RB-002

MARTEプロファイルを適用したUMLモデルによる組込みシステムの性能評価シミュレーション
Performance simulation of embedded system using MARTE-based UML model

河原 亮* 小野 康一* 中田 武男* 豊田 学† 坂本 佳史‡ 福岡 直明§
Ryo Kawahara* Kouichi Ono* Takeo Nakada* Manabu Toyota†
Yoshifumi Sakamoto‡ Naoaki Fukuoka§

1. はじめに

マルチコアプロセッサーの普及など，組込みシステムが複雑，大規模になるにつれ，性能要件を満たすように適切なアーキテクチャーを選択することが難しくなっている．システムテストの段階になって性能面の問題が判明し，その原因が基本的な設計にあった場合，大きな手戻りとなる．開発期間の短縮要求に対応するためには開発プロセスの上流工程で性能評価を行うことが重要であり，そのための性能予測手法およびツールサポートが必要である．

これまでに確立された性能評価手法としてはQueuing Network (QN)やPetri Net (PN)など性能評価専用のモデリング手法がある[1]．たとえばQNはシステムをサーバーとキューからなる要素のネットワークとして表現し，客の到着率やサービス率などの少ない種類のパラメーターによって特徴付ける，抽象的な手法である．これらのモデリング手法では性能評価の専門家によって解析的，ないしシミュレーションによって性能が評価される．

一方，主にソフトウェアの機能面をモデリングするための言語である Unified Modeling Language (UML)は Object Management Group (OMG)によって標準化された言語であり，ソフトウェア開発において広く受け入れられている．

これまで性能評価とUMLベースのソフトウェアモデリングを統合する試みが行われてきた[2]．UMLと適切に定義されたプロファイルを併用することで，システムを構成するコンポーネントや，それらの状態遷移など，システム設計者の理解しやすい粒度でシステムをモデル化することができる．両者の統合により設計情報と性能の対応関係がより理解されやすくなり開発プロセスの改善につながると期待される．

OMGのUML Profile for MARTE (Modeling and Analysis of Real-Time Embedded systems) [3]はそのような努力の成果のひとつである．これを用いるとUMLモデルには性能パラメーターやアーキテクチャーの情報を含めることができ，組込みシステムの分析や設計におけるUMLの使用を促進することを狙っている．MARTEはいくつかのコンセプトから成り立っているので一部を紹介する．

MARTE の Detailed Resource Modeling のコンセプトは，UML の要素を使ってハードウェアリソースやソフトウェアリソースを表せるようにするものである．標準の UML ではクラスやコンポーネントを単位としてモデリングを行うが，それに加えてプロセッサーやメモリー，セマフォなどの組込みシステムの構成コンポーネントを，その仕様(プロセッサースピードやメモリ容量など)とともに UML 中に記述することができる．

MARTE Allocation Modelingのコンセプトはシステムのリソース割り当てを宣言的に指定するためのものである．リソース割り当ての例としては，どのソフトウェアコンポーネントをどのプロセッサーで実行するかの指定がある．この概念はModel Driven Architecture (MDA)[4]に見られるようなプラットフォームの情報をソフトウェアのモデルから分離したモデリングをサポートするのに用いることができる．同様のアロケーションの概念はSysMLにも見られる[5].

一方，組込み製品の典型的な開発プロセスでは既存製品のソースコードを基に開発を始める．新製品のプラットフォーム上に既存アプリケーションソフトウェアのソースコードの多くの部分が移植される．そのためアーキテクトは新しいアーキテクチャーでのシステム性能を既存製品のアーキテクチャーでの性能などの情報から見積もる必要がある．

既存製品をベースにした開発プロセスではリファレンスとなる既存製品の実行トレースなどの動的情報が存在する．この場合，ここから性能予測のための各種のパラメーターやテストデータを抽出し新しいシステムでの挙動を予測するトレースドリブンシミュレーションが有効である[6] [7]．トレース情報を使うと，既存製品については命令レベルの細かい情報を集めなくても実行時間などの情報を得ることが出来るので，これをパラメーターとするとモデリングにかかる工数を減らすことが出来る．

このようなプロセスの中でアーキテクチャー設計にUML を用いるとすると，UML モデルに性能評価のための各種のパラメーターを記述できる必要がある．特にアプリケーションのモデル自体は機能検証用のモデルや，下流における設計文書としての役割も果たすため，性能評価用の情報は宣言的に記述できることが望ましい．具体的には

- アプリケーションモデルは性能評価のための特別なロジックを意識せずに記述できること

が望ましい．さらに複数のプラットフォームで性能を評価，検討するため，

- プラットフォームとアプリケーションのモデルが出来る限り分離されていること

* 日本アイ・ビー・エム株式会社 東京基礎研究所，IBM Research – Tokyo
† 日本アイ・ビー・エム株式会社 コンポーネントテクノロジーソリューション，Component Technology Solution, IBM Japan
‡ 日本アイ・ビー・エム株式会社 グローバルビジネスサービシズ，Global Business Services, IBM Japan
§ 京セラミタ株式会社 東京 R&D センター，Tokyo R&D Center, KYOCERA MITA Corp.

が望ましい．さらにモデルをツールで利用することを考えると

- 性能評価のための情報の記述がメタモデルレベルでサポートされていること

も必要である．

前述の MARTE による拡張要素を導入すると，これらの点において要件を満たすようなモデリングが可能である．一方，標準の UML であれば，デプロイメント図でシステムの物理的なアーキテクチャーを記述することは可能だが，それがどのようなクラスのリソースなのか(プロセッサーなのかサーバーなのか)といった情報を与えるには何らかのプロプライエタリーな拡張が必要となる．

その一方で，このように宣言的に記述されたモデルからの性能評価を行うにはツールによるサポートが必要であるが，現状ではツールサポートについては研究の途上である．これについて2節でいくつか紹介する．

本研究では，MARTE ベースの UML モデルのシミュレーションを用いた組込みシステムの性能評価手法を提案する．適用対象とする組込みシステムは非リアルタイム，またはソフトリアルタイム系とする．このクラスには MFP(Multi-Function Printer)などの比較的処理データ量が多い組込みシステムが含まれ，マルチコアプロセッサーの採用による性能向上が期待されている．また性能の指標として計算するのはアプリケーションの実行時間である．メモリー使用量や消費エネルギーなど他の性能指標も検討の必要があるが，その中で特に実行時間がどのシステムにおいてもシミュレーションによる動的な評価の需要が高いので最初に実装する必要があると判断した．

したがって今回提案するのは，ソフトウェアのUMLモデルに対してMARTE Allocation Modelingを用いて複数のプラットフォームを指定し，リファレンスシステム(既存製品)での各詳細ステップの処理時間を性能パラメーターとして与え，各プラットフォーム上での実行時間を計算するシミュレーション手法である．2節においてUMLを用いた性能評価についての関連研究を紹介する．3節では提案手法の詳細を説明する．4節ではケーススタディとして印刷システムを取り上げ，その結果を5節で議論する．6節が本論文のまとめである．

2. 関連研究

Profile for Schedulability, Performance and Time (SPT)およびその後継であるMARTEプロファイルの登場以降，これらのプロファイルをUMLモデルに対して性能評価のための情報のアノーテーションに用いる手法がいくつか研究されている[8]．そのように性能情報が付加されたモデルからどのように性能評価を行うかについての研究をここで 2 つ紹介する．

Balsamoら[9]による手法では，性能パラメーターを与えたUMLモデルをQNモデルに変換し，性能に関する統計を取得している．QNモデルは解析的に各種の性能情報が得られる点で強力なツールであるが，通常UMLで設計されるシステムの情報よりも抽象度の高いモデルであるため，UMLレベルでの設計に必要なより詳細なシミュレーション結果とはギャップがある．

Cortellessaらはソフトウェアモデルとプラットフォームモデルという 2 つの分離したモデルからシミュレーションモデルを構成し評価する手法を提案している[10]．ソフトウェアモデル自体は時間つきの実行可能UMLであるため，そのシミュレーションの粒度は設計情報の粒度に一致している．またこの関心事項の分離によって，プラットフォームの設計空間をソフトウェアの変更量を抑えながら探索することができる．一方，ソフトウェアとプラットフォームの相互作用の記述については手作業で記述する必要があり，宣言的なモデリングやシミュレーション生成の自動化が実現されていない．

そこでMARTEプロファイルのAllocation Modelingのコンセプトを利用したモデルからのシミュレーションの手法がPielらによって提案されている[11]．彼らの研究においてはMARTE Allocation Modelingによってハードウェアプラットフォームとアプリケーションの振舞いのマッピングを記述したUMLモデルからSystemC[12]モデルへのコード生成を行うツールを開発している．彼らの手法ではSystemCによるシミュレーションを利用するため，個々のトランザクションなど通常のソフトウェアのUMLモデルよりも細かい粒度の仕様が必要になる．この手法のアプリケーションは信号処理システムであったため，ロジックの振舞いに専用のドメイン特化型モデリング言語を用いることで，この細かい粒度の記述を可能にしているが，本研究のように一般的な組込みシステムへの適用を目指すとそのままでは応用できない．

性能評価を目的としたものではないが，同様の他言語への変換を応用した手法はVidalらによっても発表されている[13]．彼らの手法ではMARTE DRMを用いたモデルからVHDLコードを生成する．

総合すると，これらの関連研究では，MARTE Allocation Modeling を用いて宣言的にプラットフォームの情報を取り入れた UML モデルから，同じ抽象度のシミュレーションモデルへ変換し性能評価することを実現した手法がなかったことになる．UML を用いた設計段階では，スレッドレベルの並列性と処理間の同期，およびそれらへのプラットフォームの影響が検討されるため，シミュレーションにもこれらの粒度の効果が反映される必要がある．一般論としては QN による抽象的なレベルの性能解析は UML レベルの具体的な設計に入る前に行っておくことが望ましい．また UML より細かい粒度のモデルによる性能評価は，モデリングのコストが高いため，少なくとも UML の粒度の仕様が明確になった後の，より下流の工程で行うのが合理的である．

本研究の目的は，次の 2 点を両立して性能評価結果をUMLレベルの設計へフィードバックしやすくすることである．I) MARTEプロファイルを用いて性能評価に必要な情報を宣言的に記述することで，1節で紹介した複数プラットフォームの比較の容易さやソフトウェアの設計文書としての役割などの利点を活用すること，II) UMLレベルの粒度のシミュレーションをすることによって実行結果とUMLモデルが対応するようにすること．

本研究では，上記の手法を実現するために，MARTE モデルからこの粒度のシミュレーションへの変換をツールとして実装する．アプリケーションとプラットフォームのモデルを分離するアプローチを採用する．リソースの割り当ては MARTE Allocation Modeling を用いて指定する．アプリケーションのモデルは通常の実行可能な UML とし，一

般の組込みアプリケーションに適用可能な手法とする．シミュレーションの粒度は UML のアクションやオペレーションの粒度とし，これらに対して性能パラメーターを与える．パラメーターの多くは既存製品のトレース情報を解析することで得られる．ここから生成されるシミュレーションコードは UML の個々のステップを処理し時間を計算する．

本研究で提案する手法は，MARTE で拡張された UML を用いることを念頭においているが，モデリングの粒度がシステムレベルの粗い振舞いや構造を記述していれば UML 以外にも適用可能である．たとえば類似のモデリング言語である SysML などへの適用は容易であろう．また非 UML 系の状態遷移図を用いるモデリング言語でも原理的には使用可能であるが，実用的にはモデルの相互理解やツール連携を考慮すると，アロケーションやプラットフォーム仕様の記述に UML および標準化されたプロファイルを用いることが望ましいと考えている．

3. 手法とツール

3.1 概要

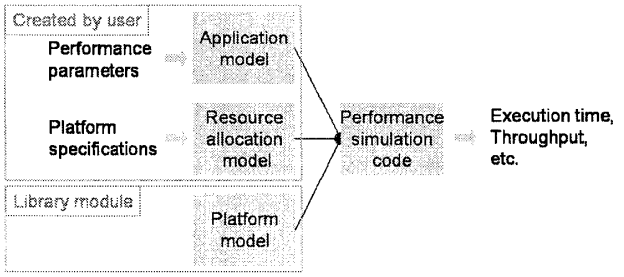


図 1 モデリングとシミュレーションコード生成の概要

本手法の概要を図 1に示す．本手法では 3 種類のモデルが登場する:

- アプリケーションモデル: システムのうちプラットフォームに独立な部分を記述する．ソフトウェアモデルと呼ぶ文献もある．
- リソースアロケーションモデル: アプリケーションコンポーネントをどのプラットフォームコンポーネントに割り当てるかを指定する．このモデルはアプリケーションモデルとプラットフォームモデルの結合方法を指定する役割を持つ．
- プラットフォームモデル: プロセッサーやバス，メモリーといったプラットフォーム要素の振舞いをシミュレートする．プロセッサースピードなどの性能にかかわるパラメーターはリソースアロケーションモデルで与える

アプリケーションモデルとリソースアロケーションモデルはユーザーによって作成されるが，プラットフォームモデルは再利用可能なライブラリーとして提供する．現在の実装では，プラットフォームコンポーネントとしてはプロセッサーのみに対応している．

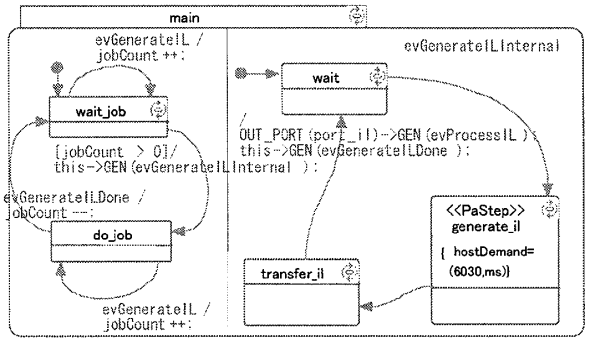
シミュレーションコード生成ツールはこれらの 3 種類のモデルを統合し，シミュレーションプログラムに変換する．シミュレーションは実時間ではなくシミュレーション時間(仮想時間)で行われる．シミュレーションプログラムは実行時間のほか実行トレースも出力する．

3.2 シミュレーション

アプリケーションモデルは IBM Rational Rhapsody®ソフトウェアの実行可能な UML モデルとして記述される．その振舞いは状態遷移図などで指定するが，そこに含まれるアクションなどのアトミックな処理のことを本論文ではステップと呼ぶ．ステップにはコード断片により詳細な振舞いが記述できるほか，後述するようにリファレンスプラットフォーム上でのステップの処理時間が性能パラメーターとしてステレオタイプによって付加される．

ステップの実際の処理時間は性能パラメーターからプロセッサーリソースの競合とリファレンスプラットフォームでのプロセッサーに対するプロセッサーの速度の違いを考慮して決定され，ステップが実行されるたびにシミュレーション時間が更新される．アプリケーションの処理時間はアプリケーションが終了した時点でのシミュレーション時間の変化の総量である．Rhapsody のシミュレーション時間の更新モデルはイベントドリブン型である．

3.3 性能パラメーターのアノーテーション



コード断片は C++言語であり，オブジェクト名->GEN()はそのオブジェクトへの非同期メッセージ送信を表す．

図 2 ILGenerator クラスの状態遷移図．

性能パラメーターとは上記のとおり，あるリファレンスプラットフォーム上でのモデルのステップの処理時間である．トレースドリブンシミュレーションにおいてはリファレンスプラットフォームを既存機種のプラットフォームとみなし，ステップの処理時間を計測によって取得する．これにより UML の記述粒度以下の細かいモデリングを避けつつ，信頼性のあるパラメーターを取得することが出来る．ただし，ステップに対応する実装コードは内部にループや条件分岐などの構造を含むため，その処理時間は一般的にはテストケース依存となる．シミュレーションによって計算される実際のステップの処理時間は，この数値に対し新しいプラットフォームのプロセッサー性能の違いや処理の競合の度合いによって補正されたものとなる．

性能パラメーターは MARTE Performance Analysis Modeling (PAM)のステレオタイプ<<PaStep>>をモデルのステップを表す要素に適用することで指定する．このステレオタイプにはhostDemandというタグがあり，ステップの処

理にかかる時間を数値で指定する．例を図 2に示す．図中の"generate_il"状態に<<PaStep>>ステレオタイプが付いている．現在のツールの実装では，状態に<<PaStep>>をつけた場合にはその状態のエントリーアクションをステップとして認識する．また現在の実装ではhostDemandには直接固定の数値を指定するほか，外部データファイルの数値をテストケースに依存して動的に与えることも出来る．

3.4 リソースアロケーションモデル

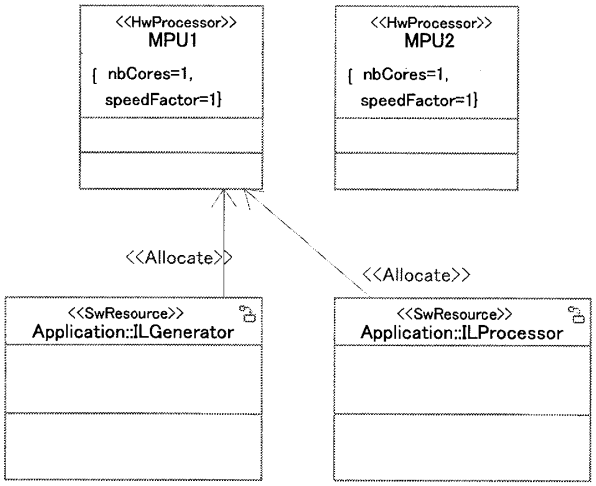


図 3 シングルプロセッサーアーキテクチャーの場合のリソースアロケーションを表すクラス図.

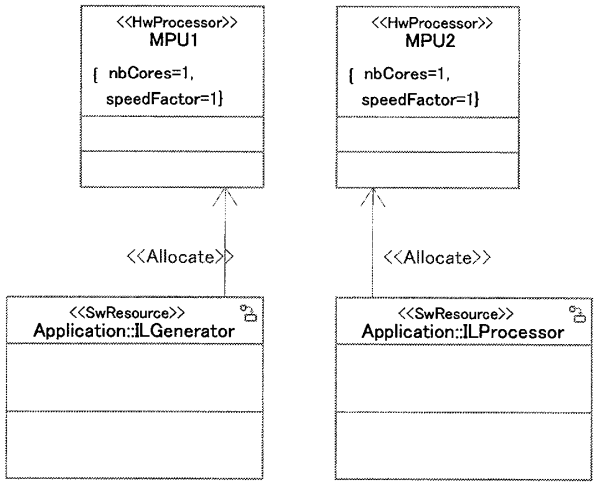


図 4 非対称マルチプロセッサー(AMP)アーキテクチャーの場合のリソースアロケーションを表すクラス図.

リソースアロケーションにおいては MARTE Detailed Resource Modeling コンセプトのステレオタイプを用いる．アプリケーションモデルのコンポーネントは<<SwResource>>ステレオタイプを適用し，プロセッサーリソースには<<HwProcessor>>ステレオタイプを適用する．

<<SwResource>>を適用されたコンポーネントはアプリケーションソフトウェアのリソース割り当てのひとつの単位として認識され，シミュレーション上はアクティブなスレッドを持つと解釈される．コンポーネントの内部構造や振舞いはアプリケーションモデルに記述される．

<<HwProcessor>>はその要素がプロセッサーであることを表す．ステップの処理時間に影響を与えるプラットフォームのリソースであり，その振舞いはプラットフォームモデルに定義され，ライブラリーとして提供される．プロセッサーの性能仕様は<<HwProcessor>>のタグで指定される．現在 speedFactor と nbCores タグに対応しており，それぞれリファレンスのプロセッサーからの処理スピード比とプロセッサー内のコアの数を意味している．

これらのコンポーネントどうしはMARTE Allocation Modelingのステレオタイプ<<Allocate>>が適用された依存関係で接続される．図 3および図 4はリソースアロケーションモデルの例である．図 3の場合，ILGeneratorとILProcessorという 2 つのソフトウェアコンポーネントがひとつのプロセッサーMPU1 を共有しており，図 4では 2 つのコンポーネントは並列に処理される．

アロケーションの記述にクラス図を使っているが，その意味は，アロケーションの対象となっている<<SwResource>>ステレオタイプの付いた要素の全てのインスタンスがアロケーション先のプロセッサーで実行されるということである．<<Allocate>>依存関係自体はクラス間だけではなく，例えばコンポジット構造図で記述されるインスタンス間に与えることも可能なので，本手法はインスタンス間のアロケーションに拡張することは可能である．

なお，<<Allocate>>依存関係による静的なプロセッサーリソース割り当てだけでなく，ラウンドロビンスケジューラーを用いた動的なリソース割り当ても実装しているが今回のケーススタディでは使用していないため説明を割愛する．

3.5 プラットフォームの振舞い

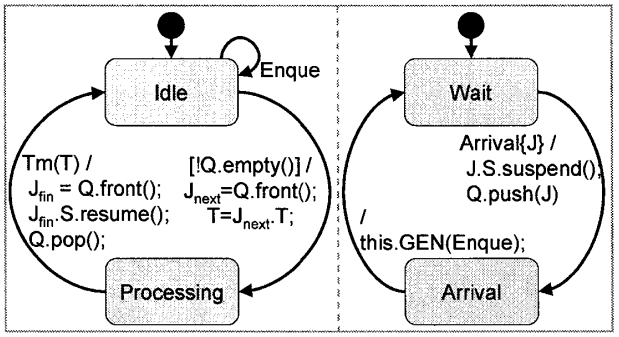
本研究ではCortellessaらが用いたのと同様のプロセッサーのモデルを採用している[10]．その振舞いを状態遷移図と擬似コードで表現したものを図 5に示す．

シミュレーションが実行されると実行可能 UML のセマンティクスに従いアプリケーションモデルの振舞いが実行されるが，実行フローが性能パラメーターのアノーテートされたステップに到達した場合，ジョブが生成される．ジョブとはステップのインスタンスであり，プロセッサーが処理すべきプログラム断片を抽象的に表したものである．ジョブ J = {T, S}はリファレンスプラットフォーム上での実行時間 T と，ジョブを生成したステップが所属するソフトウェアコンポーネントへの参照 S をパラメーターとして持つ．

ジョブはリソースアロケーションモデルでの指定に従って割り当てられたプロセッサーに送られる．プロセッサーでは到着したジョブを処理するたびにシミュレーション時間を進めることによってステップの実行時間を計算する．プロセッサーリソースの競合はキューイングによってシミュレートしている．プロセッサーはそれぞれキューを持ち，ジョブをキューに追加する際にソフトウェアコンポーネント S を一時停止する．キューの先頭の要素に対してプロセッサーの処理時間 T' = T / f を計算し，シミュレーション時間を T'だけ進める．ここで f はリファレンスプロセッサーに対するそのプロセッサーの処理速度の比(スピード係数)

であり，リソースアロケーションモデルの<<HwProcessor>>ステレオタイプが適用されたクラスのspeedFactor タグでユーザーが指定するものである．そしてS に対し実行の再開を通知しキューの先頭の要素を削除する．複数のプロセッサーが存在するため，それぞれのキューにおけるシミュレーション時間の更新は並列に行われる．

なお，キューイング以外のリソースの競合の評価方式としてジョブのタイムスライスのシミュレーションも実装しているが今回はケーススタディで使用していないため説明を割愛する．



Q はプロセッサーのキューを表し，front(), pop(), push()はそれぞれ先頭要素取得，先頭要素削除，末尾への追加の操作を，empty()は空かどうかの状態を表す．Tm(T)は時間 T 後にタイムアウトイベントが発生することを表す．Arrival{J}はジョブ J の到着イベントを表す．オブジェクト.GEN()はオブジェクトへのメッセージ送信を表す．中央の点線の左右は並列状態である．

図 5 プロセッサーの振舞いの状態遷移図．

3.6 シミュレーションコード生成

シミュレーションコード生成ツールは IBM Rational Rhapsody® ソフトウェアのプラグインとして実装した．プラグインは Java™ 言語で開発した．

シミュレーションコード生成の際に，リソースアロケーションモデルでのマッピングに従って，アプリケーションモデルとプラットフォームモデルの間の相互作用のためのコードが挿入される．この相互作用はジョブの生成や完了を通知するメッセージのやり取りとして実装される．これによりプラットフォームモデルは処理のスケジューリングに従ってアプリケーションモデルの実行を一時停止したり再開したりする．

Rhapsody のコード生成のフレームワークにおいてはモデルは中間的なモデルに変換された後，C++のコードに変換される．このプラグインはデフォルトのモデル変換の振舞いを変更し，MARTE の情報を中間モデルに反映するようにしている．

4. ケーススタディ

4.1 印刷処理システムの概要

この節では印刷システムのケーススタディについて述べる．印刷システムは外部のホストから Page Description Language (PDL，たとえば PostScript)の形式で印刷ジョブを受け取り，イメージが印刷された紙を出力する．ページごとに以下のステップが繰り返される:

1. IL 生成: システムは PDL を Image List (IL)とよばれる基本的な中間言語のコマンド列に変換する
2. IL 処理: IL をラスター化し，論理イメージに変換する
3. 印刷出力: 論理イメージを紙に物理的に出力する

このケーススタディの目的はシステムアーキテクチャーの変更によって印刷処理の時間がどれだけ改善されるかを見積もることである．ここでは以下の 2 つのアーキテクチャーを比較する:

- SP (Single Processor)アーキテクチャー: システムはひとつのプロセッサー上に実装される．IL 生成と IL 処理は逐次的に実行される．これは既存製品のアーキテクチャーである
- AMP (Asymmetric multi-processor)アーキテクチャー: システムは 2 つのプロセッサーを使って実装される．IL 生成と IL 処理は別々のプロセッサーで実行され，パイプライン処理を行う．なお 2 つのプロセッサーは，ハードウェア仕様は同じであるが処理の種類が異なっているため非対称(asymmetric)マルチプロセッシングとなる．

既存製品の解析から IL 生成と IL 処理はプロセッサーの使用率が高く，性能に強く影響する部分であることがわかっている．印刷出力のステップも同程度の処理時間がかかるが，メカニカルな振舞いがほとんどでありプロセッサーを使用しない．

今回この印刷処理システムを単純化したシステムのモデルを作成した．同時にこれらの 2 つのアーキテクチャーのプロトタイプを 2 つのプロセッサーコアを搭載した FPGA (Field Programmable Gate Array)ボード上にも実装し，シミュレーション結果を検証した．

性能パラメーターはSPアーキテクチャーのプロトタイプ機での実行トレース計測によって得た．実行トレースはアプリケーションプログラム中の関数の開始，完了を記録している．計測のオーバーヘッド自体が計測対象の実行時間に影響を与えないように，専用の入出力ポートに接続した外部ハードウェアで記録する手法を用いている[14]．今回の計測のオーバーヘッドは実行時間の 0.2%程度である．

4.2 アプリケーションモデル

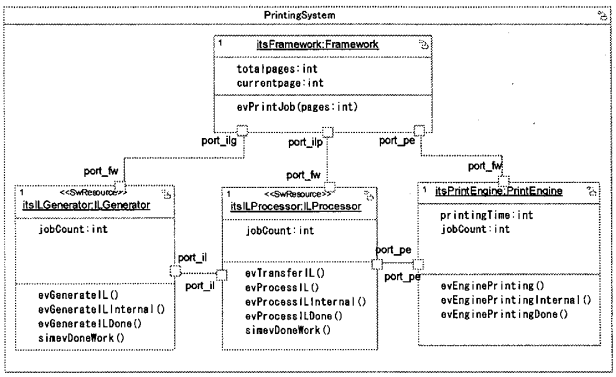


図 6 アプリケーションモデルの PrintingSystem クラスのコンポジット構造図．

図 6はシステムのアプリケーションモデルの全体構造を表したものである．印刷システム全体をPrintingSystemというクラスで定義した．設計文書の解析から 4 つのコンポーネントが導入されている:

- Framework: ジョブをディスパッチし印刷の流れを制御する．
- ILGenerator: PDL から IL を生成する
- ILProcessor: IL を処理してラスター画像に変換する．
- PrintEngine: ラスター画像を紙に印刷する．

実際のシステムではこのほかに多数のコンポーネントが存在するがここでは理解を容易にするため性能面に関係のないコンポーネントは省略している．

モデルに与える性能パラメーターは以下の 2 つである:

- 1 ページ分の IL 生成にかかる時間
- 1 ページ分の IL 処理にかかる時間

これらはSPアーキテクチャーのFPGAプロトタイプ機の実測から得られたものであり，モデルのコンポーネントの振舞いに対し図 2のようにアノーテーションとして記述する．

PrintEngine コンポーネントにおける紙への印刷出力処理は単純に定数時間の待ちに置き換えている．

4.3 リソースアロケーションモデル

SPアーキテクチャーとAMPアーキテクチャーのリソースアロケーションをそれぞれ図 3，図 4に示す．どちらの場合もプロセッサーコア数とスピード係数の変化はない．

なお，今回SPアーキテクチャーのモデルからAMPアーキテクチャーのモデルを得るのに必要だった作業は図 3を図 4に書き換える際の<<Allocate>>依存関係の変更だけである．

4.4 プロトタイプ機

今回使用したプロトタイプ機は Xilinx 社の FPGA ボードとして実装されたもので，PowerPC® 440 (450MHz)プロセッサー，メモリー(Dual Inline Memory Module, DIMM)，およびバス(100MHz)それぞれ 2 つずつで構成されている．また，実行トレース計測用の入出力ポートも持っている．

2 つのメモリーはそれぞれのプロセッサーに割り当てられており，Operating System (OS)として Linux がそれぞれのプロセッサーで動作し，アプリケーションプログラムを実行する．コンフィグレーションの切り替えによりシングルプロセッサーのみで実行した場合が既存製品の構成に，デュアルプロセッサーで実行した場合が新しい製品の構成に対応する．

アプリケーションプログラムはもともとシングルプロセッサー用に開発されたものだが，デュアルプロセッサーの AMP 構成で動作するように，一部のデータの参照をプロセッサー間通信に変更し，移植した．

4.5 テストケース

今回用いたテストケースは電子情報技術産業協会(Japan Electronics & Information technology Industries Association, JEITA)で定められた標準ベンチマーク[15]から作成した．

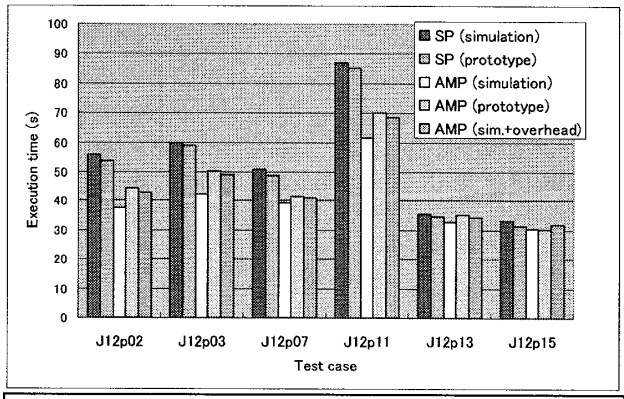
このベンチマークから 6 種類のページデータを使用した．これらは以下のラベルが付いている: J12p02，J12p03，J12p07，J12p11，J12p15.

これらのページデータは次の観点から選ばれたものである．本システムが4.1節で述べたようにIL生成とIL処理のパイプライン処理であるため，その効率は両者にかかる時間の大小関係で決まる．そこでIL生成がIL処理より長いデータ(J12p03，J12p11)，IL生成とIL処理がほぼ同じデータ(J12p02，J12p07)およびIL生成がIL処理より短いデータ(J12p13，J12p15)を選択した．

ひとつのテストケースは連続した 4 ページ分の同じページデータからなる．

なお，今回のケーススタディでは使用していないが，我々の実装では3.3節でも触れたように，性能パラメーターには数値の定数だけではなく，変数名を指定することができ，外部ファイルで指定される数値データをステップの実行のたびに供給することもできる．そのため，同じページデータの連続印刷以外にも，異なるページデータを組み合わせた連続印刷のシミュレーションも可能となっている．

4.6 シミュレーション結果



Y 軸は印刷の実行時間であり，X 軸はテストケースに用いた JEITA ベンチマークのページデータのラベルである(4.5節参照).

図 7 テストケースごとの印刷システムの実行時間.

開発したシミュレーションコード生成ツールを用いてSPアーキテクチャーとAMPアーキテクチャーのシミュレーションプログラムを作成した．これらの出力した実行時間の見積もりを図 7に示す．プロトタイプ機の実行時間の実測結果と，後述するプロセッサー間通信によるオーバーヘッドを加味したシミュレーション見積もりも同じ図 7に示している．

SP アーキテクチャーに対して AMP アーキテクチャーの実行時間の相対的な改善量はテストケースに依存し，シミュレーションでは 7.3%から 32.5%だったのに対し，プロトタイプ機では-1.9%から 17.6%であった．

5. 議論

SPアーキテクチャーのシミュレーション結果とプロトタイプ機の実測との差は図 7のデータより最大でも 5.3%であり，AMPアーキテクチャーにすることによって実行時間が削減される点ではシミュレーション，プロトタイプ機ともに同じ傾向を示しているが，AMPアーキテクチャーでの実行時間は常にシミュレーションの予測値のほうが小さくな

っている．今回のテストではAMPアーキテクチャーにおけるシミュレーションとプロトタイプ機の相対的な差は最大で 15.9%であった．この理由はシミュレーションモデルの中に取り入れてない効果があるためである．

このうちもっとも主要な効果はAMP化によって追加されたプロセッサー間通信の時間である．ILGeneratorからILProcessorへのILの転送の間，後続のIL処理と次のIL生成が停止してしまうためこの間プロセッサー処理能力が利用されない．図 7にこのデータ転送のオーバーヘッドによる効果を取り込んだシミュレーション結果(AMP (sim.+overhead))も示している．ここで 1 ページあたりのデータ転送の時間はモデルのパラメーターであり，AMPアーキテクチャーのプロトタイプ機のデータ転送部分の挙動を計測することにより得た．その結果，AMPアーキテクチャーにおいてもシミュレーションとプロトタイプの差は最大で 5.8%と，SPアーキテクチャーにおける差と同程度におさまっている．

したがって，プロセッサーの並列化によるプロセッサー計算時間の変化については本手法で正しく再現できていると考えられるが，ここで行ったモデリングは本研究で想定しているユースケースにおいては不自然である．なぜならばこのシミュレーションでは 1 ページあたりの IL データの転送時間をパラメーターとして含んでいるが，その値は AMP アーキテクチャーのプロトタイプ機の計測によって得たものであり，一方シミュレーションは AMP アーキテクチャーの製品を実装する前に行わなければならないからである．したがって，本手法では並列化による新たなプロセッサー間通信など新しいアーキテクチャー固有のステップの処理時間は類似システムや実験プログラムの処理時間など，別の方法で見積もる必要がある．

6. 結論

本研究では MARTE プロファイルを適用した UML モデルのシミュレーションによる組込みシステムの性能評価手法を提案した．

ユーザーはアプリケーションモデルとリソースアロケーションモデルを作成する．性能パラメーターを MARTE PAM のステレオタイプを用いてモデルに付加する．同様にハードウェアリソースの仕様を MARTE DRM のステレオタイプを用いて指定し，ソフトウェアリソースに対し MARTE Allocation Modeling のコンセプトを用いて割り当てる．

今回開発したシミュレーションコード生成ツールはこれらのアノーテーションを反映し，ユーザーのモデルをライブラリーとして提供されるプラットフォームモデルと結合し，シミュレーションコードに変換する．シミュレーションを実行することによりアプリケーションの実行時間の見積もりを得る．

これにより UML モデルを用いて少ない変更量で複数アーキテクチャーでの性能比較を可能にする．またシミュレーションの粒度は UML の記述の粒度であり，評価結果を UML による設計と関連付けやすい．

ケーススタディでは印刷システムに対してシミュレーションとプロトタイプ機の実測結果を比較し，プロセッサー処理時間に関してはアーキテクチャーを変更した場合でも6%程度の精度で予測することが出来たが，プロセッサー間通信の時間の評価に課題が残った．

将来課題としては，プロセッサー以外のハードウェアリソースの振舞いのサポートがある．特にバスによる通信時間の評価はマルチプロセッサーアーキテクチャーのシステムの性能評価に必要である．バスの影響を厳密に評価するには詳細なメモリーアクセスのタイミング情報が必要である．それを再現するモデルは UML よりずっと詳細でかつ低速なシミュレーションとなる可能性があり，今回目指しているシステムアーキテクチャー設計段階での UML モデリングの方向性とは異なってしまう．そのため何らかの近似を行って粗い粒度のモデリングを維持する必要がある．

また，今回のケーススタディで行ったアーキテクチャー変更は，コンポーネントの独立性が高く，並列化の粒度がコンポーネントの粒度に一致していたため，4.3節で述べたように極めて少ない作業量で行うことが出来た．今後はより一般的な並列化作業を題材としてMARTEによるモデリングの有効性を検証する必要があるだろう．

最後に，本シミュレーション手法はモデルが要求を満たすかを開発の上流工程において検証するために用いられることになるが，中でも時間制約の検証が主要な用途となりうる．時間制約は一般に多数かつ複雑になりうるため，作業の効率化および高速化が必要である．筆者らはこの点についても研究を進めており[16]，実際のシミュレーションとの併用は今後の課題である．

謝辞

著者らは技術的な議論を行ったマルチコアプロジェクトのメンバーに感謝いたします．

参考文献

[1]P. J. Fortier and H. E. Michel, “Computer Systems Performance Evaluation and Prediction,” Digital Press, (2003).

[2]S. Balsamo, A. D. Marco, P. Inverardi, and M. Simeoni, “Model-based performance prediction in software development: a survey,” IEEE Transactions on Software Engineering, vol. 30, no. 5, pp. 295–310, (2004).

[3]Object Management Group, “MARTE specification.” (2009).

[4]Object Management Group, “Model Driven Architecture Guide.” (2003).

[5]Object Management Group, “OMG Systems Modeling Language (SysML) specification.” (2008).

[6]K. Ono, M. Toyota, R. Kawahara, Y. Sakamoto, T. Nakada and N. Fukuoka, “A modeling method for performance analysis in model-driven development,” in Proceedings of the 13th Design, Automation & Test in Europe (DATE 2010), (2010).

[7]高 鏞守, 久住 憲嗣, 小野 康一, 河原 亮, 坂本 佳史, 中田 武男, “組込みソフトウェアにおける実行トレースに基づく捨象モデリ

ングツールの実装と評価," 組込み技術とネットワークに関するワークショップ (ETNET2010)会議録, 2010
[8]M. Woodside, "From annotated software designs (UML SPT/MARTE) to model formalisms," in Proceedings of SFM2007 (M. Bernardo and J. Hillston, eds.), LNCS vol. 4486, pp. 429–467, Springer-Verlag, (2007).
[9]S. Balsamo and M. Marzolla, "Performance evaluation of UML software architectures with multiclass queueing network models," in Proceedings of the 5th international workshop on Software and performance (WOSP'05), pp. 37–42, ACM, (2005).
[10] 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).
[11] É. Piel, R. B. Atitallah, P. Marquet, S. Meftali, S. Niar, A. Etien, J.-L. Dekeyser, and P. Boulet, "Gaspard2: from MARTE to SystemC simulation," in Proceedings of DATE'08 workshop on Modeling and Analysis of Real-Time and Embedded Systems with the MARTE UML profile, (2008).
[12] Open SystemC Initiative, "SystemC Specification." (2007).
[13] J. Vidal, F. de Lamotte, G. Gogniat, P. Soulard, and J.-P. Diguet, "A co-design approach for embedded system modeling and code generation with UML and MARTE," in Proceedings of the 12th Design, Automation & Test in Europe (DATE 2009), (2009).
[14] N. Ohba and K. Takano, "Hardware debugging method based on signal transitions and transactions," in Proceedings of the 11th Asia South Pacific Design Automation Conference (ASP-DAC 2006), pp. 454–459, January (2006).
[15] Japan Electronics and Information Technology Industries Association (JEITA), JEITA Printer Benchmark Test Patterns.
[16] 小野康一, 中村宏明 and 石川浩, "時間/機能制約による仕様に対する実行可能な UML/SysML モデルの動的検査手法," コンピュータソフトウェア Vol. 27, No. 2, pp.33-49, (2010)