Vol.2011-SLDM-149 No.14
Vol.2011-EMB-20 No.14
2011/3/18

次世代アーキテクチャ分析のための性能解析モデリング手法の提案

岡林弘嗣[†1] 久住憲嗣[†2] 河原亮[†3]
小野康一[†3] 中田武男[†3] 坂本佳史[†4]
長野正[†4] 中西恒夫[†5] 福田晃[†5]

組込みソフトウェアの性能検証は，開発の後期に行うことが多く，手戻りが発生すると開発工期やコストの増大を招く．このリスクを下げるためには，できるだけ開発早期に性能検証を行うことが有効である．ソフトウェア設計者がシステムレベルで性能解析を行おうとする場合，様々な代替案で設計探索できるようにすることが望ましい．しかし，その際，並列性の変更などモデルの変更が手間となる．本研究では，その手間が少なくなるように MDD(Model-Driven Development) によるモデリング手法及び，モデル変換手法を提案する．

Modeling Methodology for Performance Analysis Aiding the Analysis of the Next Generation Architectures

Hirotsugu Okabayashi,[†1] Kenji Hisazumi,[†2]
Ryo Kawahara,[†3] Kouichi Ono,[†3] Takeo Nakada,[†3]
Yoshifumi Sakamoto,[†4] Tadashi Nagano,[†4]
Tsuneo Nakanishi[†5] and Akira Fukuda [†5]

It is source of increasing of development time and cost to discover that the system does not meet performance requirements at a late stage in a development process. Performance analysis in early development stage is important to prevent increase them. It is desirable that the developer can explore the diverse alternatives for the appropriate architectures. During the design space exploration, however, it is a time-consuming work for the developers to modify software and hardware model taking concurrency into account. To reduce that modeling cost, this paper proposes modeling methodology and the rules of model transformation for MDD(Model-Driven Development).

1. はじめに

組込みシステムの性能検証は，開発の後期に行うことが多い．そのため，手戻りが発生すると開発工期やコストの増大を招く．このリスクを下げるためには，できるだけ開発早期に性能検証を行うことが有効である．また，組込みシステムの新製品は多くの場合，既存製品をもとに性能向上や新機能を追加するなどの改良をされ，一から開発される機会は少ない．本論文で例示する MFP(Multifunction Peripheral/Printer) システムのハードウェアプラットフォームのリプレースも同様である．今日の MFP などの組込みシステムでは，プロセッサや ASIC(Application Specific Integrated Circuit) に固有の機能を割り当てる粗粒度な並列化が行われている．よって，本研究での並列性とは命令レベルの粒度ではなく，スレッドレベル，またはそれ以上の抽象度での並列性を指し，粗粒度のパイプライン処理による性能解析を対象とする．既存製品をもとに改良を行う差分開発では，ソフトウェアやハードウェアに変更が生じる．性能はハードウェアだけでなく，ソフトウェアの割り当て方にも依存するため，ソフトウェア技術者が設計を行いながら性能解析ができるようにする開発方法論が必要になる．そのような方法論に求められる要件として，開発早期の段階で，容易に多くのアーキテクチャの性能を見積もることができるように設計探索の支援を行うことが挙げられる．

しかし，ソフトウェア開発者がシステムレベルで性能解析を行おうとすると，モデリングの際，並列性や，共有資源の排他制御を加味した振る舞いを作成する必要がある．また，ソフトウェアのハードウェアへの割り当てを変更すれば，並列性や，排他制御も変更が必要になる．いくつもアーキテクチャを作成する度に，これらの変更が伴うことは，設計探索において不利な特徴といえる．

†1 九州大学大学院システム情報科学府
Graduate School of Information Science and Electrical Engineering, Kyushu University
†2 九州大学 システム LSI 研究センター
System LSI Research Center, Kyushu University
†3 日本アイ・ビー・エム株式会社 東京基礎研究所
IBM Research - Tokyo
†4 日本アイ・ビー・エム株式会社 グローバル・ビジネス・サービス
Global Business Services, IBM Japan, Ltd.
†5 九州大学大学院 システム情報科学研究院
Faculty of Information Science and Electrical Engineering, Kyushu University

Vol.2011-SLDM-149 No.14
Vol.2011-EMB-20 No.14
2011/3/18

本研究では，ソフトウェア全体の振る舞いをデータフローとして記述し，ソフトウェアとハードウェアとの割り当てを明示するモデリング手法を提案する．これにより，ソフトウェア全体の振る舞いが，データを処理するノードとそれらの間のデータの流れで記述できる上，どのソフトウェアがどのハードウェアで実行されているかが分かる．そのため，処理全体がパイプライン化された場合に，各ソフトウェアの実行を開始するタイミングがわかる．したがって，開発者はモデルにハードウェアへの割り当て変更さえ明示すれば，パイプラインの構造を意識することなくモデリングを行えるようになり，ソフトウェアの振る舞いを変更する手間が省ける．さらに，本研究では，提案手法から実行可能なモデルへの変換規則を明確にし，モデル駆動型の設計・シミュレーション手法を提案する．

第 2 章で，協調設計と性能シミュレーションについての関連研究を紹介し，それらの問題点を挙げる．第 3 章で，モデリング手法を提案し，第 4 章で，モデル変換手法について説明し，第 5 章で，提案手法の評価を行う．最後に第 5 章でまとめを述べる．

2. 協調設計と性能シミュレーションの関連研究

LSI(Large Scale Integration) のハードウェアとソフトウェアの協調設計に利用される SystemC などの ESL(Electronic System Level) 言語は，ハードウェアとソフトウェア双方の設計を行うシステムレベルのアーキテクチャ設計に適している．しかし，ESL 言語はソースコードによる記述のため，各モジュールの関係性が分かりにくいなど，視認性が低く全体像を把握しにくい．

そこで近年，協調設計に UML[11] を用いた研究が盛んに行われている[2),5)]．この用途向けに UML プロファイルを拡張した代表的な仕様として MARTE(Modeing and Analysis of Real-Time and Embedded Systems)[10] がある．MARTE はシステム分析のための仕様を備えており，性能解析のための記述ができる[4]．本研究のモデリング言語として使用する．

UML を用いた高い抽象度での性能解析手法として Cortellesa らの手法がある[8]．高い抽象度でコンピュータシステムのシミュレーションを行う研究の代表例としては，Queuing Network などの計算モデルを使用したシミュレーション手法が挙げられる[1] が，Cortellesa らの手法は全て UML でモデル化している．これにより，システム設計者はより設計に注力できるようになった．しかし，この手法はプロセッサのリソース競合のみに焦点が当てられており，その他の共用リソースの性能解析を行うための具体的な計算方法を与えていない[7]．

一方，河原らは，共用リソースの中でも利用頻度が高いと思われる共有メモリについての具体的な計算方法を提案している[7]．この計算手法は期待値を算出する統計的な手法であるため，メモリアクセスのタイミングを考慮する必要がなく，モデルへの記述量が少なくてすむため UML などの抽象度の高いモデルシミュレーションに適している．これにより，シミュレーションの精度向上が見込める．本シミュレーション技術はこの技術を応用したものである．

また，差分開発の性能解析を支援する技術として，文献 3) がある．文献 3) では，既存システムの実行トレースの捨象を行うことで，リバースモデリングを行っている．これは性能解析に特化したリバースモデリングであり，既存システムから計測した性能情報をモデル実行時に，読み込むことでトレース駆動シミュレーションを行う．この提案手法により，既存システムを性能観点からシングルプロセッサからマルチプロセッサへ対応させることが可能となった．本研究でも，既存システムのシングルプロセッサからマルチプロセッサへの対応を扱うため，この既存システムの性能情報を用いたトレース駆動によるシミュレーションを行う．

しかし，文献 3)，7)，8) のいずれの手法も，手作業による並列化が必要なので，これを軽減できれば，設計探索のコストを削減できる．

3. 性能解析モデリング手法

3.1 提案する性能解析モデリング手法の流れ

本性能解析モデリング手法は Application Model 層と Executable Model 層，Resource Manager 層の 3 層構造となっている．Application Model 層は，開発者がソフトウェア全体の振る舞いをデータフローとして記述し，ソフトウェアとハードウェアとの割り当てを明示するモデリング手法を用いてモデリングを行うレイヤである．また，このレイヤでモデルへの性能情報の付与も行う．Executable Model 層は，Application Model 層から変換されたシミュレーションのためのより詳細化されたモデルを扱うレイヤである．ここでのモデルとは，UML にアクションコードによる細かい挙動を追加したもので，シミュレーション可能なモデルを指す．処理全体がパイプライン化された場合に，各ソフトウェアの実行を開始するタイミングや，共有メモリのアクセス競合による処理時間の遅延を計算するメソッド呼び出しをモデルに反映さる．なお，このレイヤはモデル変換が行われるレイヤであり，開発者が直接モデルを扱うことはしない．Resource Manager 層は，関連研究で述べた算出方法を利用し，共有メモリのアクセス競合による処理時間の遅延を計算し，シミュレーションを行うレイヤである．なお，これらの実装には IBM Rational Rhapsody©ソフトウェアを使用した．

Vol.2011-SLDM-149 No.14
Vol.2011-EMB-20 No.14
2011/3/18

次に，提案手法を利用した開発プロセスについて説明する．まず既存システムの性能情報を取得とモデル化を行い，本モデリング手法を用いアーキテクチャを変更する．さらに，本モデル変換手法によりモデル変換を行い，シミュレーションを行う．

モデル駆動型開発技術を利用し，設計探索の支援を行うことで，性能という観点から開発早期により多くの設計をより早く発見出来るようになる．

3.2 性能解析のためのモデリング手法

3.2.1 概　　要

前述のモデリングの手間を低減するには，その問題領域に特化したモデリング手法とそれを実行可能な形式に変換する規則を明確にする必要がある．ここでは，モデリング手法について詳しく説明する．一般に UML を用いて組込みシステムをモデリングしようとすると，UML プロファイルを用いることになる．リアルタイム向けの仕様として MARTE があり，本モデリング手法は MARTE プロファイルを利用している．しかし，本手法はモデリングの手間の低減のために新しくステレオタイプを定義しているため，本論文はこの新たに定義したステレオタイプを用いたモデリング手法について論じる．

提案する手法は，ソフトウェア開発プロセスに支障を来すことなく設計図として利用できなくてはならない．そこで，図 1 のように詳細に設計・解析したいソフトウェアコンポーネントの振る舞いのノードを，新たなソフトウェアコンポーネントとして入れ子構造にして割り当てる．このようにツリー状に展開すれば，段階的に詳細化でき，開発者の目的に応じたシミュレーションができるようになる．このモデリング手法は，動的モデリングと静的モデリングの 2 つの側面があり，3.2.2 項と 3.2.3 項にて詳しく説明する．

3.2.2 静的構造のモデリング

ここでは，提案するモデリング手法の静的なモデリングについて解説する．静的なモデリングはオブジェクト図で描く．1 からもわかるように，詳細化するために展開したオブジェクトには≪ child ≫というステレオタイプをつけることにする．一方，展開元のオブジェクトは≪ parent ≫というステレオタイプをつける．なお，≪ parent ≫を持つオブジェクトは≪ child ≫とリンクでつなげることとする．

3.2.3 動的構造のモデリング

ここでは，提案するモデリング手法の動的なモデリングについて解説する．動的なモデリングは，データフローを表現するためにアクティビティ図を用いる．アクティビティ図にはデータフローと，そのアクティビティで消費する処理時間，及び，バスの使用率を記述する．よって，動的なモデリングの構成要素は，オブジェクトノードとオブジェクトフロー，

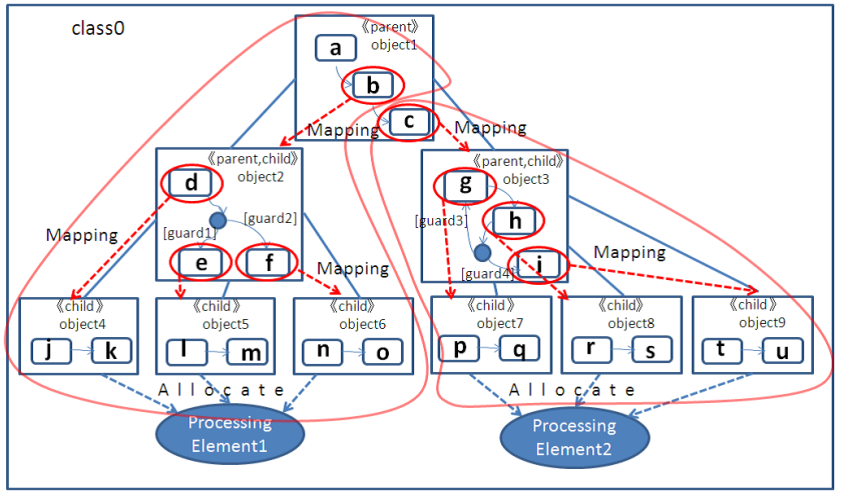

図 1　提案手法によるモデリングのイメージ

アクションノードである．ここで，本研究のシミュレーションは，共有メモリのアクセス競合による遅延を期待値により算出することが出来るように，Rhapsody のシミュレーションフレームワークを拡張している．そのため，独自のメソッドを呼び出す必要がある．このメソッド呼び出しは，アクションにて記述する．任意の処理時間とその時間内のバスの利用率がメソッドの入力となる．本手法では，≪ DCSR ≫というステレオタイプをノードに付け，Rhapsody のシミュレーションフレームワークに計算を依頼する．

また，≪ parent ≫の振る舞いのノードと≪ child ≫オブジェクトのマッピングを明示する必要がある．これには≪ parent ≫オブジェクトの持つアクションノードのタグ付き値に child 属性を追加し，Value 値にマッピングする≪ child ≫オブジェクトの名前を記述する．

本手法で扱うことができる動的なモデリングは，シーケンシャルな振る舞いに限る．

4. モデル変換手法

4.1 モデル変換手法の概要

提案するモデリング手法は，オブジェクトの振る舞いの任意のノードを入れ子構造にして新しいオブジェクトへとツリー状に展開していく．さらに，提案手法は，展開され小規模な振る舞いを持つソフトウェアコンポーネントがどのハードウェアに割り当てられているかを

Vol.2011-SLDM-149 No.14
Vol.2011-EMB-20 No.14
2011/3/18

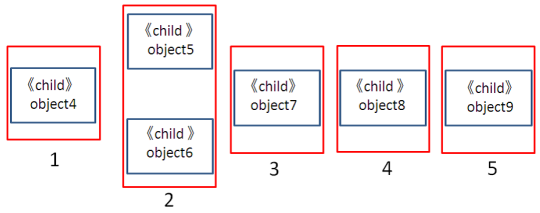

図 2 最下層ソフトウェアコンポーネントの実行順序

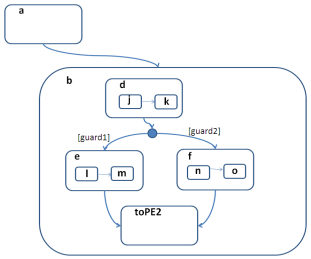

図 3 PE の振る舞いのモデル

モデルに明示する．そのため，処理全体がパイプライン化された場合に，ハードウェアに割り当てられる任意のソフトウェアコンポーネント同士 (以下，最下層のソフトウェアコンポーネント) の実行を開始するタイミングがわかることを前に述べた．よって，実行可能なモデルを得るには，最下層のソフトウェアコンポーネントが逐次的にもしくは並列的に実行されるかを判断し，最下層のソフトウェアコンポーネントの実行の制御を行うオブジェクトが存在することが必要である．また，最下層のソフトウェアコンポーネントの制御を行うためには，最下層のソフトウェアコンポーネントが実行される順序もわかっておく必要がある．以下に，最下層のソフトウェアコンポーネントの制御を行うオブジェクトの作成手順と，最下層のソフトウェアコンポーネントが実行される順序の抽出手順を前節の図 1 を例にとり説明する．

(1) 最下層のソフトウェアコンポーネントの実行順序の取得方法

前項の図 1 での最下層ソフトウェアコンポーネントは object4，object5，object6，object7，object8，object9 である．これらの実行順序を得るためには，まず object1 が持つノードを先頭から探索する．具体的にはノード b が object2 とマッピングされているので，object2 の先頭ノードから探索する．先頭ノードであるノード d は object4 とマッピングされているので，object4 が最下層ソフトウェアコンポーネントの中で実行順序が一番早いことが分かる．この手順を再帰的に行えば，図 2 のように実行順序を得ることができる．

(2) 最下層のソフトウェアコンポーネントの制御を行うオブジェクトの作成方法

基本的に，最下層のソフトウェアコンポーネントの制御を行うオブジェクトは，最下層のソフトウェアコンポーネントのマッピング先である Processing Element(PE) ごとに作成する．以下，最下層のソフトウェアコンポーネントの制御を行うオブジェクトのことを，ソフトウェアを実行するハードウェアと模して PE と呼ぶことにする．最下層のソフトウェアコンポーネントが実行順序を得れば，PE はその順序に従って最下層のソフトウェアコンポーネントへ起動通知を行う．しかし，前項の図 1 の object2 に見られるような分岐擬似状態などの制御情報も反映させる必要があるので，前項の図 1 の大きな枠のようにシステム全体の振る舞いを各 PE に分割して割り当てる必要がある．

また，モデル変換後の実行可能なモデルは，図 1 のように PE に振る舞いを割り当て，最下層のソフトウェアコンポーネントとメッセージをやりとりする必要がある．このことを考えると，モデル変換後の実行可能なモデルの静的構造は図 4 のようにする必要がある．これは前項の図 1 を変換したモデルになる．また，変換後の PE の振る舞いを図 3 に示す．

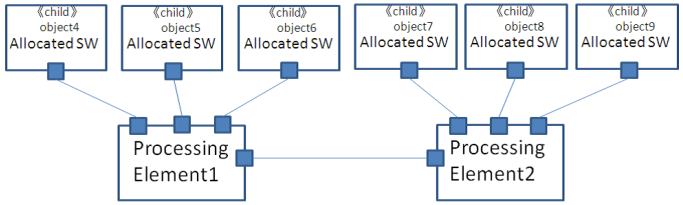

図 4 図 1 のモデル変換後の静的構造

このように，提案するモデリング手法を用いれば，ソフトウェア全体の振る舞いが，データを処理するノードとそれらの間のデータの流れで記述できる上，どのソフトウェアがどのハードウェアで実行されているかが分かる．そのため，ソフトウェアのハードウェアへの割り当てさえ明示すれば，開発者はパイプライン処理になった場合のソフトウェアの実行のタイミングや，実行順序を明示することなくシミュレーションを行うことが可能となる．

また，モデル変換を達成するための機能としては，上記以外にも，最下層のソフトウェアコンポーネントと PE 間の通信に利用するイベントの生成や，最下層のソフトウェアコンポーネントと PE または，PE 同士を結ぶリンクとポートの作成が必要となる．

Vol.2011-SLDM-149 No.14
Vol.2011-EMB-20 No.14
2011/3/18

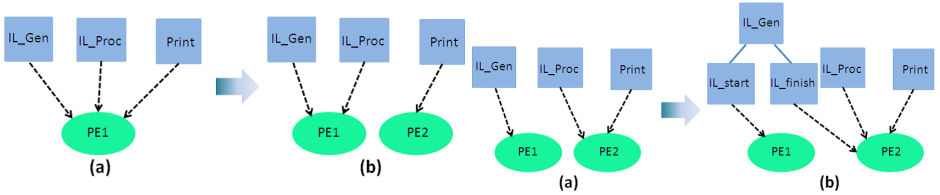

図 5　パターン 1

図 6　パターン 3

5. 評　　価

5.1 概　　要

本研究では，実際の MFP 製品の印刷処理部分に対し，提案手法の適用・評価を行った．適用にあたり，JEITA（電子情報技術産業協会）によるプリンターベンチマーク用のテストパターンの J12 セット[12] をテストケースとして利用した．今回の評価対象の MFP の印刷処理の概要を説明する．IL_Generator は PDL(Printer Description Language) によって記述された印刷ジョブを中間言語に変換する．次に IL_Processor は中間言語で記述された命令を解釈して論理ページのラスターデータを作成する．最後に Print が物理ページを出力する．

提案手法の適用・評価の手順について説明する．

(1)　MFP 製品のソフトウェアを製品と同等のアーキテクチャの FPGA 評価ボードに移植し，実行トレースを性能情報と共に取得する．性能情報の取得は，実機の関数の実行時間などを高精度で計測する技術[9] を使用する．

(2)　計測した情報から性能パラメータの算出やシステムの振る舞いの抽出を行う．

(3)　性能パラメータやシステムの振る舞いモデルを利用し，提案手法による設計・シミュレーションを行う．

(4)　モデル上のアーキテクチャを変更し，シミュレーションを行う．

(5)　シミュレーション結果を評価用の FPGA ボードの計測時間と比較する．

本論文でシミュレーションを行う際の，アーキテクチャ変更パターンを図 5 に示す．

図 5 のシミュレーションを評価するため FPGA ボードで MFP プロトタイプを作成した．2 つのアーキテクチャパターンを評価するためにシングルプロセッサーとマルチプロセッサーを持つ 2 つの MFP プロトタイプを作成した．IL_Generator と IL_Processor はソフトウェアを移植し，Print 処理はハードウェアにて処理されているため，処理時間を固定値とし，共有メモリのアクセスを DMA(Direct Memory Access) にて疑似的に再現している．

本研究の評価指標は，設計探索の際のモデリングのコストである．5.2 節で，その評価指標に対し評価を行う．

5.2 モデリングコストの比較

前節の図 5 に加え，2 つのパターンのアーキテクチャ変更に費やすコストの比較を行った．2 つのパターンとは，図 5(b) の IL_Proc の PE1 への矢印を PE2 へ変更するパターン (パターン 2) と，図 6(パターン 3) のように IL_Proc を 2 つのソフトウェアコンポーネントに分割し，片方を PE1 へもう片方を PE2 へ割り当てるパターンである．

なお，ここで言うアーキテクチャ変更に費やすコストとは，モデリングの手間であり，アーキテクチャ変更の際に開発者が実際に手を動かす手間のことを言う．モデルの変更度合いを把握する手法が文献[6] にて述べれているが，設計図のソフトウェアの構成管理の際に開発者にできるだけ少ない情報で変更度合いを伝えることを目的としており，モデリングの手間については触れられていない．モデリングの手間を定量的に扱う手法がないため，本論文では，アーキテクチャ変更に伴うモデリングの手間を以下のように定義し，モデルの編集距離と呼ぶ．

モデル編集距離の測定方法

- Action Code などのテキスト以外の作成・変更・削除したモデル要素を 1 要素につき 1 カウントする
- Action Code などのテキストは，作成・変更・削除した行を 1 行につき 1 カウントする

上記方針に従い，各アーキテクチャ変更パターン毎のモデルの編集距離を測定した結果を表 1 に示す．モデリングコストは，表 1 のパターン 1 では 61 から 1 へ，表 1 のパターン 2 では 23 から 1 へ，表 1 のパターン 3 では 64 から 11 へ削減できた．本研究では，従来手法としてリアルタイム UML[13] によるモデリングを行った．表 1 のパターン 1 のコストに大きな開きがある理由は，従来手法の場合，シンプルプロセッサからマルチプロセッサに変更されるので並列処理に変わることになる．よって，AND 状態の記述を新たに加える必要があるからである．また，表 1 のパターン 3 では両者ともにコストが高い．この理由は，ソフトウェアアーキテクチャを詳細化しハードウェアへの割り当てを変更するので，新たにオブジェクトを作成する必要があるからである．前述と同様に，リアルタイム UML によるモデリングは AND 状態も新たに加える必要もある．

Vol.2011-SLDM-149 No.14
Vol.2011-EMB-20 No.14
2011/3/18

表 1 パターン毎の編集距離の測定

	モデル化方法	object	attribute	port	link	dependency	state	transition	event	guard	actioncode
パターン 1	リアルタイム UML によるモデリング	0	3	0	0	0	9	15	13	3	18
	提案手法によるモデリング	0	0	0	0	1	0	0	0	0	0
パターン 2	リアルタイム UML によるモデリング	0	3	4	2	0	2	3	4	0	5
	提案手法によるモデリング	0	0	0	0	1	0	0	0	0	0
パターン 3	リアルタイム UML によるモデリング	0	3	4	2	0	2	3	4	0	5
	提案手法によるモデリング	0	0	0	0	1	0	0	0	0	0

6. おわりに

本研究では，ソフトウェア全体の振る舞いをデータフローとして記述し，ソフトウェアとハードウェアとの割り当てを明示するモデリング手法を提案した．これにより，システム全体をシーケンシャルに記述できる上，どのソフトウェアがどのハードウェアで実行されているかが分かるため，処理全体がパイプライン化された場合に，各ソフトウェアの実行を開始するタイミングがわかる．したがって，開発者はモデルにハードウェアへの割り当て変更さえ明示すれば，パイプラインの構造を意識することなくモデリングを行えるようになり，ソフトウェアの振る舞いを変更する手間が省ける．さらに，本研究では，共有メモリのアクセス競合による遅延を算出する手法を取り入れ，提案手法から実行可能なモデルへの変換規則を明確にすることで，モデル駆動型の設計・シミュレーション手法を提案した．

この設計・シミュレーション手法を利用することで，設計探索に費やすコストを約 1/10 に抑えることに成功した．

商　　標

IBM, Rational, Rhapsody は，それぞれ，International Business Machines Corporation (IBM Corp.) の米国およびその他の国における登録商標である．他の会社名，製品名およびサービス名等はそれぞれ各社の商標である．

謝　　辞

実行トレースを提供して頂いた京セラミタ株式会社東京 R&D センターの福岡直明氏に謹んで感謝の意を表する．

参 考 文 献

1) Balsamo, S., Di Marco,A., Inverardi,P., and Simeoni,M., “Model-based performance prediction in software development: a survey,” *IEEE Transactions on Software Engineering*, vol.30, Issue.5, pp.295–310, 2004.
2) E. Riccobene, P. Scandurra1, A. Rosti, and S. Bocchio, “A SoC Design Methodology Involving a UML 2.0 Profile for SystemC,” *Proceedings of the conference on Design, Automation and Test in Europe*,vol. 2, pp.704–709, 2005.
3) Kouichi Ono and Manabu Toyota and Ryo Kawahara and Yoshifumi Sakamoto and Takeo Nakada and Naoaki Fukuoka, “A Model-based Method for Evaluating Embedded System Performance by Abstraction of Execution Traces,” *Proceedings of 6th European Conference on Modelling Foundations and Applications*, vol.6138, pp.233–244, 2010.
4) Marco Bernardo and Jane Hillston, “From annotated software designs (UML SPT/MARTE) to model formalisms,” *SFM’07 Proceedings of the 7th international conference on Formal methods for performance evaluation*, pp.429–467, 2007
5) Nguyen, K.D., Zhenxin Sun, Thiagarajan, P.S., and Weng-Fai Wong, “Model-driven SoC Design Via Executable UML to SystemC,” *Proceedings of 25th IEEE International Real-Time Systems Symposium*,pp. 459–468,2004.
6) Ohst, D., Welle, M., and Kelter, U. “Differences between versions of UML diagrams,” *SIGSOFT Softw. Eng.*, vol.28(5), pp.227–236, 2003.
7) Ryo Kawahara and Kenta Nakamura and Kouichi Ono and Takeo Nakada and Yoshifumi Sakamoto, “Coarse-Grained Simulation Method for Performance Evaluation a of Shared Memory System,” *Proceedings of the 16th Asia and South Pacific Design Automation Conference*, pp.413–418, 2011.
8) V.Cortellessa, P.Pierini, and D.Rossi, “InteGrating Software Models and Platform Models for Performance Analysis,” *IEEE Transactions on Software Engineering*, vol.33, pp.385–401, 2007.
9) Ohba, N. and Takano, K., “Hardware debugging method based on signal transitions and transactions,” *Proceedings of the 11th Asia and South Pacific Design Automation Conference*, pp.454–459, 2006.
10) http://www.omg.org/spec/MARTE/
11) http://www.uml.org/
12) Japan Electronics and Information Technology Industries Association (JEITA), *JEITA Printer Benchmark Test Patterns*, http://it.jeita.or.jp/document/printer/pattern/J1-J12.pdf
13) ブルース・ダグラス (著), 渡辺博之 (監訳),『リアルタイム UML オブジェクト指向による組込みシステム開発入門 第２版』, 翔泳社, 2001.