情報処理学会論文誌　Vol. 51　No. 3　1056–1067 (Mar. 2010)

時間駆動処理とイベント駆動処理が共存する組み込み制御システムのための分散処理環境

石郷岡　祐[†1,*1]　伊丹悠一[†1]
兪　明連[†1]　横山孝典[†1]

本論文は時間駆動処理とイベント駆動処理が共存する組み込み制御システムのための分散処理環境について述べる．本論文の対象は，自動車制御システム等の，制御ブロック図を用いて制御設計を行う組み込み制御システムである．まず，すでに提案した時間駆動分散オブジェクトモデルに加え，イベント駆動分散処理モデルとして，純粋なイベントにより処理を起動するイベント駆動分散オブジェクトモデルと，データの到着イベントにより処理を起動するデータ駆動分散オブジェクトモデルを提案する．そして，時間駆動分散処理を実行する時間駆動セグメントとイベント駆動分散処理を実行する非時間駆動セグメントからなる実行サイクルを繰り返す，時分割スケジューリングに基づく分散処理環境を提案する．本分散処理環境は，時分割スケジューリングに基づき時間駆動タスクと非時間駆動タスクを管理するリアルタイム OS，前述の 3 つの分散処理モデルをサポートするミドルウェア，分散制御システムの開発を支援する開発環境からなる．

A Distributed Computing Environment for Embedded Control Systems with Time-triggered and Event-triggered Processing

Tasuku Ishigooka,[†1,*1] Yuichi Itami,[†1]
Myungryun Yoo[†1] and Takanori Yokoyama[†1]

The paper describes a distributed computing environment for embedded control systems with time-triggered and event-triggered distributed processing. The target domain of the paper is embedded control systems such as automotive control systems, in which the control logics are designed with block diagrams. We present a time-triggered distributed object model, an event-triggered distributed object model and a data-triggered distributed object model. We also present a distributed computing environment based on a time-division scheduling, which divides an execution cycle into a time-triggered processing segment for the time-triggered distributed processing and a non-time-triggered processing segment for the event-triggered and data-triggered distributed processing. The distributed computing environment consists of a real-time operating system that manages both time-triggered tasks and non-time-triggered tasks based the time division scheduling, middleware that supports the three kinds of distributed object models, and a development environment.

1. はじめに

自動車制御，FA（Factory Automation），ビル管理等多くの分野で，複数の組み込みコンピュータをネットワーク接続した分散型組み込み制御システムが使用されている．組み込み制御システムはリアルタイムシステムであり，入力イベントにただちに応答して処理をするイベント駆動アーキテクチャ（Event-Triggered Architecture）に基づく構成法と，所定の周期に基づいて処理をする時間駆動アーキテクチャ（Time-Triggered Architecture）に基づく構成法がある[1)]．ハードリアルタイムシステムには時間的な予測が容易な時間駆動アーキテクチャが有効といわれているが，高機能な組み込み制御システムは時間駆動処理のみで構成できるとは限らない．たとえば自動車のエンジン制御システムには，周期タスク以外に，エンジン回転に同期したイベントで起動されるタスクも存在する．そこで，時間駆動処理とイベント駆動処理が共存する組み込み制御システム向けの分散処理環境が要求されている．

リアルタイムシステムで用いられるタスクスケジューリング方式には，クロック駆動スケジューリング（Clock-Driven Scheduling）と優先度スケジューリング（Priority-Driven Scheduling）がある[2)]．これまでのリアルタイム OS の多くは優先度スケジューリングを採用しているが，時間駆動処理を実行するタスク（時間駆動タスク）はジッタの発生を嫌うため，静的に起動時刻を決定するクロック駆動スケジューリングが適している．なお本論文では，この分野の慣習に従い，スケジューリングを論じるときのみクロック駆動という言葉を使用し，それ以外では時間駆動という言葉を使用する．また，特に断らない限り，ジッタは時間駆動タスクの起動時刻のゆらぎ（周期のゆらぎ）を意味するものとする．ジッタの発生の少ないクロック駆動スケジューリングに基づく時間駆動 OS として，OSEK/VDX が仕

†1 東京都市大学（旧武蔵工業大学）
Tokyo City University
*1 現在，日立製作所
Presently with Hitachi Ltd.

様を規定した OSEKtime がある[3]．

時間駆動処理とイベント駆動処理を共存させる方法として OSEK/VDX は，優先度スケジューリングに基づく OSEK OS[4] を，OSEKtime 上のアイドルタスクとして時間駆動タスクの処理の空き時間に動作させる方式を提案している．しかし，2 つの OS を同時に動作させることはオーバヘッドやメモリ消費量の増大を招く．また Hattori らは，時間駆動タスクとイベント駆動タスクの両者を優先度スケジューリングで管理する，TT-OS と呼ぶ OS を提案している[5]．しかし時間駆動タスクを優先度スケジューリングで管理するとジッタやタスク起動時間の増大を招く．時間駆動タスクとイベント駆動タスクの両者をより効率的に扱える OS が求められている．

時間駆動とイベント駆動が共存する分散システムを対象としたスケジューリングの研究もなされている．Lonn らは分散型の自動車制御アプリケーションを対象に，タスクおよびネットワーク通信について，固定優先度スケジューリングと静的な周期的スケジューリングを組み合わせた方式の比較検討を行っている[7]．また Pop らは，時間駆動とイベント駆動が共存する分散型リアルタイムシステムのためのタイミング分析およびスケジューリング方式を提案している[8]．

時間駆動通信とイベント駆動通信の両者を扱えるネットワーク通信プロトコルとして，FlexRay が提案されている[6]．FlexRay は自動車制御分野を対象とした TDMA（Time Division Multiple Access）に基づくプロトコルで，その通信サイクルは時間駆動通信のための静的セグメント（Static Segment）とイベント駆動通信のための動的セグメント（Dynamic Segment）からなる．しかし分散システムを実現するには，ネットワーク（通信プロトコル）のみでなく，OS やミドルウェアを含む分散処理環境全体を時間駆動とイベント駆動の両者に対応させる必要がある．

しかし，時間駆動とイベント駆動が共存する組み込み制御システムのための分散処理環境の提案はあまりない．情報処理分野で広く使用されている分散処理環境 CORBA をベースに，優先度を導入することでリアルタイムシステム向けに拡張した Real-Time CORBA があり[9]，組み込みシステム向けに CAN ネットワークベース Real-Time CORBA も発表されているが[10]，優先度に基づく分散処理であるため遅延時間の予測は容易ではなく，ジッタの小さい時間駆動分散処理の実現に有効な CPU リソースやネットワークリソースの静的な割当てへの対応は考慮されていない．また，分散システムでイベントを扱うための CORBA イベントサービスも規定されているが[11]，動的なルーティングを採用しており通信処理時間が変動するため，遅延時間の予測が難しく，組み込み制御システムへの適用は困難である．ジッタの少ない時間駆動処理と遅延時間の予測が可能なイベント駆動処理の両者をサポートする分散処理環境が求められている．

そこで本論文の目的は，時間駆動とイベント駆動が共存する組み込み制御システム向けの分散処理環境を提案することである．そのため，時間駆動タスクとイベント駆動タスクの両者を効率的に扱える OS と，ジッタが小さい時間駆動処理と遅延時間の予測が可能なイベント駆動処理とを実現する分散処理ミドルウェアを開発する．また，本論文が主な対象とするアプリケーション分野は，いわゆる x-by-wire アプリケーションを含む自動車制御システムである．自動車制御分野では MATLAB/Simulink[12] 等の CAD/CAE ツールを用いたモデルベース制御開発が普及しており，それに対応した環境を実現する．

上記目的を達成するため，まず，時間駆動とイベント駆動の両者に対応した分散オブジェクトモデルを提案する．具体的には，すでに提案している時間駆動分散オブジェクトモデル[13] に加え，新たにイベント駆動分散オブジェクトモデルとデータ駆動分散オブジェクトモデルを提案する．それらはいずれもイベント駆動に基づくが，データをともなわない純粋なイベントによって起動されるものをイベント駆動，データの到着イベントによって起動されるものをデータ駆動と呼んで区別して扱う．

次に，上記 3 種類の分散オブジェクトモデルをサポートする分散処理環境を提案する．本環境はリアルタイム OS，ミドルウェア，開発環境からなり，時間駆動処理とイベント駆動処理を共存させて処理できる時分割スケジューリングを採用する．リアルタイム OS はクロック駆動スケジューリングと優先度スケジューリングの両者の機能を有し，前者により時間駆動タスクを，後者により非時間駆動タスクを管理する．ネットワークには FlexRay を使用し，FlexRay のハードウェアタイマを用いてスケジューラのタイマを制御することで，各ノード上の OS を同期実行可能とする．ミドルウェアは時間駆動分散処理の機能とともに，イベント駆動分散処理とデータ駆動分散理処理を実現するための分散イベントサービス機能を提供する．本ミドルウェアは静的ルーティングによりメッセージやイベントの転送を行うため，効率が良くジッタの少ない分散処理を実現できる．また，分散制御システムを効率良く開発するための開発環境も提供する．

本論文の構成は以下のとおりである．まず 2 章で時間駆動，イベント駆動，データ駆動の 3 つの分散オブジェクトモデルについて述べる．次に 3 章で分散処理環境の機能と構成について説明する．4 章では時間駆動，イベント駆動，データ駆動の 3 種の分散処理の動作について説明する．そして 5 章で開発した分散処理環境の評価を行い，6 章で本論文のまとめと今後の課題を述べる．

2. 分散処理モデル

2.1 時間駆動分散オブジェクトモデル

MATLAB/Simulink 等を使用したモデルベース制御開発では，制御ブロック図を用いて制御ロジックを記述し，その設計・検証を行う．**図 1** は制御ブロック図の例の一部である．この制御ブロック図はエンジントルク（EngineTorque）やスロットル開度（ThrottleOpening）を計算するブロックからなっており，それらを制御周期（サンプリング周期）に従い周期的に実行することで制御処理を行う．

時間駆動分散オブジェクトに基づく制御ソフトウェアの設計は，すでに提案した時間駆動オブジェクト指向開発法[14),15)] に従う．制御ソフトウェアを構成するオブジェクトの基底クラスであるデータ値オブジェクト（ValueObject）を**図 2** に示す．データ値オブジェクトは属性 value とメソッド update，get，set を持ち，update は value の値を算出するメソッド，get，set は value の値の読み出しおよび書き込みのためのアクセスメソッドである．

図 1 の制御ブロック図に対応するオブジェクトの組合せを**図 3** に示す．オブジェクトは制御ブロック図中のデータに対応しており，エンジントルクの値を算出し記憶するオブジェクト EngineTorque，スロットル開度の値を算出し記憶するオブジェクト ThrottleOpening 等がある．これらは ValueObject のサブクラスである．各オブジェクトの update メソッドを制御周期で呼び出すことにより，制御処理を実現できる．計算に必要な他のオブジェクトの値は，それらの get メソッドを呼び出して取得する．

図 4 に時間駆動分散オブジェクトの例を示す．自動車制御で用いられる組み込みコンピュータは ECU（Electronic Control Unit）と呼ばれる．ECU1 と ECU2 にオブジェクト EngineTorque と ThrottleOpening が分散配置されている．また，EngineTorque のレプリカが ECU2 に配置され，複製処理（replication）により ECU1 上の EngineTorque の値を周期的にレプリカにコピーする．ThrottleOpening は同じ ECU2 上の EngineTorque のレプリカの get メソッドを呼び出して，エンジントルクの値を得ることができる．ECU1 上の時間駆動タスク Task1 が EngineTorque の update と送信側の複製処理を，ECU2 上の時間駆動タスク Task2 が受信側の複製処理と ThrottleOpening の update を実行している．

図 5 は時間駆動分散処理のタイムチャートの例である．時間駆動タスクは静的にスケジュー

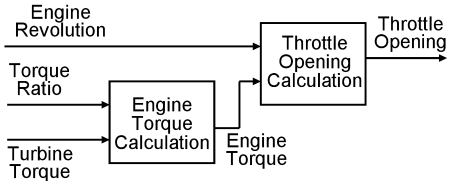

図 1　制御ブロック図の例
Fig. 1　Example block diagram.

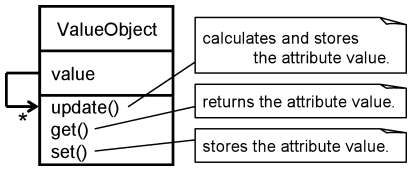

図 2　制御ソフトウェアの基底クラス
Fig. 2　Base class for control software.

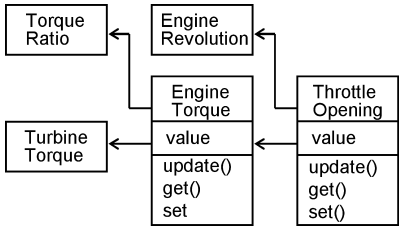

図 3　制御ソフトウェアのクラス図の例
Fig. 3　Example class diagram of control software.

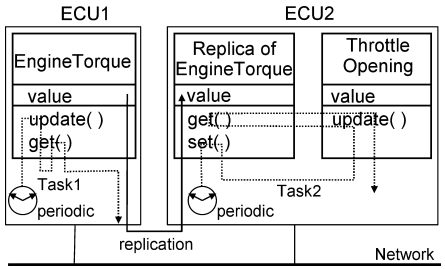

図 4　時間駆動分散オブジェクト
Fig. 4　Time-triggered distributed objects.

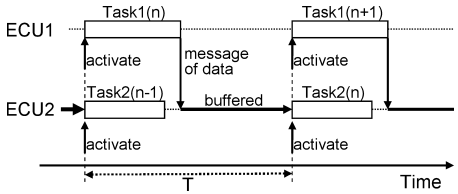

図 5　時間駆動分散処理のタイムチャート
Fig. 5　Time chart of time-triggered distributed processing.

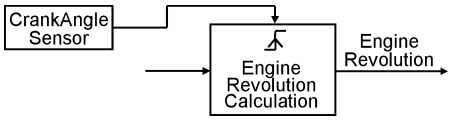

図 6　トリガサブシステムを含む制御ブロック図の例
Fig. 6　Example block diagram with triggered subsystem.

ルされ，ノード間で同期して起動される．図 5 において，Task1(n) は Task1 の n 回目（n サイクル目）のジョブを，Task2(n) は Task2 の n 回目のジョブを示しており，Task2(n) は Task1(n) によって算出された値を参照している．

2.2　イベント駆動分散オブジェクトモデル

制御ロジック設計時に起動タイミングを明示的に指定した処理を実現するにはイベント駆動を用いる．イベント駆動では，データをともなわない純粋なイベントにより処理を起動する．たとえば Simulink では，トリガサブシステム（triggered subsystem）を用いてイベント駆動を記述可能で，その例を**図 6** に示す．この例では，エンジン回転数を算出するブロック（EngineRevolutionCalculation）の処理が，クランク角センサ（CrankAngleSensor）の出力信号の立ち上がりエッジで起動される．

図 7 はイベント駆動分散処理のタイムチャートの例である．ECU1 上のタスク Task1 がイベントメッセージを送信すると，それを受信した ECU2 上のタスク Task2 が起動される．

図 8 は図 6 に対応するイベント駆動分散オブジェクトの例である．オブジェクト CrankAngleSensor が ECU1 に，オブジェクト EngineRevolution が ECU2 に配置されている．クランク角センサの入力割込みにより起動される ECU1 上の Task1 により CrankAngleSensor の update が実行され，イベントメッセージが送信される．そして，イベントメッセージの受信割込みにより起動される ECU2 上の Task2 により EngineRevolution の update が実行される．

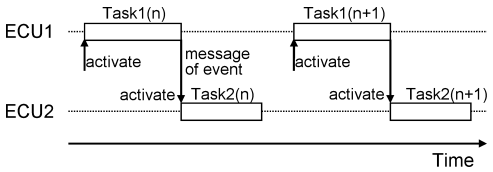

図 7　イベント駆動分散処理のタイムチャート
Fig. 7　Time chart of event-triggered distributed processing.

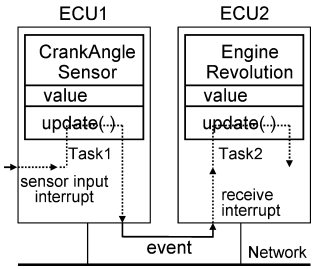

図 8　イベント駆動分散オブジェクト
Fig. 8　Event-triggered distributed objects.

2.3　データ駆動分散オブジェクトモデル

制御ブロック図はブロック間のデータフローを表現している．たとえば図 1 において，スロットル開度を計算するブロックはエンジントルクを計算するブロックが出力するデータを受け取って計算を実行する．そこで，それらの計算を異なる ECU で実行する場合，エンジントルクのデータの受信によりスロットル開度の計算処理を起動することが考えられる．

このように分散型組み込み制御システムでは，データ受信により処理を起動することもできる．本論文ではこれをデータ駆動と呼ぶ．**図 9** はデータ駆動分散処理のタイムチャートの例である．ECU2 上のタスク Task2 は，ECU1 上のタスク Task1 からのデータを待って処理を行う．たとえば，Task2(n) は Task1(n) から送られてくるデータを待って休止状態となり，データ受信後に処理を再開する．Task2(n+2) の場合はデータ待ちに入る前にデータを受信しているので，タスクは休止状態とならず処理を続行している．

図 10 はデータ駆動分散オブジェクトの例である．時間駆動の場合と同様に ECU2 上に EngineTorque のレプリカを配置する．ThrottleOpening の update を実行している Task2

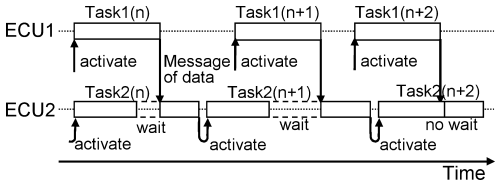

図 9　データ駆動分散処理のタイムチャート
Fig. 9　Time chart of data-triggered distributed processing.

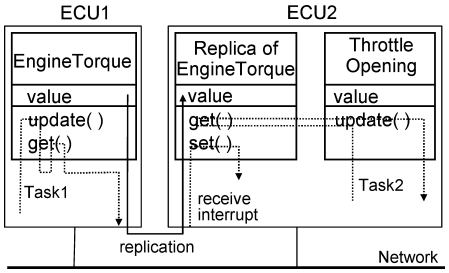

図 10　データ駆動分散オブジェクト
Fig. 10　Data-triggered distributed objects.

が EngineTorque のレプリカの get を呼び出すと，複製処理によるレプリカの value の更新が完了するまで，処理を休止する．すでに更新されている場合は休止の必要はなく，そのまま処理を継続する．

3. 分散処理環境の機能と構成

3.1 時分割スケジューリング

一般にディジタル制御では，制御周期のゆらぎが制御性能に影響を与えるため，制御周期を正確に守ること，すなわちジッタを削減することが重要である．そこで我々は，時間駆動処理を非時間駆動処理（イベント駆動処理，データ駆動処理）より優先する方針を採用し，それに基づいて両者を共存させ実行する時分割スケジューリングを提案する．本方式は，時間駆動処理を実行するセグメントと非時間駆動処理を実行するセグメントからなる実行サイクルを繰り返し実行する．これを CPU のスケジューリングとネットワーク通信のスケジューリングの両者に適用し，前者を 3.3 節で述べるリアルタイム OS により，後者

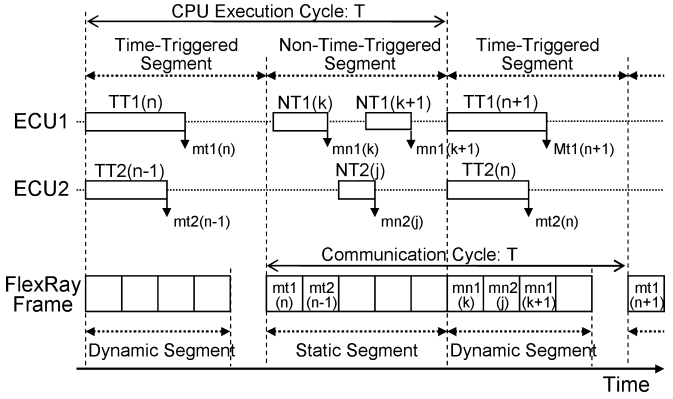

図 11　時分割スケジューリング
Fig. 11　Time division scheduling.

を FlexRay の機能を用いて実現する．FlexRay は TDMA に基づくプロトコルで，1 つの通信サイクルは，あらかじめ静的に割り当てた時間枠で周期的な通信を行う静的セグメントと，実行時の要求により通信を行う動的セグメントからなる．静的セグメントで時間駆動通信を，動的セグメントでイベント駆動通信を実行する．

時分割スケジューリングのタイムチャートの例を**図 11** に示す．CPU の実行サイクル（CPU Execution Cycle）とネットワークの通信サイクル（Communication Cycle）の周期はいずれも T である．CPU 実行サイクルは時間駆動セグメント（Time-Triggered Segment）と非時間駆動セグメント（Non-Time-Triggered Segment）からなり，時間駆動タスクを前者で，非時間駆動タスクを後者で実行する．この例では時間駆動セグメントと非時間駆動セグメントを同一長としている．また通信サイクルは，FlexRay の仕様に従い，静的セグメントと動的セグメントからなり，時間駆動通信を前者で，非時間駆動通信を後者で実行する．CPU 実行サイクルと通信サイクルは同期実行されるが，この例では CPU 実行サイクルの時間駆動セグメントの開始時刻と通信サイクルの動的セグメントの実行開始時刻が一致するように位相を決めている．図中の TT1，TT2 は時間駆動タスク，NT1，NT2 は非時間駆動タスクである．また，mt1，mt2 は時間駆動メッセージ，mn1，mn2 は非時間駆動メッセージである．TT1(n) で算出した値を mt1(n) で転送し，TT2(n) で使用している．

CPU の処理における時間駆動セグメントと非時間駆動セグメントの割当ては，各 CPU における時間駆動タスクの CPU 使用率と，非時間駆動タスクの CPU 使用率を考慮して決

定する．ネットワーク通信における静的セグメントと動的セグメントの割当ては，時間駆動メッセージのネットワーク使用率と非時間駆動メッセージのネットワーク使用率を考慮して決定する．周期の異なる複数の時間駆動タスクが存在するシステムに時分割スケジューリングを適用する場合には，それらの周期の最大公約数を実行サイクルの周期とする．自動車制御では，最小周期を T_{min} として，$2^n T_{min}$（$n = 0, 1, 2, \cdots$）の周期を用いることが多い．この場合は T_{min} を実行サイクルの周期とする．ただし，非時間駆動タスクは時間駆動セグメントにプリエンプトされ，その分応答時間が長くなる可能性がある．非時間駆動タスクの応答時間を短縮するため，実行サイクルの周期をより短く（T_{min} の整数分の 1）することも可能だが，タスク切替えの回数が増えるためオーバヘッドが増大するので，両者のトレードオフを考慮して決定する必要がある．

リアルタイム性の保障は，従来の組み込み制御システムの多くと同様に，設計時に静的に行う．システム設計者には，対象とするハードウェア上での OS のタスク管理にかかる時間（最悪実行時間），ミドルウェアの処理時間（最悪実行時間），ネットワーク通信時間等，分散処理環境のオーバヘッド時間（最悪実行時間）が定数として与えられる．これらとアプリケーションプログラムの各タスクの処理時間（最悪実行時間）を用いて，設計を行う．時間駆動処理の設計は，時分割により使用できる CPU リソースとネットワークリソースが制限されることを考慮する必要があるが，基本的に従来の時間駆動のみの場合と同様である．1 ノードあたり 1 周期分の遅延が発生するので，その分の遅延を考慮して設計する．一方，非時間駆動処理の場合は時間駆動セグメントにより発生する遅延を考慮して設計する必要がある．非時間駆動タスクは優先度ベースのスケジューリングに従うので，時間駆動セグメントを優先度が最も高い周期タスクと見なして設計する．

時分割スケジューリングは時間駆動処理とイベント駆動処理を明確に分け，それぞれ異なるセグメントで独立に処理する．したがって，CPU 実行サイクルにおける時間駆動セグメントと非時間駆動セグメントの割当て，および通信サイクルにおける静的セグメントと動的セグメントを決定した後は，使用できる CPU リソースおよびネットワークリソースは固定的に与えられるので，これまでに提案されたスケジューリング手法をそれぞれのセグメントに適用することで，システム全体のスケジュール可能性の検討が可能である．

3.2　ソフトウェア構成

開発した分散処理環境のソフトウェア構成を**図 12** に示す．分散処理環境はリアルタイム OS（RTOS），FlexRay ネットワークドライバ，ミドルウェアからなり，その上でアプリケーションプログラムが動作する．また開発環境として IDL（Interface Definition Language）コンパイラとコンフィギュレータを提供する．

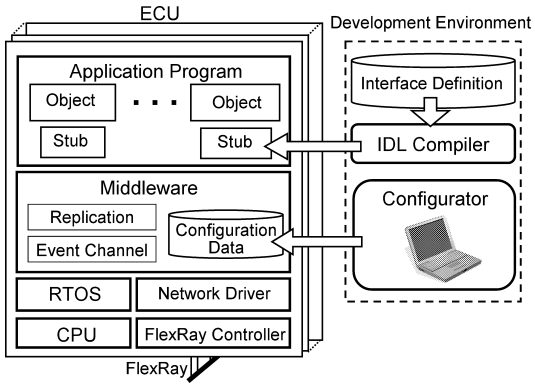

図 12　分散処理環境
Fig. 12　Distributed computing environment.

アプリケーションプログラムはオブジェクトとスタブからなる．オブジェクトは 2 章で述べたように制御ブロック図から設計するが，コード生成ツールにより自動生成することも可能である[15]．スタブはオブジェクトとミドルウェアを接続するために用いられ，IDL により記述したインタフェース定義から IDL コンパイラにより自動生成される．IDL の文法は CORBA IDL に準拠する．

OS は時分割スケジューリングに基いてタスクを管理する．ミドルウェアは複製処理機構とイベントチャネル機構を有し，コンフィギュレーションデータを参照して処理を実行する．一般に組み込み制御システムはノード構成やタスク構成を静的に決定できるので，システムを構成する ECU，タスク，イベント，メッセージ，通信周期等の情報をコンフィギュレーションデータとして静的に与えることができる．コンフィギュレーションデータを参照することで，静的ルーティングに基づくメッセージやイベントの転送が可能になる．コンフィギュレーションデータは，開発者が GUI（Graphical User Interface）により入力した情報からコンフィギュレータが生成する．

3.3　リアルタイム OS

本分散処理環境のリアルタイム OS は，TOPPERS プロジェクト[16] により開発された OSEK OS 仕様準拠の TOPPERS/OSEK カーネルに，クロック駆動スケジューリング機能を付加することで実現した．クロック駆動スケジューリングにより，時分割スケジュー

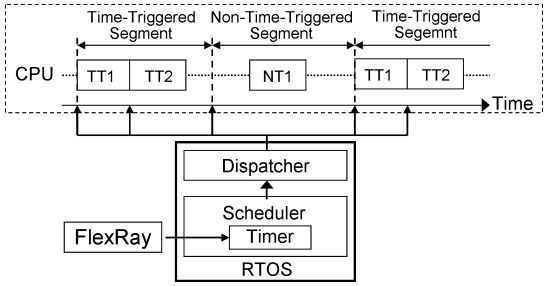

図 13　リアルタイム OS
Fig. 13　Real-time operating system.

リングにおけるセグメントの切替えと時間駆動タスクの管理を行う．非時間駆動タスクは TOPPERS/OSEK カーネル本来の優先度スケジューリングにより管理する．時間駆動タスクの仕様は，OSEKtime の時間駆動タスクの仕様に従う．したがって，時間駆動タスクには待ち状態は存在せず，OSEK OS で規定されているタスク間の同期や通信のためのシステムサービスは使用できない．非時間駆動タスクは OSEK OS のタスクの仕様に従う．

本 OS の詳細を**図 13** を用いて説明する．スケジューラのタイマを FlexRay のハードウェアタイマに同期させることで，分散システム全体で同期したスケジューリングを実現する．時分割スケジューリング用のスケジューラは OSEK OS のカテゴリ 1 の ISR（Interrupt Service Routine）（OS に管理されない割込み処理）により実行する．本スケジューラは時間駆動タスクの起動時刻情報とタイマを比較し，それらが一致した場合にタスク起動フラグをセットしてディスパッチャを呼び出す．ディスパッチャから時間駆動タスクを起動するため，カテゴリ 1 の ISR からタスクを起動するためのシステムサービスを実装した．本方式は優先度の評価やキュー操作が不要なため，オーバヘッドの少ないスケジューリングが可能である．

図 14 は本 OS が扱うタスクと割込み処理の処理レベル（優先度）を表している．上位ほど優先度が高い．前述のように時分割スケジューリング用スケジューラは最も優先度の高いカテゴリ 1 の ISR で実行する．前記方針に基づき，時間駆動タスクの優先度をカテゴリ 2 の ISR（OS の管理下にある割込み処理），OSEK OS スケジューラ（非時間駆動タスク用スケジューラ），OSEK OS タスク（非時間駆動タスク）より高くすることで，ジッタを最小限に抑える．なお，ジッタを許容できる優先度の低い周期タスクは，非時間駆動タスクとして OSEK OS のアラーム機能を用いて実装することも可能である．

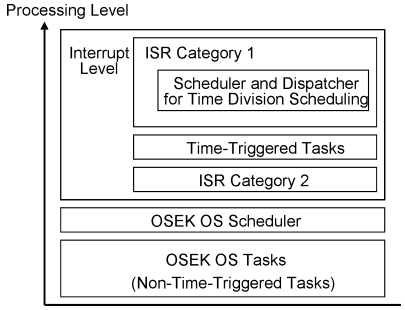

図 14　処理の優先度
Fig. 14　Processing levels.

表 1　分散イベントサービス
Table 1　Distributed event services.

Service	API	Argument
set event	mw_SetEvent	event ID
wait event	mw_WaitEvent	event mask
clear event	mw_ClearEvent	event mask
activate task by event	mw_ActEvent	event ID

3.4　ミドルウェア

すでに開発した時間駆動分散オブジェクト環境[13] のミドルウェアをベースに，分散イベントサービスのためのイベントチャネル機構を追加実装するとともに，FlexRay に対応した．OSEK OS では，データをともなわない純粋なイベントをタスク間で伝達するためのイベントサービスを提供している．本分散イベントサービスは，OSEK OS のイベントサービス仕様を分散処理環境向けに拡張したもので，その API を**表 1** に示す．OSEK OS 仕様で規定されている，イベントをセットする SetEvent，イベントを待つ WaitEvent，セットされたイベントをクリアする ClearEvent の 3 つを分散処理環境向けに拡張した mw_SetEvent，mw_WaitEvent，mw_ClearEvent の API をミドルウェアで提供する．さらに，イベントによってタスクを起動するための新しい API として mw_ActEvent を追加提供する．

イベントチャネル機構は，同一 ECU 上のタスクへのイベントは OSEK OS のイベントサービスに任せ，他の ECU 上のタスクへのイベントを処理する．コンフィギュレーションデータを参照しての静的なルーティングにより，ジッタの少ない処理が可能である．純粋な

イベントのみでなく，イベントとデータを関連づけて扱うこともできる．分散イベントサービスを用いたイベント駆動分散処理およびデータ駆動分散処理の実現方法は 4.2 節，4.3 節で述べる．

4. 分散処理環境の動作

4.1 時間駆動分散処理

図 15 は図 4 に示した時間駆動分散処理の流れを示したものである．複製処理により ECU2 上の EngineTorque のレプリカの状態を ECU1 上のオリジナルの EngineTorque と同じ状態に保つ．複製処理にはオリジナルスタブ，レプリカスタブの 2 種類のスタブを用いる．以下，時間駆動分散処理における複製処理の動作を説明する．

ECU1 上のミドルウェアの複製処理機構（Replication）は，オリジナルスタブ（pack()）を呼び出して EngineTorque の value の値をメッセージにパックし，送信メッセージバッファに書き込む．そしてネットワークドライバ（send()）を呼び出して送信する．ECU2 上のミドルウェアの複製処理機構はネットワークドライバ（receive()）を呼び出して受信メッセージを取得し，受信メッセージバッファに記憶する．そしてレプリカスタブ（unpack()）を呼び出してデータを EngineTorque のレプリカの value に書き込む．以上の処理を EngineTorque の update と同一周期で実行する．

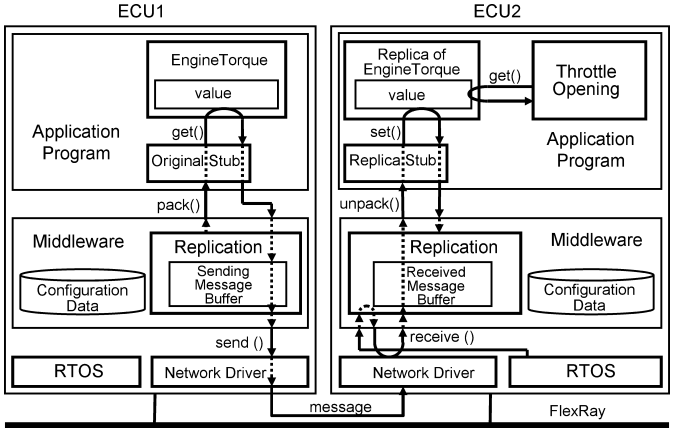

図 15　時間駆動分散処理の流れ
Fig. 15　Time-triggered distributed processing flow.

4.2 イベント駆動分散処理

図 16 は図 8 に示したイベント駆動分散処理の流れを示したものである．ECU1 上の CrankAngleSensor から送られてきたイベントにより ECU2 上のタスクを起動し，EngineRevolution のメソッドを実行する．イベントチャネル機構は API，イベントルータ（Event Router），イベント通信処理（Event Communication）からなる．以下，イベント駆動分散処理の動作を説明する

ECU1 上の CrankAngleSensor が API mw_ActEvent() を呼び出すと，イベントルータがコンフィギュレーションデータを参照して送信先を決定し，イベント通信処理を呼び出す．イベント通信処理はネットワークドライバ（send()）を呼び出してイベントメッセージを送信する．

ECU2 では，メッセージ受信割込みにより起動されたイベント通信処理が，ネットワークドライバ（receive()）を呼び出して得たイベントメッセージからイベント ID を取得して，イベントルータを呼び出す．イベントルータは OS の ActivateTask() を呼び出し，イベント ID に対応するタスクを起動する．そして起動されたタスクが EngineRevolution のメソッドを実行する．

4.3 データ駆動分散処理

図 17 は図 10 に示したデータ駆動分散処理の流れを示したものである．ECU2 上の ThrottleOpening が EngineTorque のレプリカの属性値を読み出そうとすると，それがす

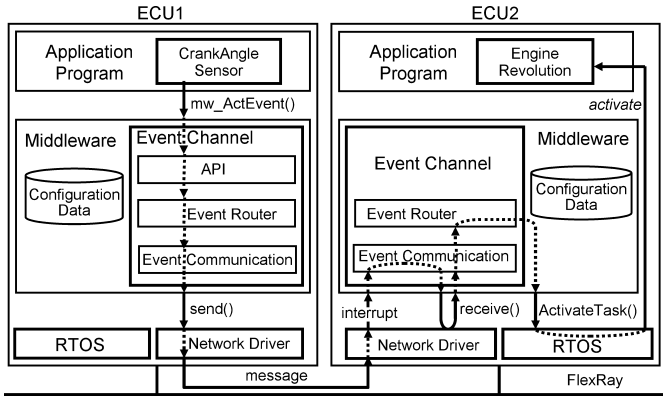

図 16　イベント駆動分散処理の流れ
Fig. 16　Event-triggered distributed processing flow.

でに更新済みであればそのまま読み出すが，そうでない場合は更新完了を待って読み出す．データ駆動分散処理には set スタブと get スタブの 2 種類のスタブを用いるが，それらには分散イベントサービスの API の呼び出しが埋め込まれている．以下，データ駆動分散処理の動作を説明する

ECU1 上の EngineTorque は set スタブを呼び出して自身の属性値を更新する．スタブは属性値の書き込み後に API mw_SetEvent() を呼び出す．そしてイベントルータがコンフィギュレーションデータを参照して送信先を決定し，イベント通信処理を呼び出す．イベント通信処理はコンフィギュレーションデータを参照して送信すべき属性値データを決定し，読み出す．そしてネットワークドライバ（send()）を呼び出して ECU2 にメッセージを送信する．

ECU2 では，メッセージ受信割込みにより起動されたイベント通信処理がネットワークドライバ（receive()）を呼び出して受信メッセージを取得し，コンフィギュレーションデータを参照してデータを書き込むべきレプリカ（この場合は EngineTorque のレプリカ）を決定して書き込むとともに，イベント ID を指定してイベントルータを呼び出す．イベントルータはコンフィギュレーションデータを参照してイベント ID に対応するタスクとイベントマスクを決定し，OS の SetEvent() を呼び出してレプリカの属性値の更新完了イベントをセットする．

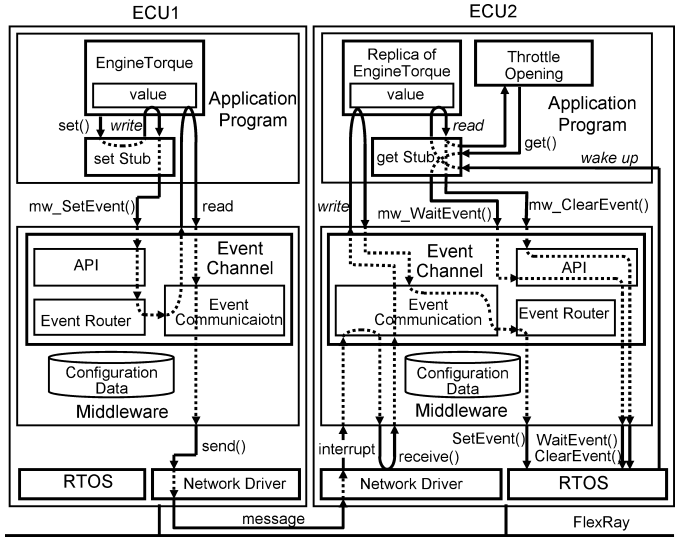

図 17　データ駆動分散処理の流れ
Fig. 17　Data-triggered distributed processing flow.

一方，ECU2 上の ThrottleOpening が EngineTorque のレプリカの属性を読み出すために get スタブを呼び出すと，スタブは API mw_WaitEvent() を呼び出して（OS の WaitEvent() が呼び出される）レプリカの属性値の更新完了イベントを待つ．イベントがすでにセットされている場合はそのまま get スタブの処理を継続する．イベントがまだセットされていない場合は待ちに入り，イベントセット後に起こされ，get スタブの処理を継続する．そして更新された属性値を読み出し，API mw_ClearEvent() を呼び出して（OS の ClearEvent() が呼ばれる）イベントをクリアしてから，属性値を ThrottleOpening に返す．

5. 実装と評価

我々は CPU V850（50 MHz）と FPGA 実装された FlexRay コントローラを搭載した FlexRay 評価ボード上に，提案した分散処理環境を実装した．

本論文で開発した OS の狙いは，時間駆動タスクと非時間駆動タスクの両者を扱えるようにするとともに，クロック駆動スケジューリングにより時間駆動タスクを効率良く実装することである．**表 2** に，本 OS の時間駆動タスク（Time-Triggered Task）の起動時間と，比較のために測定した OSEK OS のアラーム機能を用いた周期タスク（OSEK OS Periodic Task）の起動時間を示す．タスク起動時にプリエンプションが発生する場合（with preemption）と発生しない場合（without preemption）の値を示した．時分割スケジューリングの実行サイクルの周期は自動車制御の周期タスクでよく使用される 10 msec，時間駆動セグメントと非時間駆動セグメントは同一長（5 msec）として測定した．ただし，タスクの起動時間は周期やセグメントの割当てによって変化することはない．

OSEK OS の周期タスクの起動時間と比較し，時間駆動タスクの起動時間を 2/3 以下に削

表 2　タスクの起動時間の比較
Table 2　Comparison of task activation time.

Task	Condition	Time
Time-Triggered Task	without preemption	15.7
	with preemption	17.5
OSEK OS Periodic Task	without preemption	24.3
	with preemption	28.4

[μsec]

表 3　ミドルウェアの通信処理時間
Table 3　Execution time of middleware.

Processing		Ave.	Max.	Min.	Dif.
Time-Triggered	Transmit	43.20	43.28	43.20	0.08
	Receive	47.04	47.04	47.04	0.00
Event-Triggered	Transmit	33.12	33.20	33.12	0.08
	Receive	6.00	6.08	6.00	0.08
Data-Triggered	Transmit	35.04	35.12	35.04	0.08
	Receive	3.20	3.20	3.20	0.00

[μsec], resolution: 0.02 μsec

減できた．またプリエンプションの有無による差も，時間駆動タスクの場合は 1.8 μsec と，OSEK OS のタスクの場合の半分以下となった．これらはクロック駆動スケジューリングの効果であると考える．なお，非時間駆動タスクの起動時間は OSEK OS のタスクの起動時間と同じである．

ミドルウェアの処理時間の変動は，時間駆動におけるジッタ発生の原因になるとともに，イベント駆動やデータ駆動における遅延時間の予測を困難にする要因となるため，できるだけ小さくしたい．**表 3** に，時間駆動分散処理，イベント駆動分散処理，データ駆動分散処理におけるミドルウェアの通信処理時間を示す．表には送信側（Transmit）と受信側（Receive）の処理時間の平均値（Ave.），最大値（Max.），最小値（Min.），最大値と最小値の差（Dif.）を示した．表の値は OS の処理時間を除いたミドルウェアとネットワークドライバのみの処理時間である．

前述のようにミドルウェアの処理時間の差（変動）はジッタの発生原因や遅延時間の予測を困難にするため，アプリケーションの処理時間の変動より十分小さくする必要がある．最大値と最小値の差は最大 0.08 μsec（CPU の 4 マシンサイクルに相当）と，十分小さな値が得られた．前述のプリエンプションの有無によるタスクの起動時間の差は時間駆動タスクの場合 1.8 μsec，非時間駆動タスクの場合 4.1 μsec で，これらと比較しても非常に小さな値である．分散イベントサービスを静的ルーティングにより実装した効果と考える．

本論文で開発した時分割スケジューリングに基づく分散処理環境の適用範囲について述べる．時分割スケジューリングでは，実行サイクル周期の大きさがイベント駆動処理の遅延の原因となるため，その影響について考察する．一般に自動車制御における制御ロジックを実行するタスクの周期は 10 msec 程度を基準とし，その整数倍とすることが多い．入出力を扱う周期タスクの場合は最小 1 msec 程度である．したがって，時間駆動タスクとしては最小 1 msec 程度の実行サイクル周期に対応できれば，多くの自動車制御アプリケーションに適用可能と考える．また，現在は CAN ネットワークを用いた周期的通信が広く用いられているが，その最小周期は 10 msec 程度である．したがって，イベント駆動通信を導入した場合も，1 ノードあたり 1 msec 程度の遅延は許容されることが多いと考える．したがって，自動車制御の多くのアプリケーションでは，時分割スケジューリングの実行サイクル（通信サイクル）の最小周期は 1 msec 程度と想定できる．今回開発した OS における時間駆動タスクの起動時間は 20 μsec 以下，非時間駆動タスクの起動時間は 30 μsec 以下，ミドルウェアの通信処理時間は数十 μsec 程度以下であり，1 msec 程度の実行サイクル周期に対応するには，実用上問題ない値であると考える．1 msec はエンジン回転数が毎分 6,000 回転のときの 1/10 回転に対応する．したがって，1 ノードあたり 1 msec あるいはエンジン回転数換算で 1/10 回転程度の遅延が許容されるアプリケーションであれば，本分散処理環境は適用可能と考える．

時分割スケジューリングの実行サイクル周期や，CPU 実行サイクルにおける時間駆動セグメントと非時間駆動セグメントの割当て，通信サイクルにおける静的セグメントと動的セグメントの割当てについて考慮すべき事項については，3.1 節で述べた．しかし，これらは指針にとどまっており，定量的な設計法については今後の課題である．

6. おわりに

時間駆動とイベント駆動が共存する組み込み制御システムを実現するため，時間駆動，イベント駆動，データ駆動の 3 種類の分散処理モデルを提案するとともに，時分割スケジューリングに基づく分散処理環境を開発した．本分散処理環境は，時間駆動タスク向けのクロック駆動スケジューリング機能と非時間駆動タスク向けの優先度スケジューリング機能を有するリアルタイム OS，静的ルーティングによりジッタの少ない複製処理や分散イベントサービスを提供するミドルウェア，分散制御システムの開発を支援する開発環境からなる．

本分散処理環境で採用した時分割スケジューリングは時間駆動とイベント駆動を別々のセグメントで独立に処理するため，これまでに提案されたスケジューリング手法をそれぞれのセグメントに適用することで，システム全体のスケジュール可能性の検討が可能である．しかし，最適なセグメント分割やリソース割当てについては課題が残されており，今後検討していく予定である．

謝辞　本研究で開発したリアルタイム OS のベースに使用した TOPPERS/OSEK カーネルの開発者に感謝する．本研究は科研費（20500037）の助成を受けたものである．

参 考 文 献

1) Kopetz, H.: Should Responsive Systems be Event-Triggered or Time-Triggered?, *IEICE Trans. Information & Systems*, Vol.E76-D, No.11, pp.1325–1332 (1993).
2) Liu, J.W.S.: *Real-Time Systems*, Prentice Hall, New Jersey (2000).
3) OSEK/VDX: Time-Triggered Operating System, Version 1.0 (2001).
4) OSEK/VDX: Operating System, Version 2.2.3 (2005).
5) Hattori, H., Ohnisi S., Morikawa A., Nakamura K. and Takada, H.: Open Source FlexRay Communication: Time Triggered OS and FlexRay Communication Middleware, *Proc. IP-Based SoC Design Conference* (*IP/SOC 2006*), pp.227–233 (2006).
6) Makowitz, R. and Temple, C.: FlexRay – A Communication Network for Automotive Control Systems, *Proc. 2006 IEEE International Workshop on Factory Communication Systems*, pp.207–212 (2006).
7) Lonn, H. and Axelsson, J.: A Comparison of Fixed-Priority and Static Cyclic Scheduling for Distributed Automotive Control Applications, *Proc. 11th Euromicro Conference on Real-Time Systems*, pp.142–149 (1999).
8) Pop, T., Eles, P. and Peng, Z.: Schedulability Analysis for Distributed Heterogeneous Time/Event Triggered Real-Time, *Proc. 15th Euromicro Conference on Real-Time Systems*, pp.257–266 (2003).
9) OMG Technical Document formal/02-08-02, Real-Time CORBA Specification, Version 1.1 (2002).
10) Lankes, S., Jabs, A. and Bemmerl, T.: Integration of a CAN-based Connection-oriented Communication Model into Real-Time CORBA, *Proc. International Parallel and Distributed Processing Symposium* (*IPDPS'03*), p.123a (2003).
11) OMG Technical Document formal/04-10-02, Event Service Specification, Version 1.2 (2004).
12) Moscinski, J.: *Advanced Control with MATLAB and Simulink*, Ellis Horwood, Ltd. (1995).
13) 石郷岡祐，横山孝典：組み込み制御システム向け時間駆動分散オブジェクト環境，情報処理学会論文誌，Vol.48, No.9, pp.2936–2945 (2007).
14) 横山孝典，納谷秀光，成沢文雄，倉垣　智，永浦　渉，今井崇明，鈴木昭二：組込み制御システムのための時間駆動オブジェクト指向ソフトウェア開発法，電子情報通信学会論文誌，Vol.J84-D-I, No.4, pp.338–349 (2001).
15) 吉村健太郎，宮崎泰三，横山孝典：オブジェクト指向組み込み制御システムのモデルベース開発法，情報処理学会論文誌，Vol.46, No.6, pp.1436–1446 (2005).
16) TOPPERS Project. http://www.toppers.jp/

(平成 21 年 6 月 22 日受付)
(平成 21 年 12 月 17 日採録)

石郷岡　祐（正会員）

2006 年武蔵工業大学工学部電子情報工学科卒業．2008 年同大学大学院工学研究科電気工学専攻修士課程修了．同年（株）日立製作所入社．組み込みシステム，分散システムの研究に従事．IEEE 会員．

伊丹　悠一（学生会員）

2008 年武蔵工業大学工学部コンピュータ・メディア工学科卒業．同年同大学大学院工学研究科電気工学専攻博士前期課程入学．現在，同課程在学中．組み込みシステム向けリアルタイム OS の研究に従事．

兪　　明連（正会員）

1994 年安東国立大学校工学部コンピュータ工学科卒業．1996 年浦項工科大学情報通信専攻修士課程修了．同年安東情報大学講師．2002 年嶺南大学コンピュータ工学専攻博士課程修了．2006 年早稲田大学大学院情報生産システム研究科博士後期課程修了．2007 年武蔵工業大学．現在，東京都市大学准教授．スケジューリング理論の研究に従事．博士（工学）．電子情報通信学会，IEEE 各会員．

横山　孝典（正会員）

1959 年生．1981 年東北大学工学部通信工学科卒業．1983 年同大学大学院工学研究科電気及通信工学専攻修士課程修了．同年（株）日立製作所入社．1987 年から 1990 年まで（財）新世代コンピュータ技術開発機構出向．2004 年武蔵工業大学．現在，東京都市大学教授．組み込みシステム，分散システム，ソフトウェア工学等の研究に従事．博士（情報科学）．電子情報通信学会，IEEE，ACM 各会員．