプラットフォームをコンポーネント化することの有効性，および，TECS をソフトウェアプラットフォーム開発に用いることの有用性を評価した．プラットフォームをコンポーネント化することで，ターゲットシステムのハードウェア構成やアプリケーション要求に応じて，容易にプラットフォームの構成を変更でき，メモリ使用量や割込みサービスルーチンの応答時間の無駄を省くことができた．そして，TECS を用いることでプラットフォームをコンポーネント化でき，また，オーバヘッドも抑えられていることを確認できた．

次に，メモリ保護機能を提供するためのリアルタイム OS である，HRP2 カーネルを提案した．HRP2 カーネルでは，静的コンフィギュレーションにより，メモリ配置を静的に決定することで，MPU に対応でき，また，メモリ保護のための情報を静的に生成できるようにした．そして，SH2A と ARM Cortex-M3 の MPU を用いて，HRP2 カーネルの実現方法を述べ，HRP2 カーネルにより，アプリケーション開発者が，MPU を意識せずに，メモリ保護を考慮したソフトウェア開発をできるようにした．実装した HRP2 カーネルを用いて，サービスコール呼出しやディスパッチの実行時間，および，メモリ使用量を評価した．サービスコールの実行時間については，メモリ保護機能を提供しない場合と比較して，最大で約 3 倍大きくなったが，実行時間の値は一定であり，リアルタイム性（実行時間の予測可能性）はあることを確認した．また，メモリ使用量については，メモリ保護機能を追加したことによる RAM 使用量の増加量が，約 150byte に抑えられていることを確認した．

そして，TECS コンポーネントモデルを拡張し，メモリ保護を考慮したソフトウェア開発を可能としたコンポーネント技術，HR-TECS を提案した．HR-TECS は，コンポーネント単位でのソフトウェアのメモリ領域に対するパーティショニング設定ができるように，TECS を拡張して実現した．また，HR-TECS では，非特権モードで動作するアプリケーションと，特権モードで動作するミドルウェアやデバイスドライバを，併せてコンポーネントベース開発できるようにした．さらに，TECS の through プラグインを用いて，コンポーネントのパーティション配置に合わせて，適切な通信処理コンポーネントを挿入することで，コンポーネント間の通信処理を，個々のコンポーネントの開発者に対して抽象化できる．HR-TECS を実現する方法として，HRP2 カーネルを用いて，メモリ保護の設定や，パーティション間通信の抽象化をする方法を具体的に述べ，また，HR-TECS を用いて開発することによって生じる，実行時間やメモリ使用量のオーバヘッドが抑えられていることを，評価実験により示した．

さらに，リアルタイムシステムのスケジューラビリティ解析において，周期タスクの応答時間の確率分布を解析する手法を提案した．提案手法では，タスクの実行時

間だけでなく，さらに，初期位相も確率変数として扱い，タスクの応答時間分布を解析する．提案手法における解析では，タスクの実行時間分布と初期位相分布を離散化して扱い，さらに，応答時間分布の終端部分に着目して数学的に解析することによって，解析時間を短縮した．タスクの実行時間分布と初期位相分布の離散化について，解析結果の応答時間分布が，正確な応答時間分布よりも悲観的，つまり，安全に見積もられるように，離散化誤差を丸める手法を提案，証明した．応答時間分布の終端部分に着目した解析について，解析を行う条件と解析方法，および解析の結果得られる応答時間分布の信頼区間を示した．提案手法の有効性を評価するために，モンテカルロシミュレーション結果との比較実験を行い，提案手法を用いることで，タスクの応答時間分布の終端部分を短時間で，悲観的に解析できることを示した．そして，提案手法では，タスク数が増えるにつれて解析に要する時間が長くなるが，タスクの実行時間分布と初期位相分布の離散化の刻み幅を大きくし，解析結果の応答時間分布をより悲観的にすることによって，解析に要する時間を短くできることを示した．

7.2 今後の課題

本論文では，HRP2 カーネルや HR-TECS の評価を，小規模なマイクロベンチマークソフトウェアを用いて実施した．今後は，実システムで用いられるようなアプリケーションを対象とし，HRP2 カーネルと HR-TECS のスケーラビリティ，および，アプリケーションやプラットフォームの実装容易性を評価することが課題であると考える．また，本論文では，GNU ツールチェーンの開発環境を対象として，HRP2 カーネルと HR-TECS を開発したが，GHS 社や IAR 社，ARM 社の提供する，GNU とは異なる開発環境を対象として，HRP2 カーネルと HR-TECS を実装する必要があると考えている．開発環境が異なることによる主な影響は，リンカスクリプトの違いであり，リンカスクリプトの種類によっては，ATT_MOD を使用できない可能性があるため，HRP2 カーネルにおけるリンカスクリプトの出力内容や，HR-TECS におけるメ

モリ保護設定のための静的 API 出力内容を変えることを考慮する必要がある．

また，AUTOSAR 仕様に準拠したソフトウェアプラットフォームを，本論文で述べた手法を用いて実現することも今後の課題として考えている．AUTOSAR OS 仕様は，メモリ保護に関する静的コンフィギュレーションについては規定していないが，カーネルオブジェクトを静的に生成したり，コンポーネントを静的にパーティションに配置したりするなど，本論文の手法に類似する点が多い．そのため，本論文で述べた手法は，AUTOSAR 仕様を実現する際にも適用可能であると考えている．

また，応答時間分布の解析については，異なるタスクセットのモデルへの対応をする必要があると考えられる．今回扱ったシステムは，周期タスクのみが存在するタスクセットを対象としており，プリエンプティブな優先度ベースのスケジューリングのみを対象としている．そこで，タスク間の依存関係や割込みを考慮する場合や，スケジューリングポリシーの異なる場合についての解析手法を提案し，より複雑なタスクセットにおける解析手法を提案することが必要であると考えられる．

謝辞

本論文に関する研究の機会を与えていただき，多くのご指導，ご鞭撻を賜りました，名古屋大学情報科学研究科の高田広章教授と本田晋也准教授に心から感謝致します．また，本研究を進めるにあたり，多くの有益なご意見とご協力をいただきました，オークマ株式会社の大山博司氏，立命館大学情報理工学部の安積卓也助教，名古屋大学情報科学研究科の松原豊特任助教に深く感謝致します．

本論文を執筆するにあたり，貴重なお時間をいただき，論文の構成に関する有益なご意見を賜り，さらに，論文の内容に関する議論の機会を設けていただいた，名古屋大学情報科学研究科の関浩之教授に深く感謝致します．

研究生活を送る上で，日頃からご支援いただき，また，知的刺激や活力を与えていただきました，名古屋大学情報科学研究科の枝廣正人教授や加藤真平講師，安藤友樹氏をはじめとする，枝廣・高田研究室のみなさま，そして，柴田誠也氏，一場利幸氏をはじめとする，高田研究室の卒業生のみなさまにも，深く感謝致します．

本研究における開発成果を公開，普及するにあたり，NPO 法人 TOPPERS プロジェクトのご協力をいただきました．NXT 用ソフトウェアプラットフォームと教材の開発にご協力いただきました株式会社リコーの竹内良輔氏，HRP2 カーネルの仕様策定に携わっていただきましたみなさま，TECS の開発に携わっていただきました TECS ワーキンググループのみなさまに感謝致します．

最後に，長い研究生活の間，私生活の面から支えてくれた家族に心から感謝致します．

参考文献

[1] Airlines Electronic Engineering Committee: *AVIONICS APPLICATION SOFTWARE STANDARD INTERFACE*, 2008.

[2] ARM: *ARM Architecture Reference Manual*.

[3] ARM: *Cortex-M3 Technical Reference Manual*.

[4] ARM: *Cortex-R4 and Cortex-R4F Technical Reference Manual*.

[5] ATMEL: AT91SAM7S Series Preliminary, http://www.atmel.com/.

[6] AUTOSAR: http://www.autosar.org/.

[7] AUTOSAR: Specification of Operating System V4.1.0, 2010.

[8] Azumi, T., Oyama, H., and Takada, H.: A Memory Allocator for Efficient Task Communications by Using RPC Channels in an Embedded Component System, *Proceedings of the 9th IASTED International Conference on Software Engineering and Applications*, Nov. 2008, pp. 204–209.

[9] Azumi, T., Oyama, H., and Takada, H.: Optimization of Component Connections for an Embedded Component System, *Proc. IEEE/IFIP International Conference on Embedded and Uniquitous Computing (EUC)*, Vancouver, Canada, Aug. 2009, pp. 182–188.

[10] Azumi, T., Ukai, T., Oyama, H., and Takada, H.: Wheeled Inverted Pendulum with Embedded Component System: a Case Study, *Proc. the 13th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC 2010)*, Carmona, Spain, May 2010, pp. 151–155.

[11] Azumi, T., Yamamoto, M., Kominami, Y., Takagi, N., Oyama, H., and Takada, H.: A New Specification of Software Components for Embedded Systems, *Proc. 10th IEEE International Symposium on Object/component/service-Oriented Real-Time Distributed Computing*, May 2007, pp. 46–50.

[12] Bernat, G., Burns, A., and Llamosí, A.: Weakly Hard Real-Time Systems, *IEEE Transactions on Computers*, Vol. 50, No. 4(2001), pp. 308–321.

[13] Biswas, S., Carley, T., Simpson, M., Middha, B., and Barua, R.: Memory Overflow Protection for Embedded Systems Using Run-Time Checks, Reuse, and Compression, *ACM Transactions on Embedded Computing Systems*, Vol. 5, No. 4(2006), pp. 719–752.

[14] Bruneton, E., Coupaye, T., Leclercq, M., Quéma, V., and Stefani, J.-B.: The FRACTAL component model and its support in Java: Experiences with Auto-adaptive and Reconfigurable Systems, *Software - Practice and Experience*, Vol. 36, No. 11-12(2006), pp. 1257–1284.

[15] Bruns, F., Kuschnerus, D., Showk, A., and Bilgic, A.: An Extensible Partitioning Framework for Safety-Critical Systems, *Embedded Real-Time Software and Systems 2012*, Feb. 2012.

[16] Chase, J. S., Levy, H. M., Feeley, M. J., and Lazowska, E. D.: Sharing and protection in a single-address-space operating system, *ACM Transactions on Computer Systems*, Vol. 12, No. 4(1994), pp. 271–307.

[17] Community, L.: The L4 headquarters, http://l4hq.org.

[18] COMPASS: Component Based Automotive System Software: http://embsys.technikum-wien.at/projects/compass/index.php.

[19] Díaz, J. L., García, D. F., Kim, K., and et al.: Stochastic Analysis of Periodic Real-Time Systems, *Proc. the 23rd IEEE Real-Time Systems Symposium*, 2002, pp. 289.

[20] Díaz, J. L., López, J. M., García, M., and et al.: Pessimism in the Stochastic Analysis of Real-Time Systems: Concept and Applications, *Proc. the 25th IEEE International Real-Time Systems Symposium*, 2004, pp. 197–207.

[21] Dubey, A., Karsai, G., and Mahadevan, N.: A component model for hard real-time systems: CCM with ARINC-653, *Software - Practice and Experience*, Vol. 41, No. 12(2011), pp. 1517–1550.

[22] ET ソフトウェアデザインロボットコンテスト: http://www.etrobo.jp/.

[23] The FreeRTOS Project: *Using the FreeRTOS Real Time Kernel - A Practical Guide*.

[24] Galla, T., Schreiner, D., Forster, W., Kutscherat, C., Goschka, K., and Horauer, M.: Refactoring an Automotive Embedded Software Stack using the Component-Based Paradigm, *Proceedings of the International Symposium on Industrial Embedded Systems, 2007*, July 2007, pp. 200–208.

[25] The GNU Project: *Documentation for binutils 2.22*, http://sourceware.org/binutils/docs/ld/index.html.

[26] International Electrotechnical Commission: *IEC61508 - Functional safety of electrical/electronic/programmable electronic safety-related systems*, 2010.

[27] International Organization for Standardization: *ISO 26262:2011, Road vehicles – Functional safety*, 2011.

[28] Kim, K., Díaz, J. L., Bello, L. L., and et al.: An Exact Stochastic Analysis of Priority-Driven Periodic Real-Time Systems and Its Approximations, *IEEE Transactions on Computers*, Vol. 54, No. 11(2005), pp. 1460–1466.

[29] Kim, S. H. and Jeon, J. W.: Introduction for Freshmen to Embedded Systems Using LEGO Mindstorms, *IEEE Transactions on Education*, Vol. 52, No. 1(2009), pp. 99–108.

[30] Kumar, R., Kohler, E., and Srivastava, M.: Harbor: software-based memory protection for sensor nodes, *Proceedings of the 6th international conference on Information processing in sensor networks*, Apr. 2007, pp. 340–349.

[31] Kuz, I., Liu, Y., Gorton, I., and Heiser, G.: CAmkES: A component model for secure microkernel-based embedded systems, *Journal of Systems and Software*, Vol. 80, No. 5(2007), pp. 687–699.

[32] LegoEducation: http://www1.lego.com/education/.

[33] LegoMindstormsNXT: http://mindstorms.lego.com/.

[34] Lehoczky, J. P.: Fixed priority scheduling of periodic task sets with arbitrary deadlines, *Proceedings of the 11th IEEE Real-Time Systems Symposium*, 1990, pp. 201–209.

[35] Leiner, B., Schlager, M., Obermaisser, R., and Huber, B.: A Comparison of Partitioning Operating Systems for Integrated Systems, *Proceedings of the 26th International Conference on Computer Safety, Reliability and Security*, Sep. 2007, pp. 342–355.

[36] leJOS: http://lejos.sourceforge.net/.

[37] Liu, C. L. and Layland, J. W.: Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment, *Journal of the ACM*, Vol. 20, No. 1(1973), pp. 46–61.

[38] Loiret, F., Navas, J., Babau, J.-P., and Lobry, O.: Component-Based Real-Time Operating System for Embedded Applications, *Proceedings of the 12th International Symposium on Component-Based Software Engineering*, CBSE '09, June 2009, pp. 209–226.

[39] López, J. M., Díaz, J. L., Entrialgo, J., and García, D.: Stochastic analysis of real-time systems under preemptive priority-driven scheduling, *Real-Time Systems*, Vol. 40, No. 2(2008), pp. 180–207.

[40] Makowitz, R. and Temple, C.: Flexray - A communication network for automotive control systems, *Proceedings of IEEE International Workshop on Factory Communication Systems*, 2006, pp. 207–212.

[41] Medvidovic, N. and Taylor, R. N.: A Classification and Comparison Framework for Software Architecture Description Languages, *IEEE Transactions on Software Engineering*, Vol. 26, No. 1(2000), pp. 70–93.

[42] Microsoft Corporation: Microsoft Component Object Model, http://www.microsoft.com/com/default.mspx.

[43] NPO 法人 TOPPERS プロジェクトコンポーネント仕様ワーキンググループ: 組込みコンポーネントシステム TECS 仕様書, http://www.toppers.jp/tecs.html.

[44] nxtOSEK: http://lejos-osek.sourceforge.net/.

[45] OMG: A UML Profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE), http://www.omgmarte.org/.

[46] OMG: CORBA Component Model 4.0, http://www.omg.org/spec/CCM/4.0/.

[47] OMG: OMG Systems Modeling Language (OMG SysML), 2007.

[48] OSEK/VDX 仕様: http://www.osek-vdx.org/.

[49] Renesas Electronics Corporation: *SH72AW Group Hardware Manual*.

[50] Renesas Electronics Corporation: *V850E2/MN4 Hardware User's Manual*.

[51] RTCA, Inc.: *RTCA/DO-178C Software Considerations in Airborne Systems and Equipment Certification*, 2011.

[52] SAE International: Architecture Analysis & Design Language (AADL). AS5506A, 2009.

[53] Shiraishi, S. and Abe, M.: Automotive System Development Based on Collaborative Modeling Using Multiple ADLs, *Proceedings of ESEC/FSE 2011*, Sep. 2011.

[54] Taherkordi, A., Loiret, F., Abdolrazaghi, A., Rouvoy, R., Le-Trung, Q., and Eliassen, F.: Programming sensor networks using REMORA component model, *Proceedings of the 6th IEEE international conference on Distributed Computing in Sensor Systems*, 2010, pp. 45–62.

[55] Taherkordi, A., Loiret, F., Rouvoy, R., and Eliassen, F.: A Generic Component-Based Approach for Programming, Composing and Tuning Sensor Software, *The Computer Journal*, Vol. 54, No. 8(2011), pp. 1248–1266.

[56] TEXAS INSTRUMENTS: *Stellaris LM3S6965 Microcontroller DATA SHEET*.

[57] TEXAS INSTRUMENTS: *TMS570LS Series Technical Reference Manual*.

[58] TOPPERS: TOPPERS/ASP カーネル, http://www.toppers.jp/asp-kernel.html.

[59] TOPPERS: TOPPERS/JSP プラットフォーム, http://www.toppers.jp/etrobo-jsp.html.

[60] TOPPERS Project, Inc.: TOPPERS 新世代カーネル統合仕様書 Release 1.5.0, 2012.

[61] TRON Association: Protection Extention of μITRON4.0 Specification, 2002.

[62] Zeng, H.: Probabilistic Timing Analysis of Distributed Real-time Automotive Systems, Technical report, University of California, Berkeley, 2008.

[63] Zeng, H., Natale, M. D., Giusto, P., and Sangiovanni-Vincentelli, A.: Stochastic Analysis of CAN-Based Real-Time Automotive Systems, *IEEE Transactions on Industrial Informatics*, Vol. 5, No. 4(2009), pp. 388–401.

[64] μ ITRON4.0 仕様: http://www.ertl.jp/ITRON/SPEC/mitron4-j.html.

[65] 安積卓也, 山本将也, 小南靖雄, 高木信尚, 鵜飼敬幸, 大山博司, 高田広章: 組込みシステムに適したコンポーネントシステムの実現と評価, コンピュータソフトウェア, Vol. 26, No. 4(2009), pp. 39–55.

[66] 水野匡章, 佐藤秀夫: 10-217 Lego Mindstorms によるリアルタイム組込みシステムの理論と実践の教育 (オーガナイズドセッション「企業における技術者継続教育」), 工学・工業教育研究講演会講演論文集, Vol. 17(2005), pp. 530–531.

[67] 経済産業省: 2010 年版組込みソフトウェア産業実態調査報告書, p.24, ”http://www.meti.go.jp/policy/mono_info_service/joho/downloadfiles/2010software_research/10project_houkokusyo.pdf”.

[68] 原拓, 石川拓也, 大山博司, 高田広章: 組込み向け TCP/IP プロトコルスタックのコンポーネント設計, 情報処理学会　第 *26* 回組込みシステム研究発表会, 福岡, Sep. 2012.

[69] 後藤隼弐, 本田晋也, 長尾卓哉, 高田広章: トレースログ可視化ツール TraceLogVisualizer(TLV), コンピュータソフトウェア, Vol. 27, No. 4(2010), pp. 8–23.

[70] 松原豊, 高田広章: 組込みシステムにおけるソフトウェア障害の安全対策, Vol. 30, No. 1(2013), pp. 119–129.

[71] 石川拓也, 安積卓也, 一場利幸, 柴田誠也, 本田晋也, 高田広章: TECS 仕様に基づいた NXT 用ソフトウェアプラットフォームの開発, コンピュータソフトウェア, Vol. 28, No. 4(2011), pp. 158–174.

[72] 石川拓也, 安積卓也, 一場利幸, 柴田誠也, 高田広章: UML モデルの C 言語実装における TECS の適用事例, 情報処理学会第 *78* 回プログラミング研究会, Mar. 2010.

[73] 湯淺太一: Lego MindStorms 用の Lisp 処理系 XS, コンピュータソフトウェア, Vol. 24, No. 4(2007), pp. 51–65.

[74] 西部満, 本田晋也, 冨山宏之, 高田広章: ハードリアルタイムシステムに適したメモリ保護機構の提案と評価, 情報処理学会研究報告, Vol. 2005, No. 27(2005), pp. 115–120.

[75] 志子田有光, 加藤和夫, 菅原研, 松澤茂, 河田拓朗, 川田徳明, 井口巌, 佐藤徳男, 佐々木整: 高大連携による組み込み教材開発と高大生交流授業モデルの実践, 日本教育工学会論文誌, Vol. 32, No. 2(2008), pp. 141–148.

研究業績

主論文に関連する研究業績

査読付きの学術雑誌論文

1. 石川拓也，安積卓也，一場利幸，柴田誠也，本田晋也，高田広章，“TECS 仕様に基づいた NXT 用ソフトウェアプラットフォームの開発”，コンピュータソフトウェア，Vol.28，No.4，pp. 158-174，Nov. 2011．

2. 石川拓也，松原豊，高田広章，“周期タスクの初期位相分布を考慮した応答時間の確率的解析”，情報処理学会論文誌，Vol.52，No.12，pp. 3192-3204，Dec. 2011．

3. 石川拓也，本田晋也，高田広章，“静的なメモリ配置を行うメモリ保護機能を持ったリアルタイム OS”，コンピュータソフトウェア，Vol.29，No.4，pp. 161-181，Nov. 2012．

査読付きの国際会議論文

1. Takuya Ishikawa, Takuya Azumi, Hiroshi Oyama and Hiroaki Takada, ”HR-TECS: Component Technology for Embedded Systems with Memory Protection”, The 16th IEEE International Symposium on Object/component/service-oriented Real-time dis-

tributed computing (ISORC 2013), Paderborn, Germany, June 2013.

学会での口頭発表

1. 石川拓也，松原豊，高田広章，“分散リアルタイムシステムの端点間処理における応答時間の確率的解析”， 情報処理学会 第 18 回組込みシステム研究発表会，函館市， Aug. 2010.

2. 石川拓也，本田晋也，高田広章，“MPU を用いたメモリ保護機能対応リアルタイム OS”， 第 12 回 組込みシステム技術に関するサマーワークショップ（SWEST12），豊橋市，Sep. 2010.

3. 石川拓也，本田晋也，高田広章，“TOPPERS/HRP2 カーネルによるメモリ保護ユニットの抽象化”，第 15 回 組込みシステム技術に関するサマーワークショップ（SWEST15），下呂市，Aug. 2013.

その他の研究業績

査読付きの学術雑誌論文

なし

査読付きの国際会議論文

1. Takuya Ishikawa, Toshikazu Kato, Shinya Honda and Hiroaki Takada, ”Investigation and Improvement on the Impact of TLB misses in Real-Time Systems”, Proceedings of the 9th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT2013), pp. 6-10, Paris, France, July 2013.

学会での口頭発表

1. 石川拓也，安積卓也，一場利幸，柴田誠也，高田広章，“UML モデルの C 言語実装における TECS の適用事例”，情報処理学会 第 78 回プログラミング研究会，調布市，Mar. 2010.

2. 山口英之，安積卓也，石川拓也，小南靖雄，鵜飼敬幸，高木信尚，大山博司，高田広章，“組込みコンポーネントシステム TECS と TINET を用いた RPC 機構”，第 12 回組込みシステム技術に関するサマーワークショップ（SWEST12），豊橋市，Sep. 2010.

3. 柴田誠也，石川拓也，本田晋也，冨山宏之，高田広章，“バス調停の遅延時間見積もりのための確率的数学モデル”，情報処理学会研究報告，Mar. 2011.

4. 柴田誠也，石川拓也，本田晋也，冨山宏之，高田広章，“上流設計におけるバス調停遅延時間の見積もりのための確率的数学モデル”，DA シンポジウム 2011，pp. 159-164，Sep. 2011.

5. 加藤寿和，高瀬英希，石川拓也，本田晋也，高田広章，“リアルタイムシステムに向いたコア間通信機構の検討”，第 13 回 組込みシステム技術に関するサマーワークショップ（SWEST13），下呂市，Sep. 2011.

6. 大野惇，鵜飼敬幸，石川拓也，安積卓也，西尾信彦，“OSEK と ITRON の TECS を用いた共通開発機構”，第 13 回 組込みシステム技術に関するサマーワークショップ（SWEST13），下呂市，Sep. 2011.

7. 原拓，石川拓也，大山博司，高田広章，“組込み向け TCP/IP プロトコルスタックのコンポーネント設計”，情報処理学会　第 26 回組込みシステム研究発表会，福岡，Sep. 2012.

8. 加藤寿和，石川拓也，本田晋也，高田広章，“リアルタイムシステムにおけるTLB ミスの影響調査と改善手法”，組込み技術とネットワークに関するワークショップ ETNET2013，対馬，Mar. 2013.

商業雑誌等

1. 安積卓也，石川拓也，“コンポーネントで組み上げる　MINDSTORMS NXT 用プラットホーム TOPPERS/ASP+TECS を使いこなす”，CQ 出版社，Interface2011 年 7 月号，pp. 147-157，July 2011.

受賞等

1. 第 6 回 TOPPERS of the Year，“ET ロボコンへの協力”，竹内良輔（教育 WG 主査／(株) リコー），石川拓也（名古屋大学），June 2010.

2. 第 8 回 TOPPERS of the Year，“ET ロボコンへの協力”，竹内良輔（教育 WG 主査／(株) リコー），石川拓也（名古屋大学），June 2012.