Vol.2009-SE-166 No.14
2009/11/6

モデル駆動による組込みシステムの性能検証手法の提案

磯田　誠[†]　　田村直樹[†]

組込み制御システムの開発では，多機能化・高性能化の要求に加えて，ネットワーク接続したシステム間でのリアルタイム制御の要求が高くなっている．しかし現時点では，実行環境／ネットワーク特性を考慮した実用的な性能検証技術は確立していないため，実システムを対象にした性能検証，および検証結果の正確なフィードバックが困難という課題がある．
これに対して我々は，システム設計結果に情報を直接付加して性能検証モデルを作成し，このモデルを利用してシミュレーション検証するアプローチに取り組んでいる．本稿では，上記のアプローチをモデル駆動による性能検証手法として提案する．また，航空管制システムを題材にモデル作成とシミュレーション検証を行い，本手法のフィージビリティを確認した結果を示す．

An Approach of Performance Engineering for Embedded Systems Using Model Driven Technology

Makoto Isoda[†]　and Naoki Tamura[†]

Embedded control systems have to fulfill high level requirements for functionality and performance. Moreover, these systems are required to meet requirements for real-timeliness even if they are interconnected by communication networks. Currently, however, performance engineering methods do not take systems' execution environments or communication networks into account, so that it is difficult to model performance aspects of real systems and feedback results of analysis correctly.
In this paper, we propose a performance engineering method using model driven technology. This method allows us to make performance models by annotating performance parameters upon system design models directly, and analyze performance characteristics by using simulation methods. We show trial results of modeling and simulation on an air traffic control system specification. And we evaluate feasibility of our method based on the trial results.

1. はじめに

組込み制御システムは，自動車，航空機，人工衛星，FA（Factory Automation）といった分野で広く利用されており，社会活動における重要性がますます高まっている．また，業務の拡大やユーザニーズの向上により，多機能・高性能が要求されている．

組込み制御システムの開発では，多機能化・高性能化の要求に応えるために，システム構成の大規模化，および制御方式の複雑化が進んでいる．さらに近年は，ネットワーク接続したシステム間でのリアルタイム制御の要求が高くなっている．

しかし現時点では，機器単体でのハードウェア性能はある程度正確に見積もれるが，実行環境／ネットワーク特性を考慮した実用的な性能検証技術は確立していない状況にある．そのため，実システムを対象にした性能検証，および検証結果の正確なフィードバックが困難という課題がある．

これに対して我々は，システム設計結果に情報を直接付加して性能検証モデルを作成し，このモデルを利用してシミュレーション検証するアプローチに取り組んでいる．本稿では，上記のアプローチのフィージビリティを検討し，モデル駆動による性能検証手法として提案する．また，航空管制システムを題材に，本手法の具体的な適用例を示す．

本稿の構成は以下のとおりである．2 章では，本研究の関連技術を示す．3 章では，性能検証の課題と我々のアプローチを示す．4 章では，我々が提案する性能検証手法を定義する．5 章では，題材を用いた本手法の適用例を示す．6 章では，本手法の評価結果を示す．7 章では，本稿のまとめを示す．

2. 性能検証の関連技術

本章では，システムの性能検証に関連する技術の概要を示す．

2.1 待ち行列ネットワーク

待ち行列ネットワークは，ネットワーク接続した計算機の性能を解析的に検証するために利用されるモデルである[1]．単一の計算機を対象にした基本的な待ち行列モデルを，ネットワーク接続した構成で検証できるように拡張している．

Layered Queuing Network（以降，LQN）は，システムの振舞いをアプリケーション，プロセス／スレッド，プロセッサを用いて階層的に記述し，性能を解析的またはシミュレーションにより検証するために利用されるモデルである[2].

2.2 離散系イベントシミュレーション

離散系イベントシミュレーションとは，因果関係（順序，タイミング）にしたがったシステム上での離散的なイベントの発生を，計算機を用いて模擬する方法である．待ち行列ネットワークなどを用いた検証の処理系ツールとして利用される．既存のシミュレーション技術を表 1 に示す．

表 1　既存のシミュレーション技術

項目	記法	ツールライセンス	ドキュメント
SimPy (Simulation in Python)[3]	Python	LGPL	豊富
JSL (Java Simulation Language)[4]	Java	GPL	貧弱
PARSEC (Parallel Simulation Environment for Complex Systems)[5]	拡張C言語	有償	豊富
CSIM[6]	拡張C言語	有償	豊富

2.3 組込み向けリアルタイム性の UML プロファイル

UML Profile for Modeling and Analysis of Real-Time Embedded systems（以降，MARTE Profile）は，組込みリアルタイムシステムの開発において，リアルタイム性の設計・分析モデリングに利用するための UML2 拡張記法である[7][8]．MARTE Profile では，性能検証のための情報の記法を規定しているが，具体的な検証手法は特定していない．MARTE Profile の構成を表 2 に示す．

表 2　MARTE Profile の構成

項目	説明
MARTE foundations	組込みリアルタイムシステムの基本要素，時間，割り当ての記法
MARTE design model	システム設計のためのアプリケーション，ソフトウェアリソース，ハードウェアリソースの記法
MARTE analysis model	性能分析，スケジューラビリティ分析で必要となる情報をモデルに付加するための注釈記法

3. 性能検証の課題とモデル駆動によるアプローチ

本章では，性能検証の課題とアプローチ，我々が提案する性能検証手法を示す．

3.1 性能検証の課題

従来のアプローチでは，待ち行列モデルによるシミュレーション技術が利用されることが多い．具体的には，システム設計結果から主にハードウェアに着目した待ち行列モデルを作成し，シミュレーションを実行することでハードウェア性能を検証する．また，検証結果が要求未達成の場合，システム設計結果と待ち行列モデルの対応を分析して，設計変更が必要な箇所を特定する．

従来のアプローチの課題を以下に示す．本稿では課題(1)に取り組む．

(1) システムの複雑化により，実システムの設計結果から誤りなく待ち行列モデルを作成すること，および性能検証結果をシステム設計に正確にフィードバックすることが困難．
(2) システムの大規模化により，実システムを対象にした待ち行列モデル作成とシミュレーション検証の工数が増加．

3.2 モデル駆動による性能検証のアプローチ

我々のアプローチでは，システム設計結果に情報を直接付加して性能検証のためのモデル（以降，性能検証モデル）を作成し，性能検証モデルをシミュレーションモデルに変換して実行することで性能を検証する．また，性能検証モデル上で設計変更が必要な箇所を直接特定する．モデル駆動による性能検証のアプローチを図 1 に示す．

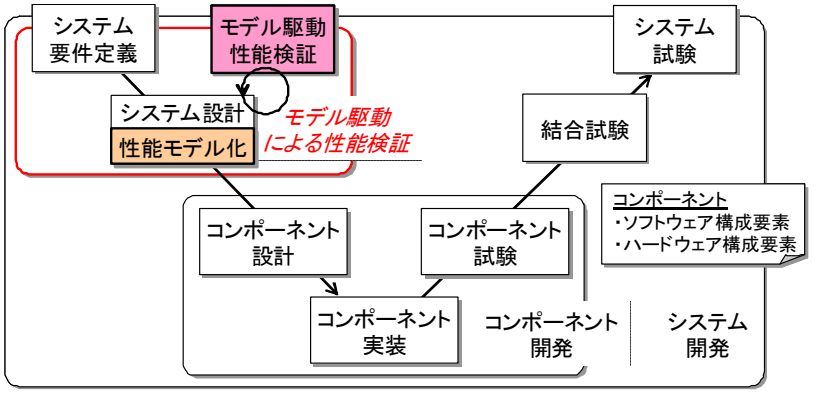

図 1　モデル駆動による性能検証のアプローチ

本アプローチの狙いを以下に示す．

(1) システム設計結果，性能検証モデル，シミュレーションモデルの対応付けを容易にすることで，性能検証モデルを誤りなく作成すること，およびシミュレーション検証結果をシステム設計に正確にフィードバックする．
(2) アプリケーションの実行環境とネットワークのモデルをライブラリ化し，性能検証ソフトウェアとして提供することで，検証作業の工数削減．

本稿では，上記の(1)に関して，我々のアプローチのフィージビリティを検討し，モデル駆動による性能検証手法として提案する．検証指標は表 3 に示す応答時間，リソース利用率とする[9]．

表 3　検証指標

項目	説明
応答時間	システムに入力された要求メッセージに対する応答が出力されるまでの平均時間
リソース利用率	各ソフトウェア／ハードウェアリソースが，アプリケーションの処理を実行している時間と実行していない時間の比率

3.3 我々が提案する性能検証手法の概要

我々が提案する性能検証手法の概要を，従来の手法との対比を含めて図 2 に示す．

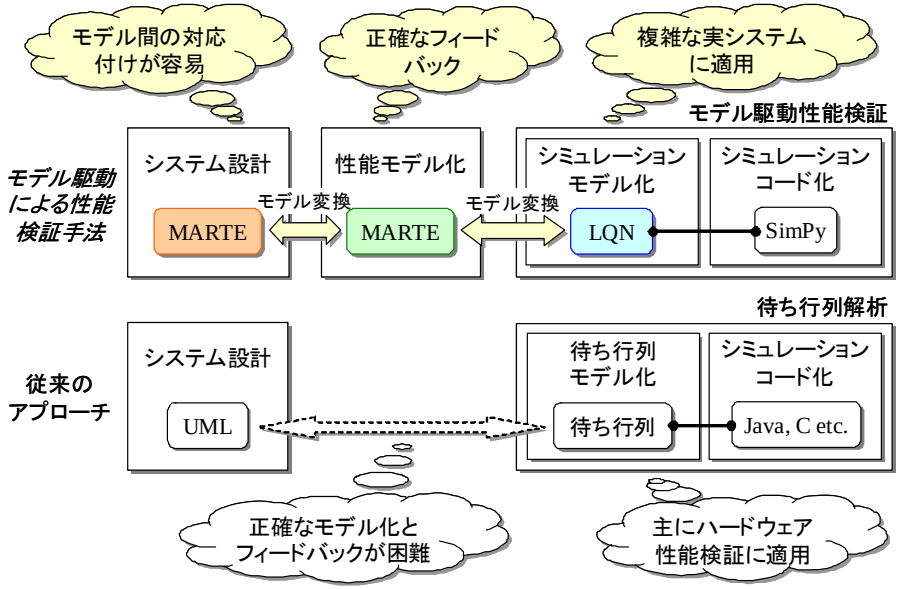

図 2　性能検証手法の概要と特徴

本手法では，まず各フェーズで作成するモデルの構成要素の意味と記法を規定する．そして，モデル変換規則にしたがって，あるフェーズのモデルを基に次フェーズのモデルを作成する．本稿で採用するモデルの特徴を以下に示す．

(1) MARTE Profile は，実開発で普及している UML に対してリアルタイム性の記法を拡張しており，組込みシステム開発での仕様記述に適している．
(2) LQN は，通常の待ち行列ネットワークに対して並行動作と通信動作を拡張しており，複雑な実システムのモデル化に適している．
(3) SimPy は，記法がオブジェクト指向で UML との親和性が高く，ドキュメントが豊富であり，シミュレーションコードに変換する際に誤りが入りにくい．

4. モデル駆動による性能検証手法

本章では，開発の各フェーズで作成するモデルの定義，モデル間の対応関係を示す．

4.1 システム設計モデルの定義

図 2 に示したシステム設計フェーズで作成するシステム設計モデルを定義する．システム設計モデルを定義する観点として，表 4 に示すシステムアーキテクチャ記述のビューポイント[10]を利用する．

表 4　システムアーキテクチャ記述のビューポイント

ビューポイント	説明	UML モデルの例
機能的	機能要素，インタフェース，コネクタ，外部エンティティを定義	クラス図 合成構造図
情報	システムの構成要素間のデータの流れを定義	アクティビティ図
並行性	構成要素の状態と状態遷移，実行時の動作シーケンスを定義	ステートマシン図 シーケンス図
開発	プログラムコードの構造，保守・拡張方法を定義	特になし
配置	ハードウェア／ネットワークノードとソフトウェアの対応を定義	配置図
運用	システムの監視制御・保守方法を定義	特になし

今回我々が対象とする組込みシステムにおいて，リアルタイム制御の要件を実現するには，機能的および並行性ビューポイントが最も重要になると考えられる．したがって，システム設計フェーズでは，モデルとして UML のクラス図，ステートマシン図，合成構造図，シーケンス図を作成することとする．

システム設計モデルは MARTE Profile を用いて作成する．MARTE Profile を用いたシステム設計モデルの概要を図 3 に示す．

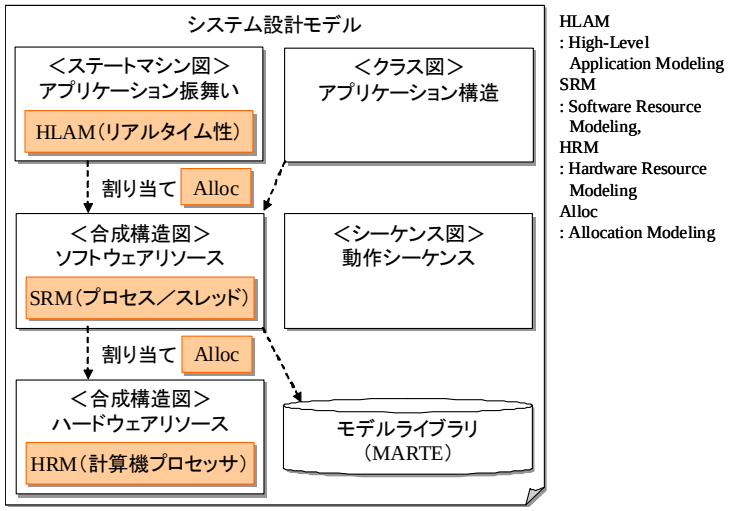

図 3　システム設計モデルの概要

図 3 に示したシステム設計モデルの特徴を以下に示す．

(1) アプリケーション構造を UML クラス図で記述.
(2) アプリケーション振舞いを UML ステートマシン図で記述. MARTE Profile の HLAM（High-Level Application Modeling）を用いて，リアルタイム性を持った振舞いを記述.
(3) ソフトウェアリソースを UML 合成構造図で記述. MARTE Profile の SRM（Software Resource Modeling）を用いて，アプリケーションを実行するプロセス／スレッドを記述.
(4) ハードウェアリソースを UML 合成構造図で記述. MARTE Profile の HRM（Hardware Resource Modeling）を用いて，計算機のプロセッサを記述.
(5) 上記のモデルに含まれる要素間の割り当て関係を，MARTE Profile の Alloc（Allocation Modeling）を用いて記述.
(6) システムの動作シーケンスを UML シーケンス図で記述.

4.2 性能検証モデルの定義

図 2 に示した性能モデル化フェーズで作成する性能検証モデルを定義する. 性能検証モデルは，MARTE Profile を用いてシステム設計モデルに情報を直接付加して作成する. MARTE Profile を用いた性能検証モデルの概要を図 4 に示す.

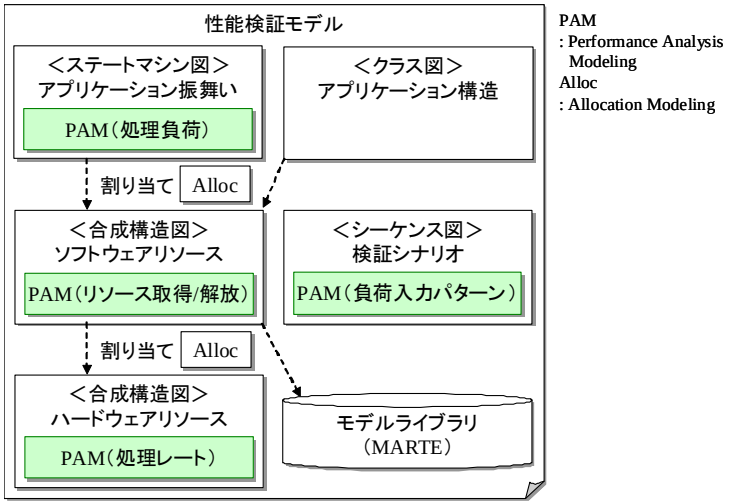

図 4　性能検証モデルの概要

図 4 に示した性能検証モデルの特徴を以下に示す.
(1) アプリケーション構造，モデル要素間の割り当て関係は，システム設計モデルをそのまま利用.
(2) アプリケーション振舞いに対して，MARTE Profile の PAM（Performance Analysis Modeling）を用いて処理負荷（処理時間，処理回数）の情報を付加.
(3) ソフトウェアリソースに対して，MARTE Profile の PAM を用いてリソース取得／解放の処理時間を付加.
(4) ハードウェアリソースに対して，MARTE Profile の PAM を用いて処理レートを付加.
(5) システムの動作シーケンスに対して，MARTE Profile の PAM を用いて負荷入力パターン（入力の順序とタイミング）を記述し，検証シナリオとする.

4.3 シミュレーションモデルの定義

図 2 に示したモデル駆動性能検証フェーズで作成するシミュレーションモデルを定義する. シミュレーションモデルは，LQN のモデルと SimPy のシミュレーションコードを組み合わせる. LQN を用いたシミュレーションモデルの概要を図 5 に示す. なお，LQN と SimPy の組み合わせ方は 4.4 に示す.

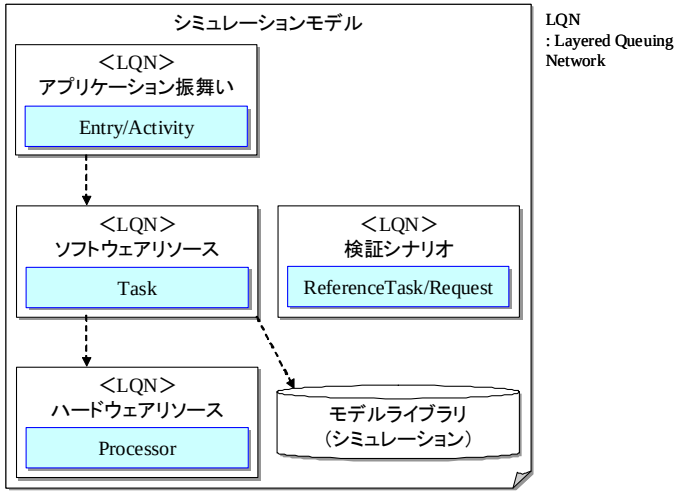

図 5　シミュレーションモデルの概要

図 5 に示したシミュレーションモデルの特徴を以下に示す.
(1) シミュレーションモデルは検証対象システムの振舞いに着目して作成するので，アプリケーション構造は記述しない.
(2) アプリケーション振舞いを，要素が提供するサービスを表す Entry と，振舞いの詳細なステップを表す Activity を用いて記述.
(3) ソフトウェアリソースを，排他制御が必要なソフトウェア資源を表す Task を用いて記述.

(4) ハードウェアリソースを，排他制御が必要なハードウェア資源を表す Processor を用いて記述.
(5) 検証シナリオを, 検証対象システムへの入力を表す ReferenceTask を用いて記述.

4.4 モデルの作成目的とモデル変換規則

4.1～4.3 で定義したシステム設計モデル，性能検証モデル，シミュレーションモデルの作成目的を表 5 にまとめる.

表 5 モデルの作成目的

項目	説明
システム設計モデル	システムに対する要件を実現するために，さまざまな観点からシステムを分析し，次フェーズのコンポーネント開発に対する要求を仕様化
性能検証モデル	性能検証のための各種パラメータを定義し，シミュレーションモデルとの双方向の対応を明確化
シミュレーションモデル	既存のシミュレーション技術に対して，検証指標を計算するための入力情報を記述

また，表 5 に示したモデル間の変換規則を表 6 に示す．表 6 のシステム設計モデルと性能検証モデルの欄では，日本語表記は UML の要素，英語表記と括弧内の記述は MARTE Profile のステレオタイプを表している.

表 6 モデル変換規則

項目	システム設計モデル	性能検証モデル	シミュレーションモデル（LQN／SimPy）
アプリケーション構造	クラス	クラス	－(Task に統合)
アプリケーション振舞い	RtBehavior (提供サービス)	PaStep (提供サービス)	Entry／Process
	RtBehavior (詳細ステップ)	PaStep (詳細ステップ)	Activity／Process
ソフトウェアリソース	SwSchedulableResource	PaRunTInstance	Task／Resource
	MessageComResource	MessageComResource	Task のキュー／Store
	SwMutualExclusion Resource	SwMutualExclusion Resource	Task／Resource
ハードウェアリソース	HwProcessor	GaExecHost	Processor／Resource
割り当て	Allocate	Allocate	－(Task－Proc 間関係)
検証シナリオ	－	GaAnalysisContext	パラメータと変数
	－	GaScenario	検証指標
	システムに対する外部入力	GaWorkloadEvent	ReferenceTask／Process
	要素間のメッセージ	PaStep	Request／Process

5. 題材を用いた性能検証手法の適用例

本章では, 航空管制システム（Air Traffic Control System, 以降 ATCS）[11]を題材に, 4 章で定義した性能検証手法の適用例を示す.

5.1 想定するシステム設計モデル

ATCS のシステム設計モデルとして，構成要素とその間の関連を表すクラス図を図 6 に示す.

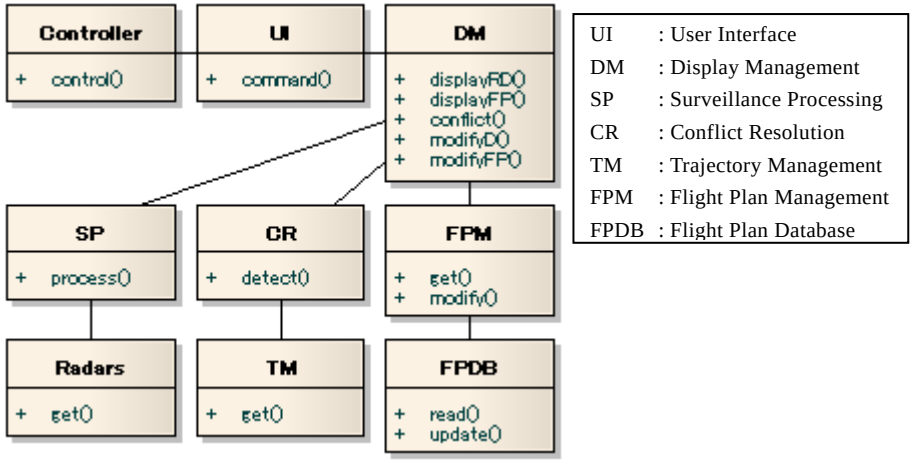

図 6 ATCS のシステム設計モデルーアプリケーション構造

ATCS の構成要素ごとの状態遷移の振舞いの一部として，DM（Display Management）のステートマシン図を図 7 に示す. 図 7 では，リアルタイム性を持った遷移アクションに<<rtBehavior>>を付加している．他のステートマシン図も同様に作成する.

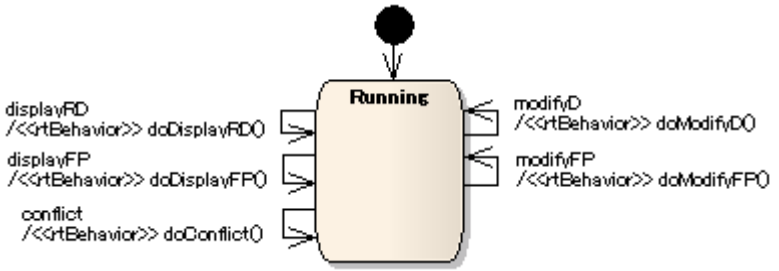

図 7 ATCS のシステム設計モデルーアプリケーション振舞い（一部）

ATCS が備えるソフトウェアリソース（<<swSchedulableResource>>，<<messageComResource>>），ハードウェアリソース（<<hwProcessor>>），これらの間の割り当て（<<allocate>>）を表す合成構造図を図 8 に示す．

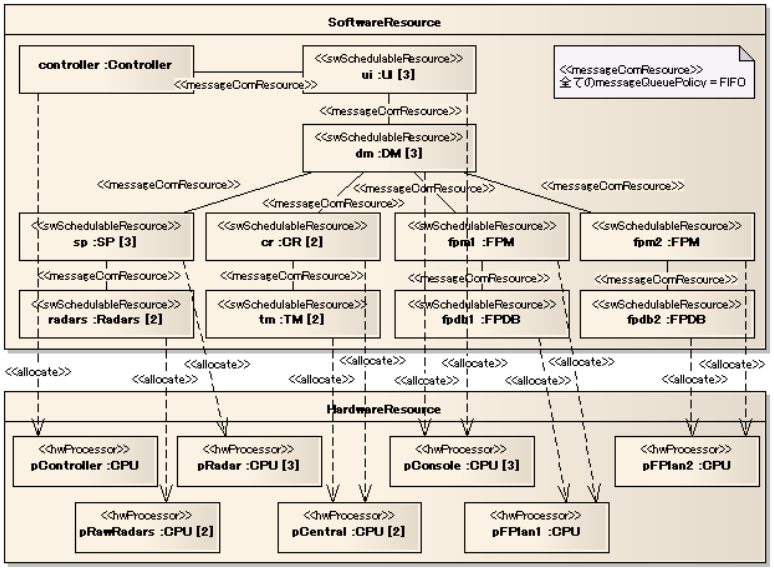

図 8　ATCS のシステム設計モデルーリソース間の割り当て

Controller を起点とした同期メッセージのやり取りの一部を表すシーケンス図を図 9 に示す．

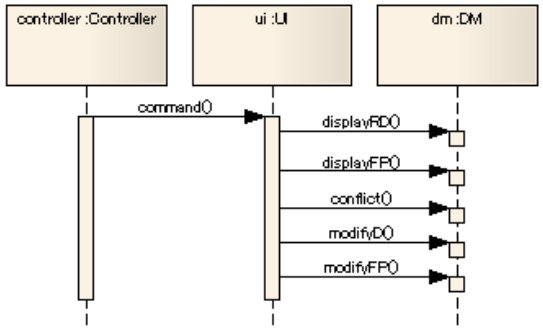

図 9　ATCS のシステム設計モデルー動作シーケンス（一部）

5.2 性能検証モデルの作成

MARTE Profile を利用して ATCS のシステム設計モデルに性能検証のための情報を付加して，性能検証モデルを作成する．図 6 のクラス図には付加する情報がないので，システム設計モデルをそのまま利用する．

図 9 に性能検証のパラメータ（<<gaAnalysisContext>>），検証指標（<<gaScenario>>），メッセージ送信の繰り返し回数（<<paStep>>）を付加した検証シナリオの一部を図 10 に示す．

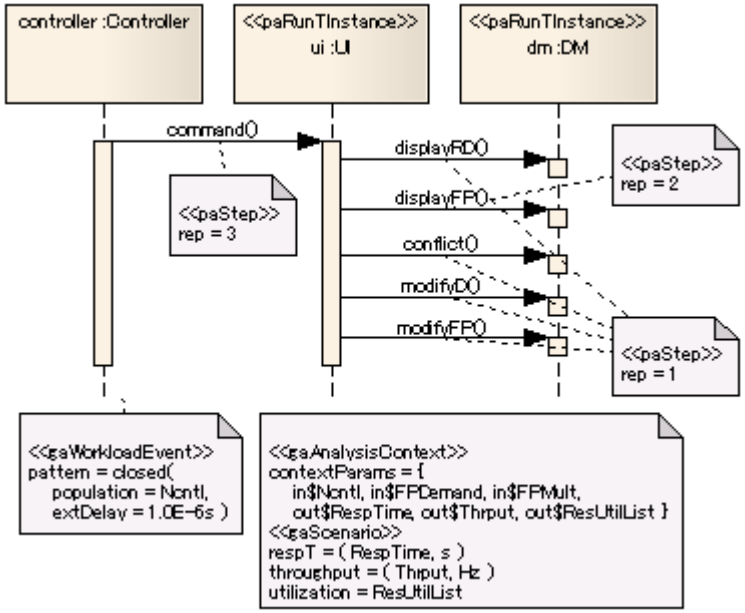

図 10　ATCS の性能検証モデルー検証シナリオ（一部）

図 8 のソフトウェアリソース部分では，fpm1，fpdb1，fpm2，fpdb2 に，性能検証のパラメータとなる可変な多重度（<<swSchedulableResource>>の resMult 属性）を付加する．同じくハードウェアリソース部分では，性能検証のパラメータとなる可変な多重度（<<hwProcessor>>の resMult 属性）を付加する．

図 7 の<<rtBehavior>>を付加した遷移アクションに，処理時間（<<paStep>>）を付加したステートマシン図を図 11 に示す．

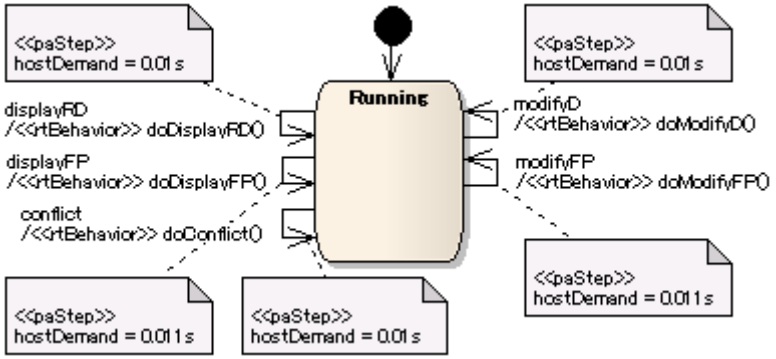

図 11　ATCS の性能検証モデルーアプリケーション振舞い（一部）

5.3 シミュレーションモデルへの変換

表 7 に示す変換規則にしたがって，性能検証モデルを機械的にシミュレーションモデル（LQN／SimPy）に変換する．変換後のシミュレーションモデルは，文献[11]に記載の範囲で同等のものとなる．

表 7　性能検証モデルとシミュレーションモデルの変換規則

項目	性能検証モデル	シミュレーションモデル（LQN／SimPy）
アプリケーション構造	クラス	－(Task に統合)
アプリケーション振舞い	<<paStep>>	Entry／Process
ソフトウェアリソース	<<paRunTInstance>>	Task／Resource
	<<messageComResource>>	Task のキュー／Store
ハードウェアリソース	<<gaExecHost>>	Processor／Resource
割り当て	<<allocate>>	－(Task－Proc 間関係)
検証シナリオ	<<gaAnalysisContext>>	パラメータと変数
	<<gaScenario>>	検証指標
	<<gaWorkloadEvent>>	ReferenceTask／Process
	<<paStep>>	Request／Process

5.4 シミュレーション検証の結果

我々が定義した性能検証手法の正しさを確認するため，シミュレーション検証を行い，文献[11]の結果と比較検討する．パラメータは Controller 数を表す Ncntl，FPM／FPDB の処理時間を表す FPDemand [sec]（プロセッサ速度に相当），FPM／FPDB のリソース数を表す FPMult（スレッド数に相当）とする．Ncntl = 3, FPDemand = 0.02, FPMult = 2 の場合のシミュレーション検証結果と文献[11]の計算結果を表 8 に示す．

表 8　シミュレーション検証結果－No.1

項目	平均応答時間 [sec]	95%信頼区間 [sec]
本手法	1.807	±0.0034
文献[11]	1.887	±0.0059

次に，Ncntl = 2～8，FPDemand = 0.02，FPMult = 1 の場合のシミュレーション検証結果として，応答時間とリソース利用率を図 12 に示す．図 12 では，リソース利用率が高い fpm1（Flight Plan Management）と fpdb1（Flight Plan Database）の値を示す．

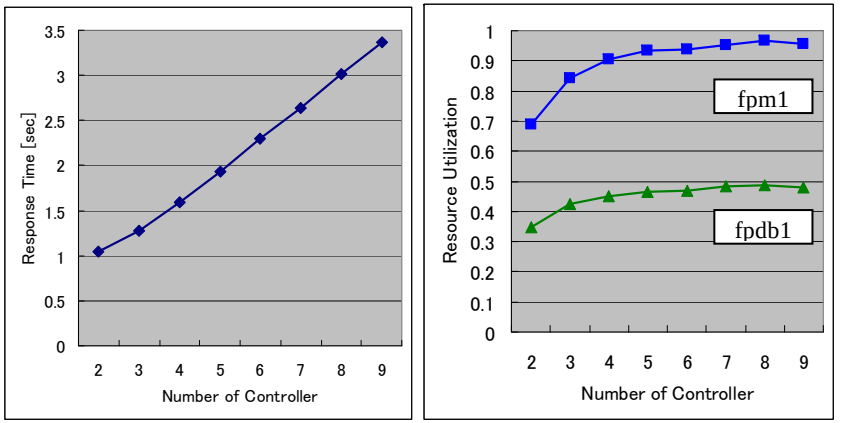

図 12　シミュレーション検証結果－No.2

さらに，Ncntl = 7 とし，FPDemand と FPMult を変化させた場合のシミュレーション検証結果を表 9 に示す．

表 9　シミュレーション検証結果－No.3

シナリオ	FPDemand	FPMult	平均応答時間 [sec]
Controller 数増加	0.02	1	2.642
2 スレッド化	0.02	2	1.349
プロセッサ速度 2 倍	0.01	1	1.320
2 スレッド化&プロセッサ速度 2 倍	0.01	2	0.759
3 スレッド化&プロセッサ速度 2 倍	0.01	3	0.668

本セクションで示したシミュレーション検証結果に対する考察を以下に示す．

(1) 表 8 では，本手法と文献[11]の計算結果はよく一致する．計算精度は 95%信頼

区間の点で同程度と判断する．

(2) 図 12 では，Controller 数に応じて応答時間が増加し，fpm1 のリソース利用率が 1 に近づく．fpm1 がボトルネックと判断する．

(3) 表 9 では，2 スレッド化 and/or プロセッサ速度を 2 倍にすることで，応答時間が改善される．ボトルネックが解消されたと判断する．さらに 3 スレッド化しても改善効果は小さく，2 スレッドで十分処理できると考えられる．

6. 性能検証手法の評価

本章では，性能検証モデルとシミュレーションモデルの評価結果を示す．

6.1 性能検証モデルの評価

システム設計フェーズで作成するモデルをアプリケーション構造，アプリケーション振舞い，ソフトウェアリソース，ハードウェアリソースに分割して定義し，各々のモデルに対して MARTE Profile を利用して性能検証情報を直接付加する方法を規定した．また，システム設計モデルと性能検証モデルの要素間の対応を 1 対 1 に定義した．

これにより，システム設計結果と性能検証モデルの対応付けが容易になり，性能検証モデルを誤りなく作成するためのフィージビリティが確認できた．

今回はソフトウェア／ハードウェア両方を考慮した性能検証モデルを定義した．システムのネットワーク化を考慮した性能検証モデルの拡張は今後の課題である．まず，時分割，Ethernet 型，トークンリングといった実運用されている代表的なネットワークのモデル化に取り組む．

6.2 シミュレーションモデルの評価

性能検証モデルとシミュレーションモデルの要素間の対応を 1 対 1 に定義した．また，シミュレーションモデルを離散系イベントシミュレータ用のコードとして実装する方法を定義した．さらに，航空管制システムを題材に，文献[11]に記載の範囲で同等のシミュレーションモデルを作成し，計算精度が 95%信頼区間の点で同程度となった．また，パラメータを変更することでボトルネックを解消した．

これにより，シミュレーション検証結果を性能検証モデルおよびシステム設計モデルへと正確にフィードバックするためのフィージビリティが確認できた．

今回は性能検証モデルに対応するシミュレーションコードをすべて手作業で実装した．シミュレーションコードのライブラリ化による検証作業の工数削減は今後の課題である．まず，ハードウェア／ソフトウェアリソースやネットワークといった共通部分のライブラリ化に取り組む．

7. おわりに

本稿では，我々が取り組んでいる性能検証のアプローチにおいて，モデル駆動による性能検証手法を提案した．具体的には，性能検証モデルをアプリケーション構造／振舞い，ソフトウェア／ハードウェアリソース，検証シナリオに分割して作成する方法を規定した．また，性能検証モデルとシミュレーションモデルの対応を 1 対 1 に定義した．また，航空管制システムを題材にモデル作成とシミュレーション検証を行い，計算精度が既存手法と同程度であること，およびパラメータ変更によるボトルネックの解消を確認した．

これにより，今回我々が定義した性能検証手法に関して，複雑な実システムを対象に性能検証を実施すること，およびシミュレーション検証結果をシステム設計に正確にフィードバックすることのフィージビリティが確認できたと判断する．

この評価結果を踏まえ，今後我々が定義した性能検証手法を実用化するために，代表的なネットワークのモデル化，共通的なシミュレーションコードのライブラリ化を課題として抽出した．今後はこれらの課題に取り組む予定である．また，今回取り上げなかったモデルライブラリと性能検証ソフトウェアの開発にも取り組む予定である．

謝辞 本稿のシミュレーションモデルおよびコードの作成・検証については，カリフォルニア工科大学からのインターンシップ研修として実施した．研修学生の Ms. Carol Wang に謹んで感謝の意を表する．

参考文献

1) 森 雅夫，宮沢正清，生田誠三，森戸 晋，山田善靖，*オペレーションズリサーチII－意思決定モデル－*，朝倉書店 (1989)

2) Greg Franks, Peter Maly, Murray Woodside, Dorina C. Petriu, Alex Hubbard, *Layered Queueing Network Solver and Simulation User Manual Revision 6840*, Department of Systems and Computer Engineering, Carleton University (2005)

3) http://simpy.sourceforge.net/index.html

4) “Java Simulation Library (JSL),” http://www.uark.edu/~rossetti/research/research_interests/simulation/java_simulation_library_jsl/

5) “UCLA Parsec Programming Language,” http://pcl.cs.ucla.edu/projects/parsec/

6) “CSIM Performance Simulator,” http://www.csim.com/

7) *A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems, Beta 2*, OMG ptc/2008-06-09, Object Management Group (2008)

8) *OMG Unified Modeling Language™ (OMG UML), Superstructure, Version 2.2*, OMG formal/2009-02-02, Object Management Group (2009)

9) 森戸 晋，逆瀬川浩孝，*システムシミュレーション*，朝倉書店 (2000)

10) Nick Rozanski, Eoin Woods，*システムアーキテクチャ構築の原理 IT アーキテクトが持つべき 3 つの思考*，翔泳社 (2008)

11) Greg Franks, Tariq Al-Omari, Murray Woodside, Olivia Das, Salem Derisavi, “Enhanced Modeling and Solution of Layered Queueing Networks,” *IEEE Trans. on Software Eng.*, vol.35, no.2 (2009)