リアルタイムアプリケーション向け タスク処理定義可能なスケジューリングシミュレータ

佐野泰正† 松原豊†
本田晋也† 高田広章†

様々なタスクスケジューリングアルゴリズムを用いて，既存のリアルタイムアプリーションをスケジュールしたときの振る舞いを，容易に確認することを目的に，スケジューリングに特化したシミュレータが開発されている．既存のシミュレータでは，複雑なアプリケーションの振る舞いを正確にモデル化することが困難であるため，シミュレーション結果が，実機での動作と大きく異なるという問題がある．本論文では，タスク処理を詳細に定義可能なアプリケーションモデルと，そのモデルを扱うシミュレータを提案する．モデリング性能を評価するため，実際のエンジン制御アプリケーションのモデルを作成し，タスクの応答時間を測定した．その結果，エンジン制御アプリケーションの実行ファイルを命令セットシミュレータ上で実行した結果に対して，タスク応答時間の平均誤差を 10.5%に抑えることができた．

A Scheduling Simulator with Task Modeling Capabilities for Real-time Application

Yasumasa Sano,† Yutaka Matsubara,† Shinya Honda† and Hiroaki Takada†

There are several real-time scheduling simulators to verify the behavior of real-time applications under different task scheduling algorithms. Current simulators cannot model the application accurately and, consequently the results of the simulation differ considerably from the actual behavior on a real computing system. This paper presents a scheduling simulator with task modeling capabilities for real-time applications. The proposed approach supports modeling of complex task control flows and dependency relations between tasks. In order to evaluate the modeling capabilities, we modeled a real engine control application and simulated it. We measured the average response times of the application model running on our scheduling simulator and compared them with the ones obtained by running the real engine application binaries on an instruction set simulator. The average of the percentage error between both simulations was only 10.5%.

1. はじめに

これまでに，リアルタイムシステム向けのタスクスケジューリングアルゴリズムが数多く提案されている．これらの多くは，理論的なアルゴリズムの提案に留まっており，既存のリアルタイム OS（RTOS と呼ぶ）に実装され，広く公開されているものは少ない．そのため，RTOS 上で動作するアプリケーションを様々なスケジューリングアルゴリズムで動作させた結果を比較，検証するためには，対象となるスケジューリングアルゴリズムを RTOS のスケジューラに実装する必要がある．しかしながら，RTOS のスケジューラを変更することは，RTOS の内部構造を詳細に把握することと，ハードウェアに依存する高度なプログラミング能力を要求されるため，容易ではない．独自に開発したスケジューリングアルゴリズムの動作を検証する場合も同様の問題がある．

複数のスケジューリングアルゴリズムの中から，応答時間やリアルタイム性などのアプリケーションの要求を満たすアルゴリズムを選択すること，あるいは，新規に開発したスケジューリングアルゴリズムの動作を検証することなどを目的として，スケジューリングアルゴリズムに特化したシミュレータ（スケジューリングシミュレータと呼ぶ）が開発されている．スケジューリングシミュレータは，アプリケーションのモ

† 名古屋大学 大学院情報科学研究科
Graduate School of Information Science,Nagoya University.

デルを入力すると，選択したスケジューリングアルゴリズムでスケジュールした結果を出力するツールである．

既存のスケジューリングシミュレータには，入力するアプリケーションモデルに，実際のアプリケーションの動作を詳細に記述できないという問題がある．具体的には，タスクの実行時間や起動周期などを固定値とする場合が多く，例えば，条件分岐により，実行時間が大きく変動するタスクを正確にモデル化することができない．このような場合には，変動するパラメータを平均値や最悪値に固定したモデルを用いるが，シミュレーション結果は，実機上での動作と大きく異なる．

本論文では，この問題を解決するために，タスク処理定義可能なスケジューリングシミュレータ *schesim* を中心とする，スクリプト言語 Ruby を用いたオープンソースのスケジューリングシミュレーション環境☆を提案する．*schesim* の主な特徴は，処理内容を詳細に定義可能なタスクモデルを導入することで，実際のアプリケーションを，既存のスケジューリングシミュレータで扱うモデルに比べて，詳細にモデル化できることである．加えて，RTOS の API と同等の機能を実現する関数群や，外部割込みやメッセージ受信などの非周期的に発生するイベントを定義する機能を持つ．標準でサポートしているスケジューリングアルゴリズムは，シングルプロセッサ向けスケジューリングアルゴリズムだけでなく，マルチプロセッサ向けスケジューリングや，階層型スケジューリングがある．さらに，シミュレーション結果を可視化するツールとの連携や，タスクの実行時間や応答時間などを容易に解析するツールを標準で用意している．以上により，スケジューリングアルゴリズム開発者やアプリケーション開発者に対して，自由に利用できる有用性の高いシミュレーション環境を提供する．

本論文の構成は，以下の通りである．まず，2 章で，スジューリングシミュレータに対する機能要件を整理する．3 章では，関連研究を述べる．4 章では，*schesim* 及び補助ツールについて詳細に述べる．5 章では，現実のエンジン制御アプリケーションを対象に，モデリング性能の異なるシミュレータを想定したアプリケーションモデルを作成し，シミュレーションした結果から，シミュレータのモデリング能力について述べる．最後に，6 章で，本論文のまとめと，今後の課題について述べる．

☆ http://www.schesim.org/

2. スケジューリングシミュレータの要件

本章では，スケジューリングシミュレータに必要な要件を大きく 6 つに整理する．

アプリケーションモデルの記述に関する要求事項

（要件 1）タスク処理の記述に関する要件
（要件 2）タスク間の依存関係の記述に関する要件
（要件 3）非同期処理の記述に関する要件

シミュレータに関する要求事項

（要件 4）スケジューリングアルゴリズムの実装に関する要件

シミュレータに付随するツールに関する要求事項

（要件 5）アプリケーションモデルの作成に関する要件
（要件 6）シミュレーション結果の解析に関する要件

要件 1 を満たすためには，大きく分けて 2 つの条件がある．1 つは，タスクの処理内容を抽象化できることである．具体的には，アプリケーションと外部装置との通信内容や，計算処理のアルゴリズムといった具体的な処理を，処理時間を指定することで隠蔽できることである．もう 1 つは，タスクの処理をプログラム形式で記述できることである．具体的には，タスクの処理に条件分岐，繰り返し処理を使用することや，タスクの処理内に内部変数を定義できることである．

要件 2 は，あるタスクに依存関係を持つタスクの動作を，より正確にモデル化するための要件である．この要件を満たすためには，タスク間の共有変数，タスクを起動する機能，タスク間の排他制御を実現する機能を持つ必要がある．

要件 3 は，発生時刻や発生間隔に周期性のないイベント（タスク起動，外部割込み，メッセージ受信など）をより正確にモデル化するための要件である．具体的には，絶対時刻，相対時刻を指定して API を呼び出せることや，指定する相対時刻を確率分布に従う乱数で生成できることである．

要件 4 を満たすためには，大きく分けて 3 つの条件がある．1 つ目は，シングルプロセッサ向けの基本的なスケジューリングアルゴリズムが実装されていることである．具体的には，FPS（Fixed-Priority Scheduling）や，EDFS（Earliest Deadline First Scheduling）が実装されていることである．2 つ目は，マルチプロセッサ向けスケジューリングアルゴリズムが実装されていることである．具体的には，AMP 型マルチプロセッサアーキテクチャや，SMP 型マルチプロセッサアーキテクチャに関するスケジューリングがサポートされていることである．3 つ目は，ユーザ定義のスケジューリングアルゴリズムを容易に実装できることである．

表 1 スケジューリングシミュレータの比較表
Table 1 A Comparative Table of Scheduling Simulator

シミュレータ名	要件 1	要件 2	要件 3	要件 4	要件 5	要件 6
FORTISSIMO	△	-	○	○	×	○
AURTSS	△	×	×	△	○	○
Realtss	△	×	×	△	○	○
RTsim	△	×	○	○	-	○
ARTISST	○	×	○	○	-	○
RTSSim	○	○	○	×	-	○
RTSIM	△	△	○	△	-	○
STRESS	○	○	×	△	×	△
schesim	○	○	○	△	○	○

△は一部満たすもの，-は論文中で言及されていないものを表す

この条件を満たすためには，ユーザ定義のアルゴリズムを容易に追加できる仕組みをもつか，シミュレータのソースコードが公開されていることが必要である．

要件 5 は，シミュレーションするアプリケーションのモデルを容易に作成するための要件である．この要件を満たすためには，アプリケーションモデルを容易に入力できるインターフェースや，アプリケーションモデル自体を自動的に生成する機能が必要である．

要件 6 は，シミュレータの出力するシミュレーション結果を容易に利用するための要件である．この要件を満たすためには，シミュレーションしたアプリケーションの動作を可視化する機能や，シミュレーション結果からタスクの応答時間やデッドラインミス回数などの情報を解析する機能が必要である．

3. 関 連 研 究

本章では，2 章で整理した要件を踏まえて，既存のスケジューリングシミュレータの持つ機能を表 1 に示す．

FORTISSIMO[1] は，C++で実装された，シミュレータのオープンフレームワークで，SMP 型マルチプロセッサアーキテクチャを想定したシミュレーションができることが特徴である．AURTSS[2] は，Web ブラウザから利用可能なリアルタイムスケジューリングシミュレータである．インターネットに接続できるコンピュータがあれば，特別なソフトウェアをインストールする必要がないことや，GUI でパラメータを容易に入力することができる利点がある．Realtss[3] は，オープンソースのリアルタイムスケジューリングシミュレータである．特徴は，スクリプト言語 TCL を用いて新しいスケジューリングアルゴリズムを定義できることや，サポートされているスケジューリングアルゴリズムの多いことなどである．RTsim[4] は，リアルタイムスケジューリングアルゴリズムの学習ツールとして開発されたシミュレータである．教育を目的としているため，サポートされているスケジューリングアルゴリズムが豊富である．これらの 4 つのシミュレータは，タスクの実行時間を指定できるが，タスク内部の処理をプログラム形式で書けないという欠点がある．そのため，要件 1 を十分満たしているとは言えない．

ARTISST[5] は，プロセッサ時間を指定する関数や，タスクの処理に制御構文を記述できるシミュレータである．そのため，要件 1 を満たすことができる．しかし，タスク処理中に，別タスクの起動や，タスク間の排他制御などを実現する API が用意されていないため，要件 2 を満たすことができない．

RTSIM[6] は，オープンソースであるため，シミュレータを拡張することも可能である[7]．RTSIM のタスク処理記述は，実行したい命令列を列挙する方法である．しかし，処理内での条件分岐や，他のタスクとの共有変数を扱うことができないという，制限がある．さらに，絶対時刻を指定したイベント発生機能や，マルチプロセッサ環境に対応していないという問題があるため，要件 1，2，4 を十分満たしているとは言えない．

RTSSim[8] は，要件 1，2 を満たすことができるスケジューリングシミュレータである．しかし，オープンソースではないため，拡張性が低い．また，マルチプロセッサアーキテクチャを想定したシミュレーションに対応していないという問題がある．そのため，要件 4 を満たしているとは言えない．

STRESS[9] は，STRESS 言語というタスク処理記述のための独自言語で作成したモデルをシミュレーションする．STRESS 言語を用いると，タスクの処理を記述するだけでなく，同じ言語でスケジューラを実装することも可能である．しかし，絶対時刻，もしくは相対時刻を指定した処理を記述できないという制約があるため，要件 3 を満たすことができない．

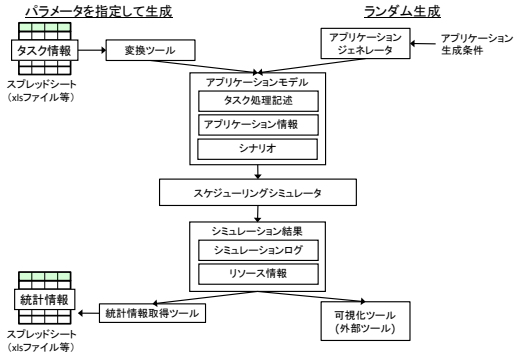

図 1 シミュレーションのフロー図
Fig. 1 Simulation flow diagram.

4. スケジューリングシミュレータ *schesim*

4.1 *schesim* の概要

2 章の要件に対応する，*schesim* の機能概要について述べる．要件 1，2 は，タスク処理記述ファイルにより実現する．要件 3 は，シナリオファイルにより実現する．要件 4 は，シングルプロセッサ向けスケジューリングアルゴリズムについては，FPS と EDFS を標準でサポートしている．マルチプロセッサ向けスケジューリングアルゴリズムについては，AMP 型マルチプロセッサアーキテクチャを標準でサポートしている．現時点では SMP 型マルチプロセッサアーキテクチャはサポートしていない．*schesim* はオープンソースのシミュレータのため，ユーザ定義のスケジューリングアルゴリズムを実装することもできる．要件 5 は，モデル作成を支援するツールを提供することで実現する．要件 6 は，シミュレーション結果を可視化するツールと連携することと，統計情報を取得できるツールを用意することで実現する．

4.2 シミュレーションの流れ

schesim でのシミュレーションの流れを図 1 に示す．まず，ユーザは，アプリケーションのモデルを作成する．アプリケーションモデルは，アプリケーション情報ファイル，タスク処理記述ファイル，シナリオファイルの 3 つで構成される（ただし，シナリオファイルは必須ではない）．アプリケーション情報ファイルとタスク処理記述ファイルは 2 つの支援ツールのいずれかを用いて作成することができる．1 つ目は，スプレッドシート形式のファイルにタスクのパラメータを入力し，生成する方法である．2 つ目は，タスクのパラメータを乱数を用いて決定し，生成する方法である．アプリケーションモデルをシミュレータに入力すると，アプリケーション情報ファイルに記述したスケジューリングアルゴリズムをシミュレーションし，アプリケーションのスケジュール結果を出力する．シミュレーション結果は，シミュレーションログとリソース情報で構成される．シミュレーションログとリソース情報を可視化ツールに入力することで，シミュレーション時のタスクの切り替えや，API 呼び出しの振る舞いを可視化することができる．また，シミュレーションログを統計情報取得ツールに入力することで，タスクの実行時間や応答時間などの情報を得ることができる．

```
"task": [{
  "period": 36,        /* 起動周期 */
  "priority": 1,       /* 優先度 */
  "id": 1,             /* タスク ID */
  "deadline": 36,      /* 相対デッドライン */
  "attr": "cyclic",    /* 起動属性 */
  "offset": 5}],       /* 初期起動オフセット */
```

図 2 アプリケーション情報ファイルでのタスク定義の例
Fig. 2 Example of Task Model in Application Information File.

4.3 アプリケーションモデル

4.3.1 アプリケーション情報ファイル

アプリケーション情報ファイルは，システムのプロセッサコア数や，アプリケーションの構成，スケジューリング方式などを JSON（JavaScript Object Notation）形式で記述したテキストファイルである．

設定できるパラメータ情報は，コアについては，そのコアで動作するアプリケーションをスケジューリングするアルゴリズム（グローバルスケジューリングアルゴリズム）とコア ID である．アプリケーションについては，所属するタスクをスケジューリングするアルゴリズム（ローカルスケジューリングアルゴリズム），プロセッサ利用率，起動周期，アプリケーション ID，優先度である．タスクについては，起動周期，優先度，タスク ID，相対デッドライン，起動属性，初期起動オフセットである．起動属性は，周期，非周期，離散から選択できる．図 2 に，アプリケーション情報ファイルにおけるタスク情報の記述例を示す．

選択できるスケジューリングアルゴリズムについて述べる．シングルプロセッサ向けスケジューリングアルゴリズムとして，FPS，EDFS がある．FPS での優先度の決め方は RM（Rate Monotonic），DM（Deadline Monotonic），ランダムから選択できる．階層型スケジューリングアルゴリズムとしては，BSSA（Bandwidth Sharing Server Algorithm）[10]，TPA（Temporal Protection Algorithm）[11] を選択できる．マルチプロセッサ向けスケジューリングアルゴリズムとしては，各プロセッサコア毎にシングルプロセッサ向け

```
 1 # リソース 1 の定義
 2 @@res1 = RESOURCE.new(1)
 3 @@mode1 = 0 # モード変数の定義
 4 def task1 # タスク 1 のタスク処理記述
 5       if @@mode1 == 0
 6           act_tsk(2) # タスク 2 を起動
 7           @@mode1 = 1
 8       end
 9       GetResource(@@res1) # リソース 1 を獲得
10       exc(1) # 1 単位時間実行
11       ReleaseResource(@@res1) #リソース 1 を解放
12       exc(1) # 1 単位時間実行
13 end
```

図 3 タスク処理記述ファイルの例
Fig. 3 Example of Task Modeling File.

スケジューリングアルゴリズムを選択できる．すなわち，AMP 型マルチプロセッサアーキテクチャのみを想定している．

4.3.2 タスク処理記述ファイル

タスク処理記述ファイルには，個々のタスクの処理内容を Ruby の関数定義と同様の方法で記述する．内部変数や，if 文，for 文といった Ruby の制御構文に加えて，API として以下の関数を呼び出すことができる．

- exc：指定した単位時間分のプロセッサ時間を消費する
- act_tsk：指定した ID を持つタスクを起動する（μITRON 仕様[12] 準拠）
- wai_sem/sig_sem：セマフォを獲得，解放する（μITRON 仕様準拠）
- GetResource/ReleaseResource：リソースを獲得，解放する（OSEK OS 仕様[13] 準拠）
- SetEvent/WaitEvent/ClearEvent：イベントをセット，待つ，クリアする（OSEK OS 仕様準拠）

exc で指定するプロセッサ時間の決め方は，例えば，実際のアプリケーションに対して，最悪実行時間解析ツール，命令セットシミュレータ（ISS と呼ぶ），実機などを使用して実行時間を計測する方法が考えられる．

図 3 にタスク処理記述ファイルの例を示す．この例では，タスク 1 の処理内容を定義している．2 行目で，アプリケーションで使用するリソースを定義する．4 行目から 13 行目までがタスク 1 の処理内容である．5 行目から 8 行目は，内部変数 mode1 が 0 のときのみ，タスク ID が 2 のタスクを起動するという処理をする．9 行目では，リソース res1 を獲得し，11 行目でリソース res1 を解放する．その間に，10 行目でプロセッサ時間 1 を消費する．12 行目でプロセッサ時間 1 を消費し，処理を終了する．*schesim* では，このように柔軟な処理記述方法により，RTOS 上で動作するアプリケーションを，より正確にモデル化できる．

```
1 12:1:chg_mod:@@mode2:2
2 15:1:act_tsk:4
```

図 4 シナリオファイルの例
Fig. 4 Example of Scenario File.

```
1 [100]:[1]: applog strtask : TASK 1 :sig_sem()
2 [110]:[1]: budget of application 1 is 10.
3 [110]:[1]: task 1 becomes DORMANT.
4 [110]:[1]: application 1 becomes EXPIRED.
```

図 5 シミュレーションログの例
Fig. 5 Example of Simulation Log.

4.3.3 シナリオファイル

リアルタイムアプリケーションには，割込みのように非同期に発生するイベントや，外部の環境に応じて処理が変化するタスクが存在する場合が多い．これらをモデル化するために，シナリオファイルを用いる．図 4 にシナリオファイルの例を示す．左から順に，イベントの発生時刻，コア ID，発生するイベント，イベントへの引数を表している．例えば，図 4 の 1 行目では，時刻 12 で mode2 の値を 2 に変更するという処理を発生させることができる．イベントとしては，4.3.2 項で挙げた関数 “act_tsk” に加えて，タスクの内部変数の値を変更する “chg_mod” を使用できる．

4.4 *schesim* によるシミュレーション

schesim は，スクリプト言語 Ruby で実装したシミュレータであり，Ruby1.8.7 以降で動作する．

シミュレータ内部では離散的なシステム時刻を管理し，システム時刻を更新するタイミングでタスクスケジューリングや API の処理を実行する．例えば，実行中のタスクが exc(2) を呼び出すと，呼び出した時刻からシステム時刻が 2 単位時間経過した後に，次の処理を実行する．その他の関数や制御構文は，同一のシステム時刻内で処理する．すなわち，タスクスケジューリングや，API 実行時間は 0 と考える．

シミュレーションが終了すると，シミュレーションログとリソースファイルが出力される．シミュレーションログは，タスクの状態遷移や API 呼び出しの履歴を時系列で出力したものである．図 5 に，シミュレーションログの例を示す．左から順に，システム時刻，コア ID，履歴を表す．このフォーマットは，TOPPERS/FMP カーネル[14] が出力するログと同様であるが，階層型スケジューリングをシミュレーションした場合のための独自の情報として，アプリケーションの状態遷移や，バジェットの残量の変化を表すログを追加している．1

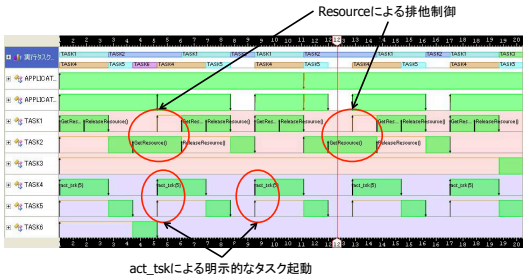

図 6 TLV によるシミュレーション結果の可視化
Fig. 6 Example of TLV Screenshot.

行目は task1 が sig_sem() を呼び出した履歴で，3 行目は，task1 が DORMANT 状態に遷移した履歴である．2 行目は，アプリケーションのバジェットの残量の変化を表す履歴で，具体的には，application1 のバジェットの残量が 10 になったことを表す．4 行目は，application1 がバジェットを使いきり，EXPIRED 状態に遷移した履歴である．リソースファイルは，可視化ツールの入力として必要なファイルであり，アプリケーション情報ファイルから自動で生成される．

4.5 外部ツール

4.5.1 シミュレーション結果の可視化ツール

シミュレーション結果を可視化するツールとして，TLV（Trace Log Visualizer）[15] を使用できる．TLV でシミュレーション結果を可視化するためには，*schesim* によって出力されたシミュレーションログとリソースファイルを TLV に入力する．図 6 に，TLV によるシミュレーション結果の可視化例を示す．

4.5.2 統計情報取得ツール

統計情報取得ツールは，シミュレーションログを入力することで，シミュレーションの統計情報をスプレッドシート形式のファイルで出力する．取得できる統計情報は以下の通りである．

- タスク毎の情報：起動回数，実行時間（平均，合計，最大，最小），応答時間（平均，最大，最小），CPU 使用率
- システム全体の情報：タスク数，CPU 使用率

5. エンジン制御アプリケーションへの適用

schesim のモデリング性能の評価として，他のスケジューリングシミュレータで採用されているアプリケーションモデルと比較をした．対象とするアプリケーションは実際のエンジン制御アプリケーションである．比較対象とするアプリケーションモデルは，2 種類である．1 つは，タスク内部の処理が定義できず，実行時間のみ指定できるモデル（シンプルモデルと呼ぶ）である．もう 1 つは，タスク内部の処理は定義できるが，制御構文が使用できない RTSIM のモデル（RTSIM モデルと呼ぶ）である．

5.1 エンジン制御アプリケーションの特徴

エンジン制御アプリケーションは，タスク数が 63 個，割込み処理の数が 62 個で構成されており，RTOS 上で動作する．タスクは FPS でスケジューリングされる．エンジン制御アプリケーションには，エンジン内部のクランクと呼ばれる部品の回転角度によって，タスクの実行時間や，割込み処理の発生時刻が変化する処理が存在する．このように複雑なアプリケーションを既存のスケジューリングシミュレータでシミュレーションすると，起動間隔や実行時間の変動をモデルに反映できないため，実機上での動作と大きく異なってしまう可能性がある．

5.2 エンジン制御アプリケーションのモデル化

今回は，ソースコードの入手が困難であるため，エンジン制御アプリケーションの実行ファイルを ISS 上で実行した結果から，タスク，割込み処理に関する情報を得て，アプリケーションモデルを作成した．タスク，割込み処理の実行時間は数 μs から千数百 μs であることから，シミュレータのシステム時刻の単位は μs と設定した．また，*schesim* には，割込み処理の概念が無いため，割込み処理は最高優先度のタスクとして扱う．

シンプルモデルは，周期，実行時間などのパラメータが全て固定値のモデルである．そこで，タスク処理記述ファイルにおいて，制御構文や RTOS の API を使わず，exc() のみを記述する．実際のエンジン制御アプリケーションでは，タスク内で他タスクを起動するという処理や，イベントにより待ち，待ち解除をするというような同期処理が存在する．しかし，シンプルモデルの制約上，このような処理が定義できないため，全てのタスクを，決められた時間毎に自動起動する周期タスクとして扱う．周期，実行時間は，ISS でのシミュレーション結果から計測した平均値を採用する．

RTSIM が扱うモデルは，タスクの処理内容に制御構文，タスク内変数，タスク間共有変数は定義できないが，RTOS の API は定義できるモデルである．実行時間は，シンプルモデルと同様に固定値であるため，シンプルモデルに “act_tsk” を追加することによって作成する．クランクの回転に同期する割込み処理に関しては，発生間隔がクランクの回転角に依存しているため，RTSIM モデルの制約では実現することができない．よって，クランクの回転に同期する割込み処理は，先ほどと同様に，全て ISS のシミュレーション結

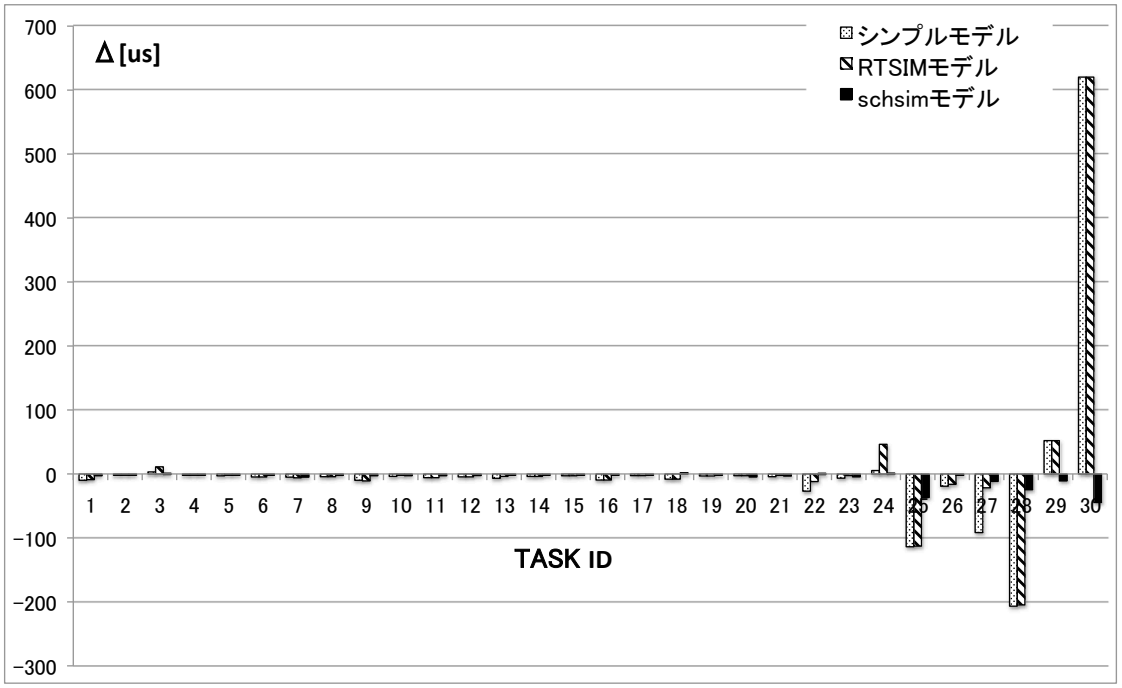

図 7 ISS での動作と，*schesim* での動作のタスク平均応答時間の差
Fig. 7 Error of Average Response Time between ISS and *schesim* simulations.

果から計測した周期の平均値で発生することとする．

schesim のモデル（schesim モデルと呼ぶ）では，4.3 節で述べた機能を使いモデルを作成する．タスク，割込み処理の実行時間が規則的に大きく変化する場合は，条件分岐やタスク内部変数によりモデル化した．不規則的に数十 μs ほど大きく実行時間が変化する場合は，不規則的に変化する実行時間のデータをタスク処理記述内の配列で宣言し，その配列から実行時間を決定するようにした．ただし，その規則性の中にも実行時間に数 μs 程度ぶれがある．このような細かいぶれはモデルには反映せず，平均値を用いた．割込み禁止区間による割込み処理の遅延，API 実行時間，タスクの切り替えや，スケジューリング等の OS の処理時間はタスクの実行時間の比べて非常に短いため，モデルに反映していない．クランクの回転に同期する割込み処理は，クランクの動作を模擬する仮想のタスクを作成し，そのタスクから，割込み処理を発生させるようにした．

5.3 評　　価

モデリング性能を評価するため，エンジン制御アプリケーションの実行ファイルを ISS 上で実行した結果と，*schesim* でシンプルモデル，RTSIM モデル，schesim モデルをシミュレーションした結果の，タスクの平均応答時間を比較した．今回は，各モデルを 167,000 単位時間分シミュレーションした結果を用いた．シミュレーション環境は，以下の通りである．

- OS：Mac OS X 10.6.7
- プロセッサ：2.13GHz Intel Core 2 Duo
- メモリ：2GB 1067MHz DDR3
- Ruby：ruby 1.8.7

図 7 に，ISS でのタスク毎の平均応答時間（R_{ISS} とする）と，各モデルでのタスク毎の平均応答時間の差（Δ とする）を示す．図 8 に，R_{ISS} に対する，Δ の割合を示す．表 2 は，モデル毎の Δ の絶対値の最大値と平均値，R_{ISS} に対する Δ の絶対値の割合の最大値と平均値を示す．

本論文で対象としたエンジン制御アプリケーションは，TASK ID が小さいタスクほど優先度が高くなる．

表 2 シミュレーション結果のまとめ
Table 2 Result of Simulation

モデル名	平均応答時間の差の絶対値 [μs]		差の絶対値の割合 [%]	
	最大値	平均値	最大値	平均値
シンプルモデル	620	41	82.8	25.7
RTSIM モデル	620	40	119.8	25.2
schesim モデル	45	6	36.5	10.5

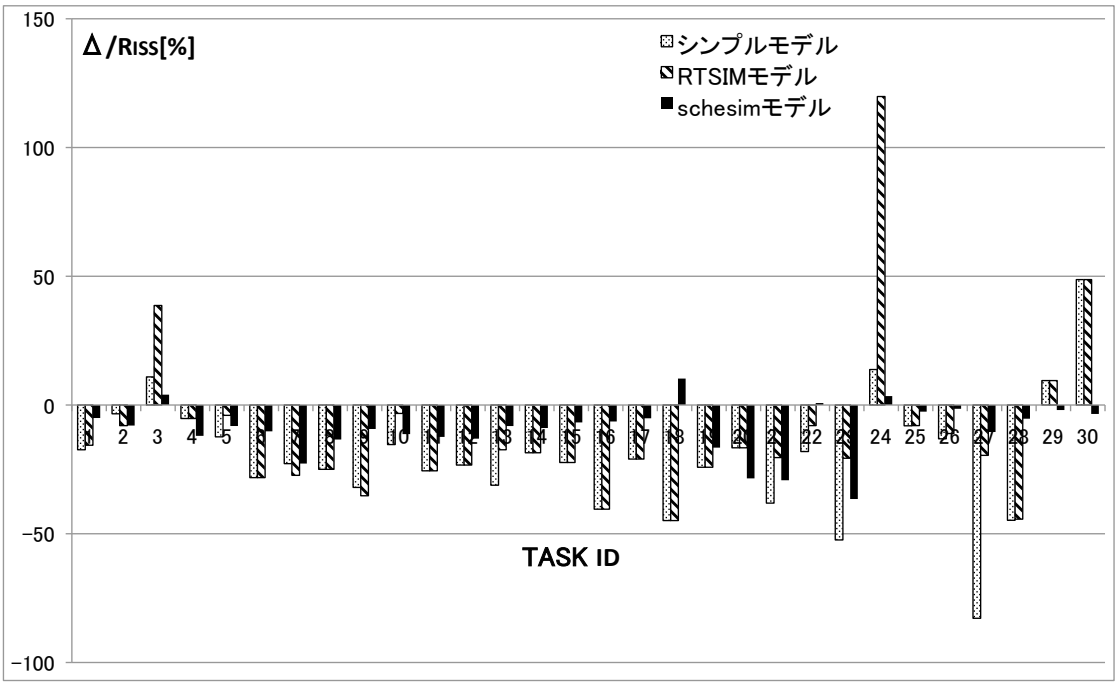

図 8 ISS での動作と，*schesim* での動作のタスク平均応答時間の差の割合
Fig. 8 Percentage Error of Average Response Time between ISS and *schesim* simulations.

図 7 から，優先度が高いタスクは，どのモデルでも ISS のシミュレーション結果とほとんど差は無い．この理由は，タスクが邪魔されることなく，応答時間のほとんどが自身の実行時間によるものだからである．一方，優先度が低いタスクでは，シンプルモデル，RTSIM モデル共に大きな差が出ている．この理由は，タスク，割込み処理の起動タイミングが，ISS でのシミュレーション結果と異なっており，低優先度タスクを邪魔する高優先度タスクや割込み処理が異なっているためである．これでは，タスクの実行順序が大きく異なっているため，正確にモデル化できているとは言い難い．図 8 についても同様のことが言える．

これに対して，schesim モデルでは，表 2 より，平均応答時間の最大誤差は 45μs となったが，このタスクは図 8 の割合でみると 3.5%程度である．また，割合でみたときの最大誤差は 36.5%であるが，このタスクは，図 7 の差でみると 5μs 程度である．これらの微小な誤差は，今回のモデルには反映していない，タスクや割込み処理の実行時間の細かなぶれや，タスク切り替えなどの OS のオーバヘッドが原因として考えられる．これらをモデルに反映させることで，より正確なモデルが作成できると考える．

以上の結果より，*schesim* を用いることで，従来のスケジューリングシミュレータのモデルより正確なアプリケーションモデルを作成し，シミュレーションできることが確認できた．正確なモデルを用いることで，例えば，本論文で対象としたエンジン制御アプリケーションに対し，マルチプロセッサ上でのスケジューリングを適用する，他の車載アプリケーションと統合し，階層型スケジューリングを適用することが可能となる．

本論文の評価で，シンプルモデル，RTSIM モデル，schesim モデルのシミュレーションに要した時間はそれぞれ，7.6 秒，6.5 秒，5.0 秒であった．シンプルモデルのシミュレーションが一番長い理由は，全てのタスク，割込み処理を周期タスクとして実装してあるため，次回の起動時刻などのイベントを管理する処理の時間が長くなったと考えられる．RTSIM モデルは，シンプルモデルに “act_tsk” を追加したため，シンプルモデルより周期タスクの数が少ない．しかし，クランクの回転に同期する割込み処理は，周期タスクとして実装してあるため，RTSIM モデルの周期タスクの数は，schesim モデルより多い．よって，シンプルモデルと同様の理由で，RTSIM モデルは schesim モデルよりシミュレーションに要する時間が長くなったと考えられる．

6. おわりに

本論文では，タスク処理を詳細に定義可能なアプリ

ケーションモデルと，そのモデルを扱うシミュレータを提案した．モデリング性能の評価のため，*schesim* が扱うモデルと既存のスケジューリングシミュレータが扱うモデルで，エンジン制御アプリケーションをモデル化して，シミュレーションした．その結果，ISS でのシミュレーション結果に対する *schesim* でのシミュレーション結果のタスク平均応答時間の平均誤差は 10.5%であった．これにより，エンジン制御アプリケーションのように複雑なアプリケーションに対しても，*schesim* を適用することで，より正確なシミュレーション結果を得ることができることを確認した．

今後の課題は，SMP 型マルチプロセッサアーキテクチャを想定したスケジューリングアルゴリズムをサポートすることである．AMP 型マルチプロセッサアーキテクチャの場合は，独立したシングルコアを複数用意することで実現できた．それに対して，SMP 型マルチプロセッサアーキテクチャの場合は，コア間でレディキューを共有するなど schesim の大幅な改良が必要となる．その他の課題としては，ユーザが新しいスケジューリングアルゴリズムを容易に実装するためのインターフェースを定義することが挙げられる．

参 考 文 献

1) T. Kramp, M. Adrian, and R. Koster, “An Open Framework for Real-Time Scheduling Simulation,” In Proceedings of International Parallel and Distributed Processing Symposium 2000 Workshops, pp.766–772, 2000.
2) C. Yaashuwanth and R. Ramesh, “Web-Enabled Framework for Real-Time Scheduler Simulator: A Teaching Tool,” In Proceedings of 2nd International Conference on Computer Research and Development, pp.826–830, May 2010.
3) A. Diaz, R. Batista, and O. Castro, “Realtss: a real-time scheduling simulator,” In Proceedings of 4th International Conference on Electrical and Electronics Engineering, pp.165–168, Sept. 2007.
4) A.M. Jr., M.B. Miola, and V.A. Nabuco, “Teaching Real-Time with a Scheduler Simulator,” In Proceedings of 31st Annual Frontiers in Education Conference, pp.15–19, 2001.
5) D. Decotigny and I. Puaut, “ARTISST: An Extensible and Modular Simulation Tool for Real-Time Systems,” In Proceedings of IEEE International Symposium on Object-oriented Real-time Distributed Computing, pp.365–372, April 2002.
6) A. Casile, G. Buttazzo, G. Lamastra, and G. Lipari, “Simulation and Tracing of Hybrid Task Sets on Distributed Systems,” In Proceedings of IEEE International Conference on Real-Time Computer Systems and Applications, pp.249–256, Oct. 1998.
7) C. Bartolini and G. Lipari, “RTSIM”. http://rtsim.sssup.it/
8) J. Kraft, “RTSSim - a Simulation Framework for Complex Embedded Systems,” Mlardalen University, Technical Report, March 2009.
9) N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings, “STRESS: a Simulator for Hard Real-time Systems,” Software: Practice and Experience, vol.24, no.6, pp.543–564, June 1994.
10) G.Lipari and K.Baruah, “Efficient Scheduling of Real-Time Multi-Task Applications in Dynamic Systems,” In Proceedings of IEEE Real-Time Technology and Applications Symposium, 2000.
11) 松原豊，本田晋也，高田広章，“時間保護のためのタスク起動遅延付き階層型スケジューリングアルゴリズム，” 情報処理学会研究報告，vol.2010-EMB-19，Dec. 2010.
12) (社) トロン協会 ITRON 仕様検討グループ，“μITRON4.0 仕様 Ver.4.03.03,” Sept. 2008.
13) OSEK/VDX, “OSEK/VDX Operating System Specification 2.2.3,” Feb. 2005.
14) TOPPERS Project, “TOPPERS/FMP kernel”. http://www.toppers.jp/fmp-kernel.html
15) 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，pp.8–23，Nov. 2010.