情報処理学会論文誌 Vol.59 No.2 775–784 (Feb. 2018)

CPUとメモリを共有資源として用いる実時間制御システムのモデリング手法

田中 輝明[1,2,a)] 武内 良典[2,b)] 今井 正治[2,c)]

受付日 2017年5月26日, 採録日 2017年11月7日

概要：近年の実時間制御システムにおいては，その基幹となる制御機能とともにネットワーク通信等の各種サービス機能を搭載した実時間制御用コントローラの実現が必要とされている．その実現検討では，ハードウェア資源の共有を積極的に行いつつ，確度の高い性能見積りが可能なアーキテクチャ設計手法が必要である．本論文では，実時間制御システムを制御するためのコントローラ・アーキテクチャの検討を効率的に実施可能な共有資源モデル記述を用いたシステム・モデリング手法を提案する．本手法は，既存のモデル化手法では扱いが困難であったアーキテクチャ上のCPU（計算資源）およびメモリ（記憶資源）に対する資源制約を，設計者が理解しやすい記述によりモデル化し，対象アーキテクチャのシミュレーションによる動作確認と性能評価を可能とするものである．本手法により，機能実現に必要となる並行処理動作，時間指定動作，および資源共有にともなう並行プロセスの逐次的動作を，システムアーキテクチャ検討の初期段階で簡単なモデル記述により表現し，対象システムの正確な動作の記述および性能評価が可能となる．

キーワード：システムレベル設計，実時間制御用コントローラ，組込みシステム，共有資源，計算資源，記憶資源，システムシミュレーション，SystemC

A Modeling Method of Real Time Control System with CPU and Memory as Shared Resource

Teruaki Tanaka[1,2,a)] Yoshinori Takeuchi[2,b)] Masaharu Imai[2,c)]

Received: May 26, 2017, Accepted: November 7, 2017

Abstract: This paper studies a modeling method for real time control system using shared resource models. Functions of modern real time control systems require not only conventional real time controls but also other services such as network communication functions. In order to implement these real time control systems, hardware sharing is indispensable technique and accurate estimation in early design stage becomes more important. This paper proposes a modeling method for real time control system using shared resource models. Proposed modeling enables compact design description to designer, and enables to estimate functional validation and performance evaluation. Experimental result shows that proposed model can model target real time control system compactly in early design stage, and estimate accurate performance evaluation.

Keywords: system-level design, realtime controller, embedded system, shared resource, computational resource, memory resource, system simulation, SystemC

1. はじめに

近年，M2M（Machine to Machine），IoT（Internet of Things）やサイバーフィジカルシステム（Cyber-Physical Systems，CPS）等で用いられる装置はネットワーク通信機能を持ち，相互のデータ交換やクラウド環境のような上位の大規模計算機との間での通信機能が求められている．

[1] 三菱電機株式会社先端技術総合研究所
Advanced Technology R&D Center, Mitsubishi Electric Co., Amagasaki, Hyogo 661–8661, Japan
[2] 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University, Suita, Osaka 565–0871, Japan
[a)] tanaka.teruaki@ak.mitsubishielectric.co.jp
[b)] takeuchi@ist.osaka-u.ac.jp
[c)] imai@ist.osaka-u.ac.jp

これらのシステムでは，高速・大容量のネットワーク通信を使用し，様々な計算リソースと有機的に接続することで，より知的な制御を行いうる産業システムおよび産業用機器の実現が期待されている．文献 [1] には製造産業に特化した CPS において，システムオブシステムズ（System of Systems，SoS），IoT，クラウド技術，Big Data や Industry 4.0 等により，新しい製造システムが出現可能であることが示されている．

産業システムの実現形態はフィールドレベル，制御レベル，運用管理レベルで階層化されており，物理世界からのデータセンシングや物理世界への制御を行う基幹装置は，制御レベルにあるプログラマブル・ロジック・コントローラ（Programmable Logic Controller，PLC）等の実時間制御を行う産業用コントローラである [2], [3]．一般的にプログラマブルな産業用コントローラは，ある時刻でのスナップ情報として全入力データを取り込み，それら入力データと現内部状態に基づきプログラム記述された制御処理を実行し，その結果を規定時刻でいっせいに外部出力することでリアルタイム性を有する制御機能を実現している．

近年の実時間制御用コントローラは上述の新しい分野で必要とされる様々な通信プロトコルへの対応が必要となっており，産業用途向け Ethernet や一般的な IP/UDP/TCP に基づくアプリケーション・プロトコルへの対応が増加している．その実現に必要となるプロトコルスタックは処理内容の複雑さや追加プロトコルへの継続的な対応から CPU による SW としての機能実現が求められるが，制御用コントローラとして要求される通信応答性能と制御性能とをともに確保する必要がある．また実時間制御用コントローラ実現においてコスト有効性（Cost-Effectiveness）を考慮する場合，CPU/メモリ/バス等の様々な HW リソースを最大限共用するアーキテクチャを検討する必要がある．特に近年は独自や専用の IC はできるだけ使用せず商用 LSI である MCU（Micro Control Unit）を核として使用するアーキテクチャ実現が重要となっている．近年の MCU は Ethernet 通信機能を搭載した製品も多く，また MCU のバリエーションも幅広く，その内部アーキテクチャや回路機能，たとえばバスのデータ幅や周波数，内蔵メモリ容量やアクセスポート数，ネットワークコントローラ機能，外部バス IF や関連 DMA 機能等は少しずつ異なっており，その選択肢は広がっている．

このような実時間制御用コントローラに対し，必要とされる制御機能と通信機能とを電子システム・プラットフォームで実現する場合，使用する HW コンポーネント間で様々な競合および干渉が発生する．たとえば，制御と通信の両処理がともに CPU を使用する場合，競合が生じる．またネットワーク通信で使用するパケットデータを保持するメモリ領域では，アプリケーション処理から送信データを書き込むのと同じタイミングでプロトコルスタックが受信データを書き込む場合，メモリへのアクセス競合が発生する．一般的にプロトコルスタックが使用する送信バッファと受信バッファは兼用されることが多く，通信内容により変動するバッファサイズと，非同期で発生する送受信要求タイミングにより，その競合・干渉の状況は変動する結果となる．このような競合や干渉の発生，またその状況の複雑さにより，実時間制御システムを実現するコントローラ開発においては，その設計検討時に制御性能と通信性能とをシミュレーションベースで確定させることが不可欠になってきている．

本問題の解決のため本論文では，共有資源モデル記述を用いたシステムモデリング手法を提案する．

本論文で提案する共有資源モデルは，CPU への処理割当てにより発生する計算資源共有による排他と，各処理が使用するデータの保持領域であるメモリ資源共有による排他に着目し，優先度付きでプリエンプティブかつ時間待ちを扱える排他制御ポリシを用いてシステムレベルにおける共有資源のモデル化とシミュレーション動作を実現したものである．また提案するモデルはシステムレベル記述言語である SystemC のモデルに対して，共有資源のモデル化要素を拡張し，関連する時間指定動作や資源獲得方法を追加し SystemC で実現したものである．

共有資源モデルは，処理の主体となるプロセス間で共有され，処理に既定した実行時間を，共有資源を獲得している間は消費し，獲得できない間は消費を停止する機能を有している．また各処理が必要とするデータへの R/W アクセスにより，データが保持されるメモリへの複数の処理からのアクセス競合が発生した場合に，処理のデータ入出力待ちを行う機能を有している．

この共有資源モデルにより，実時間制御システムを実現するコントローラ・アーキテクチャに必要となる処理実行時の競合・排他要素を簡潔かつ明確に表現可能となり，コントローラ設計の初期段階で適切なアーキテクチャ構成を検討・評価および確定することが可能となる．

本論文の構成は以下のとおりである．続く 2 章で関連する先行研究について紹介し，3 章で提案する共有資源モデルによるシステムモデリング手法の説明を行う．4 章では提案するシステムモデリング手法を用いたコントローラ・アーキテクチャモデルとその評価手法および結果について説明する．5 章では提案するシステムモデリング手法を考察する．最後に 6 章で本論文内容に関してまとめる．

2. 先行研究

文献 [4] ではネットワーク通信機能を持ち，マルチコア/メニーコア CPU を用いた PLC の実現検討がなされている．また同文献ではその実現において**スキャンサイクル時間**（Scan Cycle Time，SCT）と**通信のスループット**（high-throughput communication，HTC）が，PLC アーキテク

チャの主要な**重要評価指標**（Key Performance Indicator, KPI）であり，アーキテクチャ上でPLC特有であるスキャン処理の時間を評価可能であることが重要であることが示されている．ここでスキャンとは，データ入出力と制御処理および通信処理の基幹となる処理を合わせたコントローラ上での処理の単位であり，一定周期もしくは連続的にスキャンを繰り返し実行することにより実時間制御用コントローラのリアルタイム性が必要となる主機能を実現する．

また文献 [5] では主要なネットワークと想定されるEthernetに着目し，Ethernetベースのネットワークに対応したPLCのハードウェア設計について検討されている．また文献 [6] では，ネットワークアプリケーション例として様々なリモート環境からPLCが保有するデータ読み出しアクセスを行うためのWebサーバ機能を搭載したPLCの実現検討がなされている．しかしながら，これらの設計検討は，特定のハードウェアデバイスの使用を前提とし，その特定のハードウェアをもとに詳細な性能見積りを実施する手法であり，コントローラ設計の初期段階で必要となるハードウェア部品選定を含む設計プロセスで使用するには抽象度が低い手法となっている．製品価格や実行性能等の様々な要因に対して最適なハードウェア設計を行うためには，想定するハードウェアデバイスの組合せごとに抽象度の低い手法での見積りを実施する必要があり，設計工数が大きくなるという問題がある．

このような設計作業を効率化するため，モデルベース設計手法の導入も検討されている．たとえば文献 [7] では，IEEE 1666として規定されたシステムレベル記述言語であるSystemCを用いた産業オートメーションシステムのシステム設計が検討されている．また文献 [8] では，パケットベースのネットワークをモデル化するためのSystemCライブラリであるSCNSL（SystemC Network Simulation Library）を用いた分散制御システムの協調シミュレーション手法が検討されており，文献 [9] では，SystemCによるEthernet通信のモデル化を用いたCPSのシステム検討がなされている．同様に文献 [10] では組込みリアルタイムシステム向けのUML Profileを用いたモデル記述からMPSoC（MultiProcessor Systems-on-Chip）向けSystemCコードを実現する方式が検討されている．SystemCや関連するTLM（Transaction Level Modeling）の言語規定では標準的なセマフォやミューテックスは定義されているが機能は限定されており，SystemCに基づくこれらの文献においても，排他資源獲得の順序規定やプリエンプション等のポリシを実現する機能は検討されていない．また文献 [11] ではSystemCを用いたデジタル信号処理のシステム設計を行うための方法論および共有資源を扱う際に発生する競合状態やプリエンプションに関する検討がなされているが，時間待ちと上記排他制御ポリシとを連動させた実現は検討されていない，そのためシステムアーキテクチャ検討時に必要となる共有資源の排他を直接的に表現し，また動作実現する手段を有していないという問題があった．

3. 共有資源モデルを用いたシステムモデリング手法

コントローラ設計時のアーキテクチャ評価において，その主要なKPIは先述のとおりSCTとHTCであり，それらを定量的に評価するため「時間」を陽に扱うことが可能な評価環境が必要である．またMCUのアーキテクチャ差がコントローラにおける性能差やコスト差を与える結果となるため，その差異の内容を先述の共有資源とともに簡潔にモデル化し，上記KPIを簡潔に評価し，最適な選択肢を得ることがコントローラ・アーキテクチャ設計のポイントとなる．

コントローラ実現手段である電子システムでのシステムアーキテクチャを検討する場合，SystemCを用いたシステムレベル設計手法が幅広く使用されている．SystemCを用いることで，対象システムの

- 並行動作
- 明確な時間動作や時間待ち動作
- データ通信
- イベント
- アルゴリズム処理

を簡潔にモデル化し，シミュレーション実行および定量的な評価を行うことが可能である．しかしながらSystemCでは，共有資源利用に対して発生する排他による非実行時の残時間待ち動作を簡潔に表現かつ実現するモデル要素は提供されていない．そのため資源制約の多いシステムのアーキテクチャの検討では，多くのモデル要素を開発する必要があり，その使用は容易ではなかった．

これらの課題を解決すべく本論文では共有資源を持つシステムモデリング手法を提案する．以下に本手法の詳細内容と，提案するモデルライブラリの実現方式およびその動作について説明する．

3.1 システムレベル共有資源モデル

共有資源を持つシステムモデリングに必要な要素としては，

(1) 構造的なシステム要素（CPU，メモリ等）
(2) 処理時間や開始時間等の時間制御
(3) データ通信量等のデータサイズ
(4) 複数処理の並行実行動作
(5) 複数処理間での資源排他
(6) 資源排他での待ちに関する詳細動作

が考えられる．ここで (1) から (4) まではSystemC言語がサポートする標準モデルに含まれており，(5) および (6) が未サポートである．そのため，アーキテクチャ上の様々な共有資源と，資源の共有による制約を容易に表現可

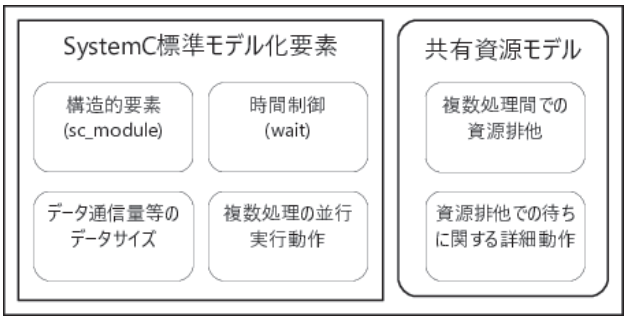

図 1 SystemC 標準モデルと本提案の拡張モデル
Fig. 1 SystemC standard model and proposed extended model.

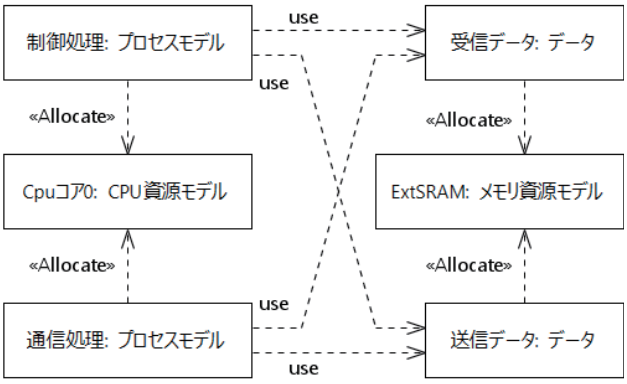

図 2 共有資源モデリング手法（インスタンスモデル）
Fig. 2 Shared resource modeling method (instance model).

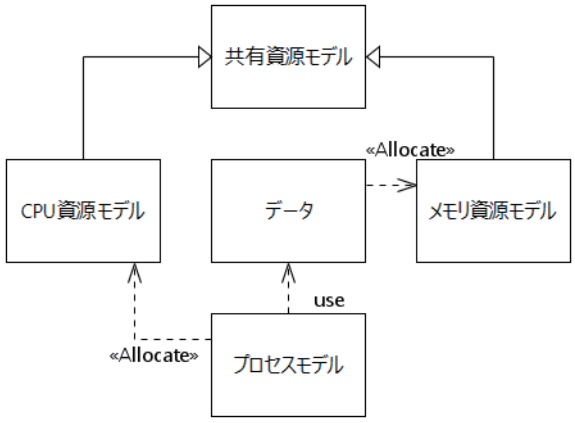

図 3 共有資源モデリング手法（概念モデル）
Fig. 3 Shared resource modeling method (conceptual model).

能かつ構成可能であることを目的とし，共有資源モデルを導入することで SystemC モデルを拡張した．その拡張部を**図 1** に示す．また提案する共有資源モデリング手法の概要を**図 2** および**図 3** に示す．なお，システム設計の効率化には，モデル化する処理への資源の割当てと割当て解除を簡潔に設定可能であることが様々なアーキテクチャ評価に対して必要であると考え，提案するシステムモデリング手法において簡潔に共有資源モデルを扱えることを設計方針とした．

図 2 で示す CPU やメモリのような共有資源の特性を簡単かつ共通的に実現するため，図 3 の「共有資源モデル」で示す基底モデルを導入している．共有資源を使用する処理は「プロセスモデル」として定義している．プロセスモデルは共有資源モデルを参照し，それらにアクセス可能である．プロセスモデルは，共有資源モデルを使用する処理を実行する際に共有資源の獲得処理を発行し，プロセスモデル側で既定した処理の実行時間に対応する時間待ちを実施し，処理の実行後には共有資源の解放処理を発行し，資源解放を通知する．

共有資源獲得の際，その共有資源が空いていればプロセスモデルは処理を継続するが，空いていない場合はプロセスモデルは共有資源の獲得待ちとなる．獲得待ちとなっている状態で，他のプロセスモデルにより共有資源が解放され使用可能となった場合には，待ちを行っているプロセスモデルは資源を獲得可能となるが，どのプロセスモデルが次に共有資源を獲得可能であるかを示す資源獲得特性を規定する必要がある．低コスト化を念頭においた現時点のシステム実現では，ハードウェア機能としてメモリ資源アクセスにおけるバスの FIFO アクセス，ソフトウェア機能として組込みリアルタイム SW における固定優先度タスクスケジューリングが主要な資源獲得特性であるため，先着順（First-Come-First-Serve，FCFS）と優先度順（PRIORITY）の 2 つの特性に対応した．FCFS は資源獲得を要求したプロセスモデルの要求順に沿って資源の割当てが行われる特性である．PRIORITY では，資源獲得に対する優先順位を各プロセスモデルは保有し，資源獲得を発行した際にすでに資源を使用中であるプロセスモデルよりも優先度が高かった場合には，資源使用中で最も低い優先度を持つプロセスモデルから共有資源を横取りする．横取りされたプロセスモデルは，その時点までの時間待ちを消費したとし，処理終了までの残時間を計算し，残時間が存在する場合には自動的に再度資源獲得待ちを発行する．この共有資源の動作に関する意味的な側面を規定するため，共有資源モデルは，同じく本モデリング手法中で定義した排他制御オブジェクトを，その基本特性として保有する構造としている．排他制御オブジェクトは，資源獲得/解放処理の実体となる `lock` 操作と `unlock` 操作を提供している．

3.2 共有資源モデルの実現

上述した各種モデルの実現においては，OSCI（Open SystemC Initiative）が提供している SystemC ライブラリをシミュレーションフレームワークとして使用し，共有資源モデルや同資源を使用するプロセスモデルは，同ライブラリで提供される C++言語の各種クラスを用いて構築している．そのクラス構造を**図 4** に示す．なお本図は UML（Unified Modeling Language）のクラス図を用いオブジェクト指向モデルとして，そのクラス構造を表現したもので

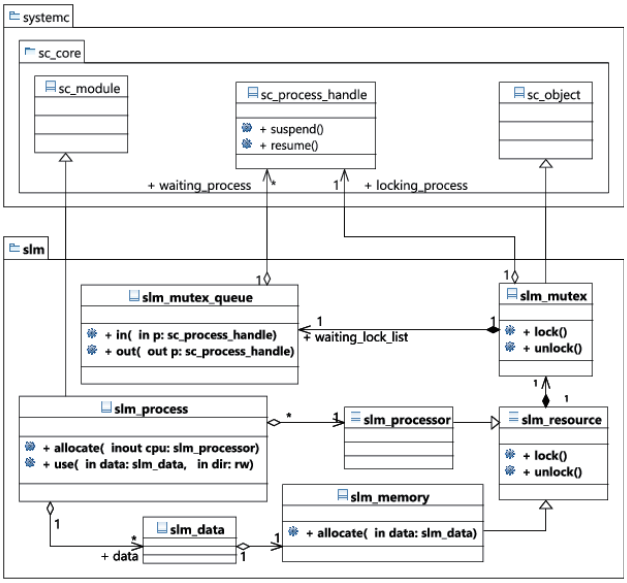

図 4 共有資源に関する基本的なクラス構造
Fig. 4 Foundamental class structure of shared resources.

ある．図 4 において `systemc` パッケージ表現が OSCI が提供する SystemC ライブラリを示しており，`slm` パッケージ表現が本論文で提案する共有資源モデルのクラスと共有資源を使用するプロセスのクラスを示している．

slm パッケージにおいて，共有資源モデルの基底クラスは slm_resource であり，CPU 資源モデルを表現するクラス slm_processor およびメモリ資源モデルを表現するクラス slm_memory は slm_resource を継承した構造としている．またプロセスモデルとなる slm_process クラスは SystemC における基幹動作モデルである sc_module クラスを継承した構造としている．slm_process クラスが CPU 資源を共有する場合には，slm_processor のインスタンスを slm_process の allocate() メソッドの引数として渡し，slm_process は渡された slm_processor を内部参照し排他制御動作を実現している．

slm_resource は，その排他処理としての特性を実現するため，排他制御クラスとして作成した slm_mutex をコンポジットする構造としている．共有資源クラス slm_resource は，その資源獲得を行うための `lock` 操作と，資源解放を行うための `unlock` 操作とを保有しているが，それらは slm_resource が保有する slm_mutex の該当操作の呼び出しにより規定されている．

また slm_process がメモリ共有資源上のデータアクセスを行うことによりメモリアクセスの競合が発生するモデル化は，メモリ資源を示す slm_memory と，slm_memory と slm_process とを媒介するための抽象的なデータモデルである slm_data により実現している．slm_process は，その処理が使用するデータを，そのデータ容量（バイト数）指定とともに slm_data をインスタンス化することで定義する．その slm_data のインスタンスを slm_process から use() メソッドで参照することで，slm_process から slm_data への R/W アクセスを関連付ける．なお Read or Write の指定は，use() メソッドの引数で指定する．また slm_data は，そのデータ置き場となる slm_memory のインスタンスに関連付けられる．slm_memory はそのインスタンスの生成時に，データアクセス速度（Byte per Second）とともに規定する．slm_data の slm_memory への割当ては，slm_memory の allocate() メソッドに slm_data を指定することにより実施する．これにより slm_process は slm_data を介して slm_memory を参照可能となり，slm_process 内部からデータアクセスを行った際，slm_memory へのアクセスが発生している状況では，資源の獲得待ち（メモリへのアクセス待ち）が発生する結果となる．

本共有資源モデルの実現では，slm_mutex における `lock`，`unlock` 操作，およびロック中のプロセスに対するロック強制解除にともなう時間待ちの制御が重要な処理となる．その詳細について 3.3 節で説明する．

3.3 lock，unlock 操作と強制待ち解除の実現

slm_mutex クラスは共有資源を保有している動作モデルを識別するとともに，共有資源の獲得待ちを行っている複数の動作モデル（プロセス）を管理するためのキュー slm_mutex_queue を保有している．キューの要素となるプロセス識別子には OSCI SystemC ライブラリの sc_core パッケージに含まれるプロセスハンドル sc_process_handle を使用している．slm_mutex での `lock` 操作および `unlock` 操作は以下に示す Algorithm 1 および Algorithm 2 として実現している．

`lock` 操作では，呼び出しプロセスのハンドルを取得し，共有資源が空いていればロックし，呼び出しプロセスをロック中のプロセスとして登録する．空いていない場合は資源をロックしているプロセスの優先度と呼び出しプロセスの優先度とを取得し，その値を比較する．比較結果として呼び出しプロセスの優先度が高い場合には，呼び出しプロセスをロック中のプロセスとして登録するとともに，呼び出し時点で資源をロックしているプロセスに対しロックを解除するための通知を行う．その通知は sc_process_handle の throw_it メソッドを用いて例外をスローすることで実現している．資源をロックしているプロセスは本例外により，後述する waste_processing_time() における時間待ちを行っている wait() を抜けることができる．また上記の優先度比較において，呼び出しプロセスの優先度が低い場合には，呼び出しプロセスを取得待ちリストに登録し，呼び出しプロセスの実行状態を Suspend 状態に移行させる．これにより処理の途中でプロセス動作は一時停止し待ち状態に入る．プロセスが再開された場合，継続動作として資源をロックし，自身を呼び出しプロセスとして登録する．その後，シミュレーションカーネル中のデルタ時間を進める操

作を発行する．デルタ時間を進める操作はSystemCカーネルでは SC_ZERO_TIME を引数として wait() を発行することにより実現可能であり，シミュレーション時間を進めずに他のプロセスへの切替えを行うことができる．

Algorithm 1 排他資源の獲得 [lock()]

```
proc_hndl ← sc_get_current_process_handle()
proc_kind ← proc_hndl.proc_kind()
if SC_NO_PROC_ ≠ proc_kind then
  if !locked then
    locked ← true
    locking_process ← proc_hndl
  else
    self_priority ← get_priority(proc_hndl)
    locked_priority ← get_priority(locking_process)
    if self_priority > locked_priority then
      cur_proc_handle ← locking_process
      locking_process ← proc_hndl
      cur_proc_handle.throw_it(std :: exception)
    else
      waiting_lock_list.in(proc_hndl)
      proc_hndl.suspend()
      locked ← true
      locking_process ← proc_hndl
    end if
  end if
  update delta time step
end if
```

unlock 操作では，同じく呼び出しているプロセスのハンドルの取得後，自身が資源を獲得していた場合，資源獲得を待つプロセスが存在するかどうかを確認し，獲得待ちプロセスがある場合には，そのプロセスを Resume 状態とする．その後，デルタ時間を進める操作である，先述の wait(SC_ZERO_TIME) を発行する．

Algorithm 2 排他資源の解放 [unlock()]

```
proc_hndl ← sc_get_current_process_handle()
proc_kind ← proc_hndl.proc_kind()
if (SC_NO_PROC_ ≠ proc_kind)&(locked == true)&(locking_process == proc_hndl) then
  rest_proc ← waiting_lock_list.out(nxt_locking_proc)
  if rest_proc then
    nxt_locking_proc.resume()
    update delta time step
  else
    locked ← false
    locking_process ← nullhndl
  end if
end if
```

なお lock，unlock の両操作ともに，シミュレーションカーネル上においてアトミックに実行される．

共有資源を使用する動作モデルの定義となる slm_process は，使用する共有資源の参照 resource を保有する sc_module のサブクラスである．動作モデルの共有資源の獲得が PRIORITY に基づき実施される場合，共有資源をロックする動作モデルをその優先度に応じて動的に切り替える必要がある．その実現を行うのが slm_process の waste_processing_time() 操作である．本操作は，プロセスの処理時間経過を時間待ちをすることで模擬するために使用する操作であり，途中でプロセスの切替えが発生しない場合には，指定した時間を経過後，操作を終了する．もしプロセス切替えが発生した場合には，処理時間の残経過時間を計算し，その時間待ちを再度シミュレーションカーネル上で発生させる操作となっている．その処理内容を Algorithm 3 に示す．

Algorithm 3 共有資源使用モジュール処理時間待ち [waste_processing_time(resource, proc_time_rest))]

```
repeat
  resource.lock()
  begin_time ← sc_time_stamp()
  try{
    wait(proc_time_rest)
    end_time ← sc_time_stamp()
  }
  catch(std :: exception& e){
    end_time ← sc_time_stamp()
  }
  proc_time_rest −= (end_time − begin_time)
until proc_time_rest > sc_time(0, SC_NS)
resource.unlock()
```

waste_processing_time() 操作では，まず共有資源の獲得要求を *resource.lock*() で発行する．共有資源を獲得できなかった場合，先述の lock 操作のとおり獲得まで時間待ちを行う．獲得後，その時点からの処理時間待ちを行うため sc_time_stamp() で開始時刻 begin_time を取得する．その後，指定時間待ちを行う wait() を待ち時間 proc_time_rest を引数として発行することで時間待ちに入る．そのまま処理時間待ちを完了した場合は，sc_time_stamp() で終了時刻 end_time を取得し，残処理時間 proc_time_rest を計算する．その値が 0 となっている場合は，そのままループを抜け，resource.unlock() で資源を解放して本処理を抜ける．もし wait() を発行している途中に他のプロセス上の lock() により資源獲得要求があり，優先度を比較した結果として他の動作モデルに資源が横取りされる場合，lock() から投げられる例外で wait() を抜け，catch に記述した例外処理に遷移する．例外処理では，それまでの経過時間を計算するため，現時刻を end_time として取得し，開始時刻 begin_time からの差分で残待ち時間を計算し，待ち時間が残っている場合はループ先頭に戻り，再度，共有資源の獲得待ちに入る．

4. システムのモデル化と評価

3 章で説明した内容に基づき CPU に対応する計算資源

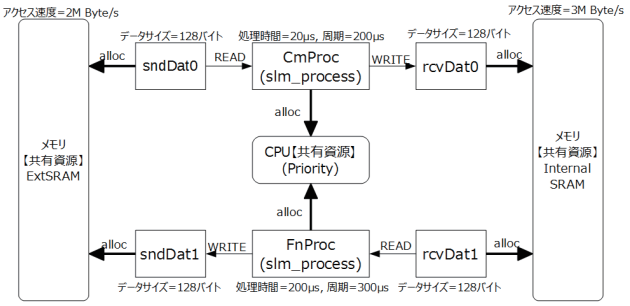

図 5 共有資源モデルを使用した対象システムモデル
Fig. 5 Target system model using shared resource model.

(slm_processor)，メモリ資源（slm_memory）およびプロセスモデル（slm_process）を用いて，対象とする実時間制御システムに必要となるモデル化と様々なアーキテクチャの性能見積りが可能であるかを評価した．以下，4.1 節で基本的なモデル化について説明し，4.2 節で複数のアーキテクチャ上での評価実施結果を説明する．

4.1 対象システムのモデル

対象システムのモデルを**図 5** に示す．本システムにおいてコントローラでは 2 つの主要な処理機能（プロセスモデル）が存在している．それらは

FnProc 基幹となる制御処理（スキャン処理）
CmProc 外部入出力データの更新を行う通信処理

である．両処理は共有資源である CPU を使用して動作することとし，その処理時間（データ転送時間を除く CPU 上での演算時間）は各々以下であるとする：

- 通信処理（CmProc）= 20 μs
- 制御処理（FnProc）= 200 μs

また通信処理は，外部から受信したデータを rcvDat0 とし，外部に送信するデータを sndDat0 として参照するとする．制御処理は，外部から受信したデータを rcvDat1 として参照し，外部に送信するデータを sndDat1 として書き込むとする．

このシステムでは

(1) CPU を 1 つとし，通信処理が制御処理よりも優先度が高く，CPU を横取りする状況では，どのような制御処理動作となるか，
(2) 使用する 4 つのデータをすべて同じメモリ上に割り当てる場合と異なるメモリ上に割り当てる場合とは，どのような性能差がでるか，
(3) CPU を 2 つとし，制御処理と通信処理を独立して実行させる場合，どのような動作となるか，

等の様々なアーキテクチャ検討を設計初期段階で実施する必要がある．なお上記 (1) に示す通信処理の実行優先度の方が制御処理の実行優先度よりも高いという構成は，通信応答時間の高速化要求を見越して，通信処理が発生するタイミングで制御処理に割込みを発生させても，制御処理の

```
1  int sc_main(int argc, char* argv[])
2  {
3    const unsigned int KILO = 1024;
4    const unsigned int MEGA = KILO * KILO;
5
6    /* 計算資源(CPU) */
7    slm_processor CpuCore("CpuCore",PRIORITY);//優先度
8
9    /* 処理の定義 */
10   slm_process<0,0,PERIODIC_RUN,200,SC_US>
11     CmProc("CmProc");
12   slm_process<0,0,PERIODIC_RUN,300,SC_US>
13     FnProc("FnProc");
14
15   /* 処理時間の設定 */
16   CmProc.proc_time = sc_time( 20, SC_US);
17   FnProc.proc_time = sc_time(200, SC_US);
18
19   /* 計算資源(CPUへの処理の割り当て) */
20   CmProc.allocate(CpuCore, 20); // 優先度 20
21   FnProc.allocate(CpuCore, 10); // 優先度 10
22
23   /* データの宣言 */
24   slm_data sndDat0("sndDat0", 128 /* Byte */);
25   slm_data sndDat1("sndDat1", 128 /* Byte */);
26   slm_data rcvDat0("rcvDat0", 128 /* Byte */);
27   slm_data rcvDat1("rcvDat1", 128 /* Byte */);
28
29   /* データを割り当てるメモリの宣言 */
30   slm_memory ExtSRAM("exSRAM",2*MEGA,FIFO,64);
31   slm_memory IntSRAM("IntRAM",3*MEGA,FIFO,64);
32
33   /* データのメモリ割当て */
34   ExtSRAM.allocate(sndDat0);
35   ExtSRAM.allocate(sndDat1);
36   IntSRAM.allocate(rcvDat0);
37   IntSRAM.allocate(rcvDat1);
38
39   /* 処理のデータ使用宣言 */
40   CmProc.use(sndDat0, READ);
41   CmProc.use(rcvDat0, WRITE);
42   FnProc.use(sndDat1, WRITE);
43   FnProc.use(rcvDat1, READ);
```

図 6 共有資源を使用するシステムモデルのプログラム記述
Fig. 6 Program description of system model using shared resources.

周期に問題が発生しないかを確認するための設定である．

作成したシミュレーションモデルのプログラム記述を**図 6** に記載する．本図は上記 (1) の評価で使用したプログラムコードである．ここでは計算資源として 1 つの CpuCore を slm_processor として定義している（7 行目）．また定義の際，CpuCore 資源の獲得ポリシを優先度順（PRIORITY）として規定している．

メモリ資源としては，外部 SRAM（ExtSRAM）および内部 SRAM（IntSRAM）を slm_memory のインスタンスとして生成している（30–31 行目）．宣言の際，アクセス速度（2 M バイト/秒），アクセス競合時の排他属性（FIFO），単位アクセスバイト数（64 バイト）を規定している．

また処理として CmProc と FnProc を slm_process のインスタンスとして生成している（10–13 行目）．両処理の宣

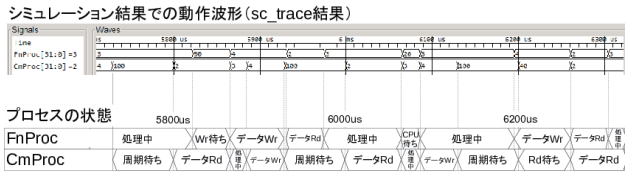

図 7 動作結果波形
Fig. 7 Simulation result as waveform.

言の際，周期実行であること（PERIODIC_RUN），またその周期がそれぞれ 200 μs，300 μs であることを指定している．両処理の処理時間は各々 20 μs，200 μs である（16–17 行目）．

共有資源への割当て記述として，処理の計算資源の割当ては 20–21 行目で実施している．なお CpuCore 資源の獲得ポリシが優先度順であるため，割当て時にその優先度を指定している．処理が使用するデータは，24–27 行目に sndDat0，sndDat1，rcvDat0，rcvDat1 として宣言している．各々のデータサイズは 128 バイトである．なお本データは通信処理と制御処理との間でやりとりする通信データを想定しており，データ名に付記した末尾の 0/1 はダブルバッファであることを示している．CmProc は sndDat0 への Read と rcvDa0 への Write を行い，FnProc は sndDat1 への Write と rcvDat1 への Read を行うことを規定している（40–43 行目）．データのメモリ資源の割当ては，sndDat0 および sndDat1 を ExtSRAM に，rcvDat0 および rcvDat1 を IntSRAM に指定している（34–37 行目）．

モデル化した対象システムをシミュレーション動作させた際の動作波形を**図 7** に示す（SystemC における vcd trace 出力）．動作波形の値は動作状態に付加した状態番号を示している．動作状態の意味を同じく図中に示す．CmProc は FnProc よりも優先度が高いため，FnProc の計算資源の獲得中に CmProc の計算資源獲得要求が発生した場合，計算資源を CmProc が取得し，FnProc は計算資源待ちとなっている（6,100 μs 付近）．また CmProc が処理を終了した時点で FnProc に処理が切り替わり，処理を再開していることが分かる．

また CmProc が周期動作を開始した時点で ExtSRAM に置かれた sndDat0 へのデータリードを開始するが，その後 FnProc が同じく ExtSRAM に置かれた sndDat1 へのデータライトを開始しようとして Write 待ちになっていることが分かる（5,800 μs 付近）．同様に FnProc が sndDat1 にデータライトを実行中に CmProc が sndDat0 へのデータリードを発行した際，CmProc の Read 待ちが発生していることも分かる（6,200 μs 付近）．

なお本提案のモデルライブラリは動作時の統計情報を出力可能となっており，CmProc および FnProc の処理開始時刻から処理完了時刻までの最大，最小，平均の処理時間値を取得することが可能である．その結果を**表 1** に示す．

表 1 CmProc および FnProc の処理時間
Table 1 Processing time of CmProc and FnProc.

処理	最大	最小	平均
CmProc	189.5 μs	129.3 μs	139.3 μs
FnProc	372.4 μs	329.3 μs	349.9 μs

表 2 6 種類の評価アーキテクチャ
Table 2 Evaluation architectures (Six types).

#	FnProc	CmProc	RcvDat	SndDat
1	CpuCore0	CpuCore0	ExtSRAM	ExtSRAM
2	CpuCore0	CpuCore0	IntSRAM	IntSRAM
3	CpuCore0	CpuCore0	IntSRAM	ExtSRAM
4	CpuCore0	CpuCore1	ExtSRAM	ExtSRAM
5	CpuCore0	CpuCore1	IntSRAM	IntSRAM
6	CpuCore0	CpuCore1	IntSRAM	ExtSRAM

表 1 において，CmProc は一定の処理時間で動作するよう優先度も高く指定しているが，メモリアクセスの競合により最大時間が大きく伸びるケースがあることが分かる（平均 139.3 μs → 最大 189.5 μs）．また平均値と最小値が近いことから多くが最小値に近い処理時間で動作していることも見てとれる．FnProc はメモリ資源や計算資源の競合により大きく時間変動が発生し，349.9 μs を平均値とし最大 372.4 μs から最小 329.3 μs となる ±20 μs 程度のばらつきを発生させ動作していることが分かる．

4.2 各種アーキテクチャ評価

4.1 節の基本的なモデル化をベースとして，共有資源への割当てが異なる 6 種類のアーキテクチャによる性能差の評価を実施した．各アーキテクチャでの割当ての切替えは，先述のプログラム記述における割当て先指定である図 6 における 20–21 行目：

```
CmProc.allocate(CpuCore0, 20);
FnProc.allocate(CpuCore1, 10);
```

や，34–37 行目：

```
ExtSRAM.allocate(sndDat0);
ExtSRAM.allocate(sndDat1);
InternSRAM.allocate(rcvDat0);
InternSRAM.allocate(rcvDat1);
```

の計 6 行分の指定を単に変更したのみである．

Cpu コアを最大 2 つとし FnProc および CmProc の割当て先を変更するとともに，送受信データ SndDat および RcvDat の割当て先メモリを ExtSRAM および IntSRAM に選択するようにした．その組合せを**表 2** に示す．

アーキテクチャ#1〜3 が CPU コアが 1 つの場合，#4〜6 が FnProc と CmProc を 2 つの CPU コアで独立して動作させる場合であり，その両者に対し送受信データの配置メモリをすべて外部 SRAM・すべて内部 SRAM および内部と外部の SRAM に割り当てた場合のアーキテクチャを

表 3 6 種類のアーキテクチャ評価
Table 3 Architecture evaluation results (Six types).

Arch.	1		2		3	
Fn/Cm	Fn	Cm	Fn	Cm	Fn	Cm
最大	498.8	262.1	379.1	185.2	379.1	141.5
最小	478.8	151.1	345.8	107.4	347.7	129.2
平均	488.9	195.2	346.4	133.1	348.2	133.2
Arch.	4		5		6	
Fn/Cm	Fn	Cm	Fn	Cm	Fn	Cm
最大	413.2	262.1	340.6	190.5	347.7	190.5
最小	413.2	151.1	287.4	107.4	347.7	129.2
平均	413.2	206.4	313.7	133.2	347.7	133.6

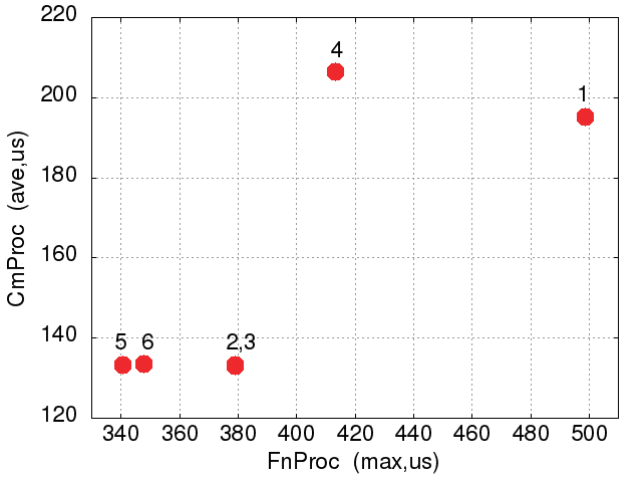

図 8 アーキテクチャ評価結果
Fig. 8 Architecture evaluation result.

評価した．

各アーキテクチャに対してシミュレーションを実施し，FnProc および CmProc の処理開始から完了までにかかった時間の最大・最小および平均を**表 3** に示す．なお数値の単位は 4.1 節と同じく μs である．

この結果からシステム設計のトレードオフを選択可能なように FnProc の最大処理時間と CmProc の平均処理時間をプロットしたものが**図 8** である．FnProc は制御処理（スキャン処理）であるため，その最大処理時間が先述の **SCT** に相当する．この値が小さければ制御周期を短くできるという効果がある．

また CmProc は通信処理であり，その平均処理時間が先述の **HTC** に相当する．この値が小さければより多くの要求に対応するという効果がある．

図上の点のところにある数値が上述のアーキテクチャの番号を示している．なおアーキテクチャ 2 と 3 は，結果がほぼ同じであるため，重なった位置に描画されている．

この結果から，FnProc/CmProc が独立した Cpu コアで動作可能であるアーキテクチャ 5 および 6 が SCT の効果が高いが，データアクセスでの競合により，アーキテクチャ 4 のように，単一の Cpu コア上での動作であるアーキテクチャ 2 および 3 よりも SCT が低くなることが分かる．また，アーキテクチャ 5，6 の CmProc 平均処理時間の差のとおり，すべて内部 SRAM に置かなくても一部データを外部 SRAM にオフロードしても HTC の性能差が変わらず，また複数 CPU であるアーキテクチャ 5，6 と単一 CPU であるアーキテクチャ 2，3 とでは，HTC に関してはさほど性能差が出ていないことが分かる．これにより

- 制御周期を 360 μs 以内に収める必要がある場合には CPU が 2 コア必要であるが，400 μs 程度以内でよければ CPU コアは 1 コアでよい，
- 通信に使用するデータはすべて内部 SRAM に置かなくても一部を外部 SRAM に配置しても性能差は大きくは出ない，

等の KPI に即したアーキテクチャトレードオフ下の良否判断を，先述のプログラム記述の 6 行程の修正を行いシミュレーションを行うだけで，簡単に行うことができた．

5. 提案手法に関する考察

このように提案するシステムモデリング手法では，共有資源というハードウェア資源に関するモデル，およびその資源獲得という動作のモデルを導入し，その割当てと使用とを宣言するモデル記述により，共有資源を使用する際のシステム動作を簡潔に記述できることが確認できた．

本手法で使用するモデル記述およびシミュレーション基盤は SystemC 言語および SystemC 環境を使用しており，同言語で提供される並行動作，時間指定動作のシミュレーション機能に今回提案の共有資源モデルとを組み合わせることにより，必要なシステムの挙動を，設計の初期段階で簡潔に模擬動作させることが可能であることが分かった．

特に，システムにおける必要となる処理機能と処理機能間の制御依存関係を最初に定義し，その後，使用する HW リソースの選定や割当てに対する数多くの選択肢を評価する場合には，本提案手法による簡潔な操作は設計工数を削減し，結果として，より適切なアーキテクチャ選定を実施可能となる．

また共有資源モデルとしては，1 つのインスタンスで複数の資源を宣言し，その獲得と解放を同じように扱うようにすることも可能である．これにより，マルチコア/メニーコア CPU における CPU コアの模擬を行うことも可能であり，そのシステムの振舞いをシミュレーション動作として確認することが可能である．

6. まとめ

本論文では，共有資源モデル記述を用いて実時間制御用コントローラを設計するためのシステムモデリング手法に関する提案を行った．提案したモデリング手法は，CPU やメモリのような処理の実現を行うための共有資源を簡潔に表現できるとともに，時間軸上でのシミュレーション動作

を実行・評価可能である方式である．

実際に，CPU とメモリを共有資源として想定し，SW として実現される複数の処理間で CPU 資源の使用要求を切り替えながら動作するシステム構成を簡潔に記述し，所望の動作を実現可能であることを確認できた．

また複数の共有資源を作成し，複数資源を使用するようなアーキテクチャ上の割当てを簡潔に指定し，同じくそのシミュレーション動作を簡潔に実行・評価可能であることを確認することができた．

また提案手法では電子システムレベルの設計言語である SystemC のシミュレーションライブラリを基盤として構築しており，通常の SystemC の機能動作とともに提案手法のモデルも動作可能である．そのため SystemC 設計ライブラリの流用による設計作業の効率化も行うことができる．

参考文献

[1] Wang, L.A., Törngren, M. and Onori, M.: Current Status and Advancement of Cyber-Physical Systems in Manufacturing, *Journal of Manufacturing Systems*, 37(Part 2), pp.517–527 (2015).

[2] Alphonsus, E.R. and Abdullah, M.O.: A review on the applications of programmable logic controllers (PLCs), *Renewable and Sustainable Energy Reviews*, Vol.60, pp.1185–1205 (2016).

[3] Vyatkin, V.: Software engineering in industrial automation: State-of-the-art review, *IEEE Trans. Industrial Informatics*, Vol.9, No.3, pp.1234–1249 (2013).

[4] Canedo, A., Ludwig, H., Faruque, A. and Abdullah, M.: High communication throughput and low scan cycle time with multi/many-core programmable logic controllers, *Embedded Systems Letters*, Vol.6, No.2, pp.21–24, IEEE (2014).

[5] Wang, B. and Wang, G.: A new intelligent programmable logic controller based on switched networks, *2013 9th International Conference on Natural Computation* (*ICNC*), pp.1712–1717, IEEE (2013).

[6] Mahato, B., Maity, T. and Antony, J.: Embedded Web PLC: A New Advances in Industrial Control and Automation, *2015 2nd International Conference on Advances in Computing and Communication Engineering* (*ICACCE*), pp.156–160, IEEE (2015).

[7] Fennibay, D., Yurdakul, A. and Sen, A.: A Heterogeneous Simulation and Modeling Framework for Automation Systems, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol.31, No.11, pp.1642–1655 (2012).

[8] Pedersen, N., Madsen, J. and Vejlgaard-Laursen, M.: Co-Simulation of Distributed Engine Control System and Network Model using FMI & SCNSL, *IFAC-PapersOnLine*, Vol.48, No.16, pp.261–266 (2015).

[9] Zhang, Z. and Koutsoukos, X.: Modeling Time-Triggered Ethernet in SystemC/TLM for Virtual Prototyping of Cyber-Physical Systems, *Embedded Systems: Design, Analysis and Verification*, pp.318–330, Springer Berlin Heidelberg (2013).

[10] Piel, É., Atitallah, R.B., Marquet, P., Meftali, S., Niar, S., Etien, A., Dekeyser, J.-L. and Boulet, P.: Gaspard2: From MARTE to SystemC simulation, *DATE'08 Workshop on Modeling and Analysis of Real-Time and Embedded Systems with the MARTE UML Profile* (2008).

[11] Haubelt, C., Falk, J., Keinert, J., Schlichter, T., Streubühr, M., Deyhle, A., Hadert, A. and Teich, J.: A SystemC-based design methodology for digital signal processing systems, *EURASIP J. Embedded Syst.*, p.15 (Jan. 2007).

田中 輝明 （正会員）

平成 4 年大阪大学工学部電子制御機械工学科卒業．平成 6 年同大学大学院修士課程修了．同年三菱電機株式会社入社．現在，同社先端技術総合研究所主席研究員．組込みリアルタイムシステム，コンパイラ，モデルベースシステムズエンジニアリング等の研究開発に従事．IEEE，INCOSE 各会員．

武内 良典 （正会員）

昭和 62 年東京工業大学工学部電気・電子工学科卒業．平成 4 年同大学大学院博士後期課程修了．博士（工学）．平成 8 年大阪大学大学院基礎工学研究科情報数理系専攻講師．現在，同大学大学院情報科学研究科准教授．ディジタル信号処理，VLSI 設計および VLSI CAD の研究に従事．IEEE，電子情報通信学会各会員．本会シニア会員．

今井 正治 （正会員）

昭和 49 年名古屋大学工学部電気工学科卒業．昭和 54 年同大学大学院博士後期課程修了（工学博士）．同年豊橋技術科学大学奉職．平成 6 年同教授．平成 8 年大阪大学大学院基礎工学研究科情報数理系専攻教授．平成 14 年大阪大学大学院情報科学研究科教授．平成 28 年同退職，名誉教授．これまでに組合せ最適化アルゴリズム，ハードウェア/ソフトウェア協調設計等の研究に従事．平成 3 年より日本電子機械工業会および IEEE/DASC において EDA 標準化作業に従事．IEEE，ACM 各会員，IFIP Silver Core Member，電子情報通信学会フェロー，本会フェロー．