情報処理学会論文誌　データベース　Vol.5　No.3　36–50 (Sep. 2012)

車載組込みシステム向けデータストリーム管理の静的スケジューリング方式

勝沼 聡[1,†1,a)]　本田 晋也[1]　佐藤 健哉[1,2]　高田 広章[1]

受付日 2012年3月20日, 採録日 2012年7月7日

概要：車載組込みシステムでは車載データの種類・量が増加し管理が複雑化しているため，我々はデータストリーム管理システム（DSMS）による車載データの統合管理を検討してきた．しかし従来の汎用システム向け DSMS のスケジューリング方式では各オペレータをストリームキューで接続し，スケジューラによりオペレータ間の接続切替えを動的（実行時）に行うため，メモリ使用量や処理レイテンシが大きい．本稿で提案する，車載組込みシステム向け DSMS の静的スケジューリング方式では，オペレータの動的接続が不要な箇所を静的（開発時）に検出し，検出箇所ではストリームキューを除去し，オペレータがスケジューラを介さず次に実行するオペレータを呼び出す．提案方式を組込み環境で評価した結果，従来方式に対して優位性を示した．

キーワード：DSMS，ストリーム，組込みシステム，車載システム，スケジューリング

The Static Scheduling Method in Data Stream Management for Automotive Embedded Systems

Satoshi Katsunuma[1,†1,a)]　Shinya Honda[1]　Kenya Sato[1,2]　Hiroaki Takada[1]

Received: March 20, 2012, Accepted: July 7, 2012

Abstract: The types and amount of vehicle data has increased and then the automotive embedded systems have become difficult to manage these data. Therefore, we considered the data stream management system (DSMS) to manage vehicle data. However, in existing DSMS for general-purpose systems, these memory usages and processing latencies are high, since the stream queues connect the operators and the scheduler dynamically (in execution) switches connections between the operators. We propose the static scheduling technique in DSMS for automotive embedded systems. This technique detects the unnecessary dynamic connections between the operators statically (in development). In these detection points, the stream queues are eliminated and the operators call other operators without the scheduler. We evaluated this technique in the embedded systems and the experimental result showed that this technique is efficient.

Keywords: DSMS, stream, embedded system, automotive system, scheduling

1. はじめに

近年，プリクラッシュセーフティ技術など，車両の状態や周辺状況を判断し，ドライバへの警告や自動制御により運転の支援を行うシステム（以下，運転支援システム）が登場している [1]．たとえば車両に搭載された複数のセンサからの情報に基づき，操舵回避の支援を行い，衝突が避けられない状況では介入ブレーキを作動させることで衝突衝撃を緩和し被害を軽減するシステムがある [2], [3]．また車

[1] 名古屋大学大学院情報科学研究科
Graduate School of Information Science, Nagoya University, Nagoya, Aichi 464–8601, Japan
[2] 同志社大学モビリティ研究センター
Mobility Research Center, Doshisha University, Kyotanabe, Kyoto 610–0321, Japan
[†1] 現在，日立製作所中央研究所
Presently with Central Research Lab., Hitachi Ltd.
[a)] katsunuma@nces.is.nagoya-u.ac.jp

両追従，レーン逸脱警告，自動駐車などもある [4]．一方，カーナビゲーションにおいても交差点右折時の対向車両などの周辺車両や，事故，渋滞などの道路状況に対する警告，案内が求められる．このような運転支援システムやカーナビゲーションにおいて，周辺の物体を検知するためにミリ波レーダやレーザレーダ，カメラをはじめ車輪速センサ，加速度センサ，位置検出センサなど多様なセンサを複数搭載し，車々間通信や路車間通信の利用も加わって，相互に情報交換を行う必要がある．

運転支援システムやカーナビゲーションなどの車載組込みシステムでは，センサやアプリケーションの増加にともない，システムの開発に要するコストが増加している．我々はこの問題を解決するため，センサに依存するデータ処理（車載データ処理）をアプリケーションから切り離し，プラットホームで統合的に管理する車載データ統合プラットフォーム [5] を検討している．プラットフォームではデータストリーム管理システム（Data Stream Management System，DSMS）を用いることで，車々間通信データなどのストリームデータの処理の保守性，再利用性を向上させ，開発コストを削減する．

STREAM [6]，Aurora [7] など従来の DSMS は，株自動取引，電子マネーなどを扱う汎用システムを想定し，動的（実行時）に DSMS に対して処理したいクエリを追加する．そして新たに追加したクエリを，登録済みのクエリとオペレータなどを共有することで，DSMS で実行するクエリ（以下，実行クエリ）を変更する．またデータの到着頻度の増加など実行状況の変化にともなって，実行するオペレータの優先度を変更する．このため DSMS では各オペレータをストリームキューで接続し，スケジューラにより動的にオペレータ間の接続切替えを行うスケジューリング方式（以下，動的スケジューリング方式）が必要となり，メモリ使用量や処理レイテンシが大きい．

そこで我々のプロジェクトでは車載組込みシステム向けの DSMS（embedded DSMS，eDSMS）を検討している．eDSMS では，車載組込みシステムにおいてアプリケーションを固定し車載データ処理の内容を静的（開発時）に確定することに着目し，実行するすべてのクエリを静的に登録する．そして複数クエリのオペレータの共有化などの処理最適化を行い，実行クエリやオペレータの優先度を静的に決定する．

本稿では，eDSMS のストリームキューのメモリ使用量および，スケジューラの処理レイテンシの削減を目的とし，実行クエリやオペレータの優先度に従ってオペレータの動的な接続切替えを削減する方式（以下，静的スケジューリング方式）を提案する．静的スケジューリング方式では，優先度に従って複数のオペレータをグループとして抽出し，同一グループ内のオペレータ間のストリームキューを除去し，同期してオペレータを実行する．またグループ内において，動的にオペレータが切り替わらない箇所を実行クエリから検出し，検出箇所では，スケジューラを介さずオペレータが直接，次のオペレータを実行する．

本稿の構成は以下のとおりである．まず 2 章で DSMS のスケジューリング方式の課題を述べる．3 章で eDSMS のコンセプトを述べる．そして 4 章において静的スケジューリング方式を説明し，5 章でその評価について述べる．6 章で関連研究を述べる．

2. 汎用システム向け DSMS のスケジューリング方式の課題

本章ではまず汎用システム向け DSMS のスケジューリング方式について述べる．そして DSMS の実行に関連する車載組込みシステムの要件を述べ，その要件をふまえ，スケジューリング方式を車載組込みシステムに適用するうえでの課題を述べる．

2.1 汎用システム向け DSMS とそのスケジューリング方式

以下では**図 1**（II）に示す汎用システム向けの DSMS の動的なクエリの登録および，処理最適化について述べ，その処理最適化を実現するための動的スケジューリング方式を述べる．

2.1.1 動的なクエリ登録，処理最適化

DSMS のクエリでは図 1（1）に示すように，データをストリームとして扱い，そのストリームに対する処理をオペレータ（**表 1**）により定義する．定義したクエリは，図 1（2）に示すように，実行環境にインストール済みのランタイム（図 1（2））に登録する．そしてランタイムではクエリの変更や，実行状況に変化に応じて動的な処理最適化（以下，動的処理最適化（図 1（4））を行う．動的処理最適化は様々な方法が提案されているが [6], [7], [8], [9], [10]，その一例として 1. クエリ追加時のオペレータ，ストリームの共有化，2. 実行状況に応じたオペレータの優先度変更，について以下で説明する．

- 最適化 1. クエリ追加時のオペレータ，ストリームの共有化

汎用システム向けの DSMS では，登録されたクエリに対して，すでに登録済みのクエリとオペレータやストリームを共有化し実行クエリを変更する．たとえば図 1（II）ではクエリ 1 が登録されているランタイムに，クエリ 2 を新たに登録する場合を示す．この場合，クエリ 2 はオペレータ Filter A，Aggregate B および，ストリーム a，b，d がクエリ 1 と共通であるため，これらの共通のオペレータ，ストリームをクエリ 1 と共有化し，クエリ 1 に含まれないオペレータ Map E，ストリーム g のみ新たに実行クエリに追加する．

- 最適化 2. 実行状況に応じたオペレータの優先度変更

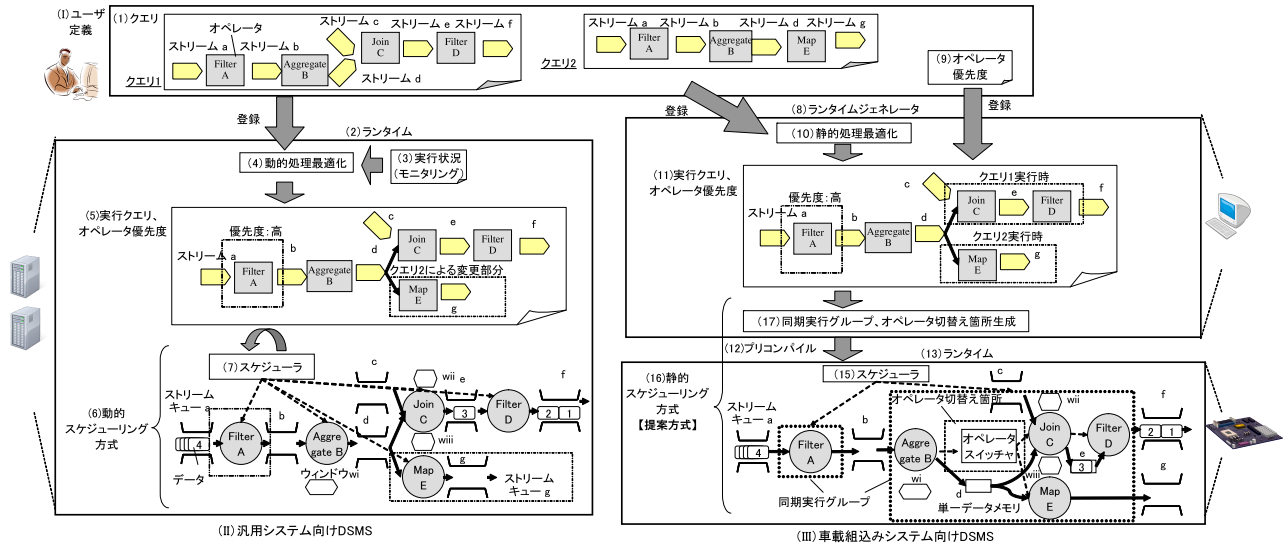

図 1　汎用システム/車載組込みシステム向けの DSMS
Fig. 1　DSMS for general-purpose systems/automotive embedded systems.

表 1　DSMS（Borealis）のオペレータの例
Table 1　Example of the operators in DSMS.

オペレータ	オペレータの動作
Filter	単一ストリームのデータを条件に従って，複数のストリームに出力
Map	単一ストリームのデータに対して，関数や演算を実行する．そして実行した結果をストリームとして出力
Join	複数ストリームの過去の特定期間のデータをウインドウとしてメモリ上に保持し，特定の条件によって保持したデータを結合し 1 つのストリーム上のデータとして出力
Aggregate	単一ストリームのデータを一定期間，ウインドウとしてメモリ上に保持し，保持したデータに対し集計関数を実行した結果をストリームとして出力

汎用システム向けの DSMS では，データの到着頻度や傾向などの実行状況（図 1（3））をモニタリングし，その実行状況に応じてオペレータの優先度を変更する．たとえば図 1（II）において，ストリーム a へのデータの到着頻度が大きく増加した場合，文献 [9] ではメモリ使用量の増加を防ぐため，データの削減率が高いオペレータ Filter A に高い優先度を設定し，他のオペレータよりも優先的に実行する．

2.1.2　動的スケジューリング方式

汎用システム向け DSMS では動的処理最適化を実現するために，実行するオペレータや，オペレータの実行順を動的に決定するスケジューリング（以下，動的スケジューリング方式）を行う．動的スケジューリング方式では，各ストリームを可変長のキュー（以下，ストリームキュー）として実現し，オペレータをストリームキューで接続する．そしてスケジューラが実行クエリ上のオペレータの接続関係やオペレータの優先度により，次に実行するオペレータを決定する．

このように動的スケジューリング方式では，動的処理最適化により変更される実行クエリに従って次に実行するオペレータを決定するため，最適化 1 を実現できる．たとえば図 1（7）では，クエリ 1 のみ登録されている場合にはスケジューラはオペレータ Filter A～Filter D を順番に呼び出す．そしてクエリ 2 の登録後には，動的処理最適化により実行クエリにオペレータ Map E が追加されるため，スケジューラは実行クエリを参照しオペレータ Aggregate B の実行後に，Map E を呼び出すことが可能になる．

またオペレータ間をストリームキューで接続しオペレータの優先度に従って，スケジューラが各オペレータを非同期に実行するため最適化 2 を実現できる．たとえばストリームキュー a のデータが増加したため，オペレータ Filter A に高い優先度を設定した場合，スケジューラ（図 1（7））は優先度が高いオペレータ Filter A を実行し続け，その出力データをストリームキュー b に蓄積する．そしてオペレータ Filter A の実行完了後に，ストリームキュー b に蓄積されたデータをオペレータ Aggregate B から順番に実行する．

2.2　DSMS の車載組込みシステムへの適用

本節では DSMS の車載組込みシステムへの適用について述べる．車載組込みシステムでは，電子制御ユニット（Electronic Control Unit，以下，ECU）が多数搭載されており，ECU ごとに異なるサプライヤがアプリケーションを開発するため，仕様が公開されず，データフォーマットも統一されていない．したがって各 ECU のアプリケーションで同一センサデータに対する前処理などを行っており，アプリケーションやセンサの増加により開発コストが増大する．また処理が重複するため CPU などのリソースの使用量が大きい．

そこで車載データ統合プラットフォーム [5] ではアプリケーションから，センサに依存する処理である車載データ

処理を切り離し，ストリーム処理言語（以下，SPL（Stream Processing Language））で記述する．SPL ではデータのフォーマットがストリームとして統一される．またストリームに対する処理が特定のオペレータがデータフローで接続されたクエリという形式をとるため，処理の流れが明示化され，クエリの部分的な変更や，他のクエリへの活用が比較的，容易である．このためクエリの保守性，再利用性は高く，異なるアプリケーションから車載データ処理を切り離し，共有化することができる．また SPL ではオペレータが SQL と同様に関係代数を基にしており，オペレータの共有，分解，結合による処理共有化も可能となる．

我々のプロジェクトでは先行研究 [11] で，汎用システム向け DSMS である Borealis [12] を用いて PC 上で SPL の車載データ処理への適用性について評価を行っていた．したがって先行研究では PC ではなく車載組込みシステムで DSMS のメモリ使用量や処理レイテンシを検証することが課題となっていた．そこで以下では車載組込みシステムの特性を説明し，その特性に対応するうえでの，Borealis を含む動的スケジューリング方式の課題について述べる．

2.3　車載組込みシステムの特性

DSMS の実行に関連する，汎用システムとは異なる車載組込みシステムの特性として以下の 1〜4 があげられる．

- 特性 1. 静的に処理内容を確定

車載組込みシステムではアプリケーションを固定し，車載データ処理の内容を静的に確定することにより，コードに対してリアルタイム性を含めテストを実施し信頼性を確保する．また車の走行状況や天候などにより，要求される車載データ処理が切り替わる場合にも，切り替える対象の処理内容は静的に決まる．さらに車々間通信データの増加など実行状況の変化により，静的に確定した車載データ処理をすべて実行できない場合にも，優先的に実行する処理を決定する．

- 特性 2. 小容量のメモリ

ハードウェアの大きさなどの物理的な制約や，価格の制約から小容量のメモリが搭載される．たとえば運転支援システムなどに搭載される V850E2/Px [13] では内蔵のメモリが 24〜80 KB と小さく，またノイズの防止のためメモリの外付けが可能でない．

- 特性 3. 動的なメモリ確保に非対応

車載組込みシステムでは，Linux などの汎用システム向け OS 以外にも多種多様な OS が存在する．そして運転支援システムなどで用いられるリアルタイム OS（RTOS）の中には，ヒープ領域が存在せず動的にメモリを確保できないものも存在する．

- 特性 4. 低レイテンシ

運転支援システムでは動作周期が約 10〜100 ms と短く，また車々間通信データを扱う場合にも周期 100 ms で最大で約 90 台の他車からのデータを扱う必要があり [14]，要求される処理レイテンシが小さい．

2.4　車載組込みシステムへの適用に向けた課題

特性 1〜4 を持つ車載組込みシステムへの適用に向けた，動的スケジューリング方式の課題を述べる．

- 課題 1. ストリームキューによるメモリ使用量増加

車載組込みシステムの 1 つである運転支援システムでは，特性 3 で述べたように必ずしも動的にメモリを確保できない．この場合，動的スケジューリング方式ではストリームキューを，リングバッファを用いて実現されるサイズが固定長のキューとして実現し，静的にメモリを確保する．しかしストリームキューに格納されるデータ数はオペレータの処理などにより変動するため，固定長のキューでは最大格納されるデータ数を想定し，そのデータ数分のメモリを静的に確保する必要があるためメモリ使用量が大きい．たとえば図 1（2）のランタイムではストリームキュー a〜g が存在するため，合計 7 個のストリームキューに対しメモリの確保が必要となる．したがって特性 2 で述べた小容量のメモリに搭載することが課題となる．

一方，カーナビゲーションでは主に Linux などの汎用システム向け OS が用いられ，動的なメモリ確保が可能である．この場合にはストリームキューはサイズが可変長のキューとして実現される．したがってストリームキューごとに最大格納されるデータ数分のメモリを確保する必要はないが，この場合，実行状況に従ってメモリ使用量が大きく変動する．たとえばオペレータ Join は結合条件に従って 1 つの入力データから多数のデータを出力する．したがって図 1（2）のランタイム上のストリームキュー e には，オペレータ Join C の出力データが 1 度に多数格納される．この場合，後段のオペレータ Filter D によりストリームキュー e のデータを間引きする場合にも，オペレータ Join C 実行後，一時的にメモリ使用量が増加する．

- 課題 2. スケジューラによる処理レイテンシ増加

動的スケジューリング方式ではスケジューラを介してオペレータが呼び出される．スケジューラは実行クエリにおけるオペレータの接続関係だけでなく，オペレータの優先度も考慮して実行するオペレータを決定する必要があり，またすべてのオペレータはスケジューラを通して呼び出されることから処理レイテンシが大きい．したがって特性 4 で述べた低レイテンシを満たすことが課題となる．

3.　車載組込みシステム向け DSMS

我々のプロジェクトで検討してきた eDSMS [15] のコンセプトを図 1（III）に示す．eDSMS では特性 1 をふまえ，車の走行状況などにより切り替わるクエリを含め，実行するすべてのクエリを静的に定義することを前提とする．定義されたクエリはランタイムジェネレータ（図 1（8））に

登録し，静的な処理最適化（以下，静的処理最適化（図 1 (10)））により実行クエリやオペレータの優先度（図 1 (11)）を決定する．そして静的に決定した実行クエリやオペレータの優先度の情報に従ってプリコンパイル（図 1 (12)）によりランタイム（図 1 (13)）を生成することで，メモリ使用量および処理レイテンシの増加を防ぐ．

eDSMS の静的処理最適化については，文献 [16] に示すように，登録されたすべてのクエリに対し，最適化 1 と同様に複数クエリで共通するオペレータやストリームを共有化し実行クエリを生成する．たとえば図 1 (III) では，クエリ 1，2 ともにランタイムジェネレータに登録し，静的処理最適化によりクエリ 1，2 に共通するオペレータ Filter A，Aggregate B を共有化する．そしてクエリ 1 実行時にはオペレータ Join C，Filter D に接続し，またクエリ 2 実行時にはオペレータ Map E を接続する．また車載組込みシステムではデータの到着頻度や傾向などが仕様として定まっていることが多いため，オペレータの優先度（図 1 (9)）については，一般的な RTOS と同様に仕様に従ってユーザが静的に設定する．本稿では，静的に確定した実行クエリやオペレータの優先度を用いて，ランタイムのメモリ使用量および処理レイテンシを削減する方法を検討する．

4. 静的スケジューリング方式

本章では，本稿で提案する静的スケジューリング方式について述べる．

4.1 コンセプト

静的スケジューリング方式 (図 1 (16)) では，静的に決定した実行クエリおよびオペレータの優先度に従って，ランタイムのオペレータ間のストリームキューや，スケジューラによるオペレータの呼び出しを削減する．まずランタイムジェネレータが，各オペレータの優先度などに従って複数のオペレータをグループとして抽出する．そして同一グループのオペレータ間のストリームキューを除去し，ランタイムでは各オペレータを同期して実行することにより，ストリームキューによるメモリ使用量の増加を防ぐ（課題 1 の解決）．また同じくランタイムジェネレータにより，グループ内で，実行するオペレータが動的に切り替わらない箇所を検出する．そしてランタイムでは，検出箇所ではスケジューラを介さず各オペレータが次に実行するオペレータを直接，呼び出すことにより，スケジューラによる処理レイテンシの増加を防ぐ（課題 2 の解決）．

4.2 フレームワーク

静的スケジューリング方式のフレームワークを図 1 (16) に示す．静的スケジューリング方式では実行クエリおよび，オペレータの優先度（図 1 (11)）により，各オペレータを同期して実行するグループ（以下，同期実行グループ），動的にオペレータが切り替わる箇所（以下，オペレータ切替え箇所）を生成する（図 1 (17)）．そしてランタイム（図 1 (13)）ではスケジューラ（図 1 (15)）により同期実行グループ単位のスケジューリングを行う．また同期実行グループ内ではオペレータ切替え箇所以外では，各オペレータがその出力データを入力とするオペレータ（以下，後段オペレータ）を呼び出す．そしてオペレータ切替え箇所では，オペレータ間に配置されるオペレータスイッチャにより，接続されているオペレータを呼び出す．さらに同期実行グループ内のオペレータはストリームキューを介さず，単一のデータサイズ分のメモリ（以下，単一データメモリ）を用いてデータを受け渡しする．単一データメモリは，同一の同期実行グループに属するオペレータ間を接続するすべてのストリームに対して，ストリームキューの代わりに割り当てる．たとえば図 1 (13) ではストリーム d，e に対応する単一データメモリを確保する．以下では静的スケジューリング方式の詳細について述べる．

4.3 eDSMS のオペレータ

静的スケジューリング方式の前提として，eDSMS のオペレータについて説明する．eDSMS では表 1 に示すオペレータを持つ．eDSMS の各オペレータはストリームから時刻順に 1 つずつ入力データを読み出し，オペレータごとに決められた処理を行い，その入力データの時刻を付加した出力データを生成する．なおオペレータ Join のように複数ストリームからデータを入力する場合にも，複数ストリームの中から時刻が最も古い入力データを順次，読み出し，その出力データには入力データの時刻を付加する．

また eDSMS の各オペレータはストリームのデータを入出力とし，STREAM などとは異なりウィンドウ演算を持たず，オペレータ Join，Aggregate ではオペレータ内にウィンドウを持つ．そしてオペレータ Join，Aggregate では件数または，時間と最大保持する件数を指定して，ウィンドウに過去の入力データを指定数分，保持する．たとえば図 1 (13) ではオペレータ Aggregate B は 2 件分のデータを保持するウィンドウ wi を持ち，またオペレータ Join C ではストリーム c，d の各 1 件分のデータを保持するウィンドウ wii，wiii を持つ．なおウィンドウを持つオペレータの場合にも，ウィンドウ内の過去の入力データではなく，ストリームから最後に読み出された最新の入力データの時刻を出力データに付加する．

4.4 同期実行グループ，オペレータ切替え箇所の生成

本節では，同期実行グループおよびオペレータ切替え箇所を生成する方法について述べる．

4.4.1 同期実行グループ

動的スケジューリング方式ではすべてのオペレータを非同期に実行可能とすることで，オペレータの優先度の動的

な変更に対応していた．一方，静的スケジューリング方式ではオペレータの優先度を静的に確定させ，そのオペレータの優先度と実行クエリから 1，2 の両方の条件を満たすオペレータのグループを抽出し，同期実行グループとする．

1　グループ内のすべてのオペレータの優先度が同一
2　グループ内の各オペレータは，同グループ内のオペレータのみを介して接続

たとえば図 1 (11) の実行クエリでは，オペレータ Filter A の優先度のみが高く，その他のオペレータの優先度は低い．また低優先度のオペレータはすべて，低優先度のオペレータのみを介して接続し，低優先度のオペレータ間に低優先度以外のオペレータが挟まることはない．したがって**図 2** に示すように，オペレータ Filter A のみから構成される同期実行グループ S1 と，その他のオペレータから構成される同期実行グループ S2 を生成する．

なお優先度の設定によっては処理レイテンシが大きくなることもあり，ユーザがオペレータ単位で優先度を設定するのが難しい場合もある．ユーザの優先度設定を支援するユーザインタフェースなどの検討については今後の課題とする．

4.4.2　オペレータ切替え箇所

動的スケジューリング方式ではすべてのオペレータが動的に切り替わることを想定していた．一方，静的スケジューリング方式では，クエリの違いによりオペレータの後段オペレータ，すなわちオペレータの出力データを入力とするオペレータが変わらない場合には，オペレータ切替え箇所とはせずオペレータが後段オペレータを呼び出す．そしてオペレータの後段オペレータが変わる場合のみ，オペレータと後段オペレータの間にオペレータ切替え箇所を生成する．

オペレータ切替え箇所の生成アルゴリズムを**図 3** に示す．同期実行グループ gh のオペレータ切替え箇所を生成するために，まずグループ gh を構成するオペレータ ol と，その出力ストリーム sk で接続された後段オペレータ oj を抽出する（図 3，1〜4 行）．そしてオペレータ oj がグループ gh に属し，かつオペレータ oj と ol が属するクエリの集合が異なる場合に，ストリーム sk にオペレータ切替え箇所を生成する（図 3，5〜7 行）．

たとえば図 1 (11) の実行クエリでは，オペレータ Join C と，その出力ストリーム e を介する後段オペレータ Filter D はいずれもクエリ 1 に属するため，ストリーム e にオペレータ切替え箇所を生成しない．一方，オペレータ Aggregate B はクエリ 1，2 の両方に属し，ストリーム d を介する後段オペレータ Join C，Map E はそれぞれクエリ 1，2 の一方に属するため，いずれもオペレータ Aggregate B とは属するクエリの集合が一致しない．したがって図 2 に示すように，ストリーム d にオペレータ切替え箇所 C1 を生成する．

なおランタイムジェネレータによるオペレータ切替え箇所の生成方法は他にも様々な方法が考えられる．またユーザが明示的にオペレータ切替え箇所を生成する場合も考えられるが，オペレータ切替え箇所の生成方法の詳細検討については今後の課題とする．

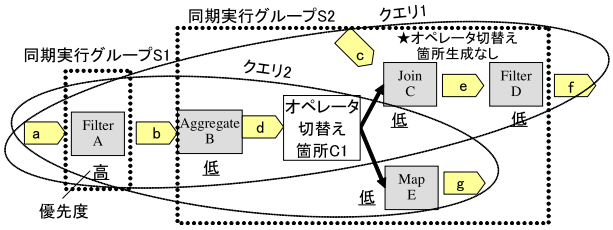

図 2　同期実行グループ，オペレータ切替え箇所の生成
Fig. 2　Generations of synchronous execution groups and operetor switching points.

```
1  FOR gh IN 同期実行グループの集合:
2    FOR ol IN ghを構成するオペレータの集合:
3      FOR sk IN olの出力ストリームの集合:
4          FOR oj IN olのskで接続する後段オペレータの集合:
5            IF ojが同期実行グループghに属する AND
6               ojとolが属するクエリの集合が異なる:
7                skにオペレータ切替え箇所を生成;
```

図 3　オペレータ切替え箇所の生成アルゴリズム
Fig. 3　The algorithm of generating the operetor switching points.

4.5　同期実行グループ内の基本的な動作

同期実行グループ内の基本的な動作について述べる．同期実行グループは，スケジューラから入力データを与えられることで実行が開始する．各オペレータには対応する関数があり，スケジューラはまずその入力データを受け取るオペレータ（以下，先頭オペレータ）の関数を呼ぶ．先頭オペレータではその出力データをストリームキューに蓄積せずに，単一データメモリに格納し，静的に確定している後段オペレータの関数を呼び出し，またその後段オペレータの関数終了後，他の後段オペレータの関数を呼び出す．このようにして呼び出したすべての後段オペレータの関数が終了したら，先頭オペレータの関数を終了する．先頭オペレータの後段オペレータやその他のオペレータの関数も同様に動作し，すべてのオペレータの関数が終了すると，スケジューラに制御が戻る．スケジューラでは継続的に入力データを監視し，再び同期実行グループに入力データを与える．なおオペレータの出力データを他の同期実行グループのオペレータや外部のアプリケーションに渡す場合には，単一データメモリではなくストリームキューに出力データを格納する．

たとえば**図 4** では，図 2 に示す同期実行グループ S2 内の動作を示す．図 4 に示すように同期実行グループ S2 の入力データとしてデータ α が与えられた場合，スケジューラはデータ α を入力データとするオペレータ Join C の関数を呼び出す．オペレータ Join C ではデータ α をウィン

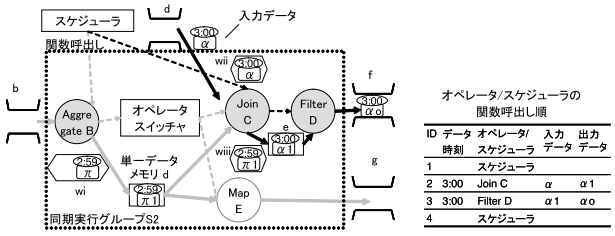

図 4　同期実行グループ内の基本的な動作
Fig. 4　Basic behavior in the synchronous execution group.

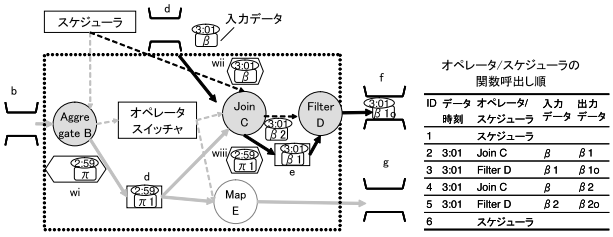

図 5　複数データを出力するオペレータ
Fig. 5　The operator outputting multiple data.

ドウ wii に格納し，ウィンドウ wiii 内のデータ π1 と結合し出力データ α1 を生成する．そして出力データ α1 をストリームキューではなく，単一データメモリ e に格納し，後段オペレータ Filter D の関数を呼び出す．オペレータ Filter D では出力データ αo のストリームキュー f への格納後，後段オペレータが存在しないため関数を終了する．同様にオペレータ Join C においてもオペレータ Filter D 以外に後段オペレータが存在しないため関数を終了し，スケジューラに制御を戻す．

4.6　同期実行グループ内のストリームキューの除去

静的スケジューリング方式では同期実行グループ内のストリームキューを除去するために，前節で述べたように，各オペレータの実行後に必ず後段オペレータを実行する．そして複数ストリームを入力とするオペレータおよび，1 つの入力データに対し複数データを出力するオペレータについては以下のように動作することで，各オペレータにおいて入出力データを格納するストリームキューを不要にする．

4.6.1　複数ストリームを入力とするオペレータ

DSMS では一般的に，各オペレータは時刻順に入力データを処理する．したがって従来の DSMS では，オペレータ Join など，複数ストリームのデータを入力するオペレータ（以下，複数入力ストリームオペレータ）では入力データをストリームキューに格納し，他のストリームキューに格納される時刻が古い入力データを待つ．そして時刻が古い入力データの到着後，時刻順に入力データを整列し，オペレータで処理する．逆にいえば，もしストリームキューに入力データを格納しない場合，1 つのストリームの入力データを複数保持することができず，他のストリームの入力データを待つ間に，同一ストリームの古い入力データが上書きされる．

そこで静的スケジューリング方式では，以下のように動作することにより，複数入力ストリームオペレータの入力データの時刻順整列を不要にし，入力データをストリームキューではなく単一データメモリに格納可能にする．まず 1 つの同期実行グループの入力データをすべて時刻順に与えることで，先頭オペレータは時刻順整列なしにその入力データを処理可能とする．また eDSMS では STREAM などの従来の DSMS と同様にオペレータの出力データは入力データの時刻を引き継ぐ．したがって先頭オペレータの後段オペレータやその他のオペレータの入力データも時刻順に与えられ，先頭オペレータと同様に時刻順整列なしに入力データを処理することが可能である．たとえば図 4 においてオペレータ Join C の関数が呼ばれた際に，その入力データ α の時刻 3:00 よりも古い入力データは同期実行グループ S2 に到着しない．したがって単一データメモリ d にも入力データ α よりも古い入力データが生じないため，時刻順整列することなくオペレータ Join C により入力データ α を処理できる．

4.6.2　複数データを出力するオペレータ

オペレータ Join など，1 つの入力データに対して複数の出力データを生成する可能性のあるオペレータ（以下，複数出力オペレータ）では，出力データを単一データメモリに格納するために，以下のように動作する．まず複数出力オペレータでは複数の出力データのうち，従来の DSMS とは異なり 1 つの出力データのみを生成し，その出力データを単一データメモリに格納後，後段オペレータの関数を呼び出す．そして後段オペレータの関数終了後，その他の出力データを 1 つ生成し単一データメモリへの格納を行い同様に後段オペレータの関数を呼び出す．このようにして出力データ数分，後段オペレータの関数を呼び出し，すべての関数が終了した後に，複数出力オペレータの関数を終了する．

たとえば**図 5** では同期実行グループ S2 の入力データとしてデータ β が与えられた場合である．データ β はデータ α とは異なり，オペレータ Join C によりデータ β1，β2 の 2 つの出力データを生成するとする．そこでオペレータ Join C ではまずデータ β1 のみを生成し単一データメモリ e に格納し，オペレータ Filter D の関数を呼び出す．そしてオペレータ Filter D がその出力データ β1o をストリームキュー f に格納し，関数を終了した後に，再びオペレータ Join C の出力データ β2 を生成し，単一データメモリ e に格納し，オペレータ Filter D の関数を呼び出す．

4.6.3　詳細な動作例

同期実行グループ内の単一データメモリを活用したオペレータの実行方法の詳細について，**図 6** に示すクエリ 3 を用いて説明する．クエリ 3 のオペレータの実行方法につ

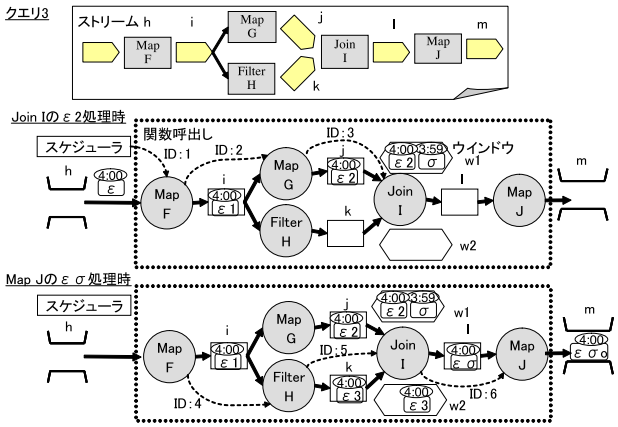

図 6　詳細な動作例

Fig. 6　The example of the detail bahavior.

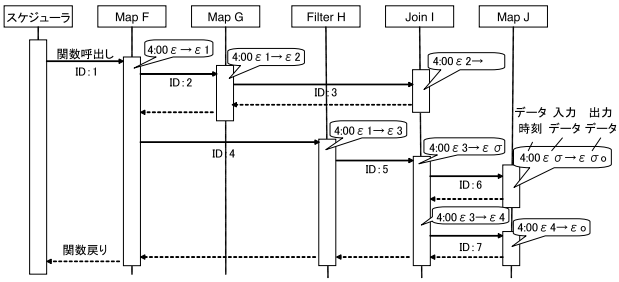

図 7　詳細な動作例のシーケンス図

Fig. 7　The sequence graph of the example of the detail bahavior.

いて，図 6 で動作例を示し，**図 7** にシーケンス図を示す．前提としてクエリ 3 のオペレータの優先度をすべて同一とし，したがってすべてのオペレータが同じ同期実行グループに属するとする．

まず 4.5 節で述べたようにスケジューラにより同期実行グループの先頭オペレータ Map F の関数が呼び出される．そして Map F では単一データメモリ i に出力データ $\varepsilon 1$ を格納し，その後段オペレータ Map G の関数を呼び出す．その後，Map G は出力データ $\varepsilon 2$ を単一データメモリ j に格納し後段オペレータ Join I の関数を呼び出す．

4.6.1 項で述べたように先頭オペレータ Map F は時刻順に入力データが与えられ，また Map G など各オペレータの出力データは入力データの時刻を引き継ぐため，Join I の入力データも時刻順に与えられる．したがって Join I にはデータ $\varepsilon 2$ の時刻 4:00 よりも古いデータが入力されることがないため，Map G からの呼び出し時にデータ $\varepsilon 2$ を Join I で処理することが可能である．Join I では単一データメモリ j，k の入力データを格納するウィンドウ w1，w2 を持ち，異なるウィンドウのデータと結合し出力データを生成する．ここではデータ $\varepsilon 2$ はウィンドウ w1 に格納されるためウィンドウ w2 内のデータと結合されるが，ウィンドウ w2 にデータが存在しない．したがって Join I では出力データを生成することなく関数が終了する．そして Join I の関数の終了後，Map G の関数に制御が戻り，Map G にも Join I 以外に後段オペレータが存在しないため，Map F の関数に制御が戻る．その後，Map F ではもう 1 つの次段オペレータ Filter H の関数を呼び出し，Filter H ではデータ $\varepsilon 3$ を出力し再び Join I の関数を呼び出す．

Join I では，データ $\varepsilon 3$ がウィンドウ w2 に格納されるためウィンドウ w1 内のデータ σ，$\varepsilon 2$ と結合し，出力データを生成する．4.6.2 項で述べたように，Join I では 1 つずつ出力データを生成し，その出力データを単一データメモリに格納し，次段オペレータを呼び出す．この場合，まずデータ σ と結合することでデータ $\varepsilon\sigma$ を生成し，単一データメモリ l に格納後に Map J の関数を呼び出す．そして Map J の関数から Join I の関数に制御が戻った後に，データ $\varepsilon 3$ がデータ $\varepsilon 2$ と結合することでデータ $\varepsilon 4$ を生成し同様に Map J を呼び出す．こうして Join I のすべての出力データの処理が完了すると，Join I の関数が終了し，続いて Filter H，Map F の関数終了後，スケジューラに制御が戻る．

以上のように同期実行グループ内のオペレータを実行することにより，4.3 節で述べた eDSMS のオペレータのセマンティックスを満たせる．すなわちオペレータ Join など複数のストリームを入力とする場合にも，単一データメモリから時刻順に 1 つずつ入力データを読み出すことができる．そしてその入力データ，あるいはオペレータ Join などではウィンドウに保持された過去の入力データも用いて，オペレータごとに決められた処理を行い，最新の入力データの時刻を付加した出力データを生成できる．

4.6.4　ストリームキュー除去によるメモリ使用量削減

静的スケジューリング方式におけるストリームキュー除去によるメモリ使用量削減の効果について述べる．前述のように同期実行グループ内では，オペレータ Join を含む，eDSMS のオペレータ間のストリームキューが不要となり代わりに単一データメモリにより実現する．単一データメモリは固定長のメモリであるため，動的なメモリの確保が不要となる．また単一データメモリは単一のデータサイズ分のメモリ領域であるためメモリ使用量も小さくなる．たとえば図 6 ではストリーム i，j，k，l を単一データメモリとして実現している．一方，動的スケジューリング方式では，オペレータから同時の複数のデータがストリームキューに格納される可能性がある．たとえば 4.6.2 項で述べたようにオペレータ Join では複数の出力データをストリームキューに格納する．したがって動的スケジューリング方式ではストリームキューを単一データサイズに抑えることはできず，メモリ使用量は大きい．

なおオペレータの出力データが分岐しない場合など，必ずしもすべてのストリームに対して単一データメモリを確保する必要はなく，複数のストリームに対し単一データメモリを共通化することも可能である．ただし 1 つの単一

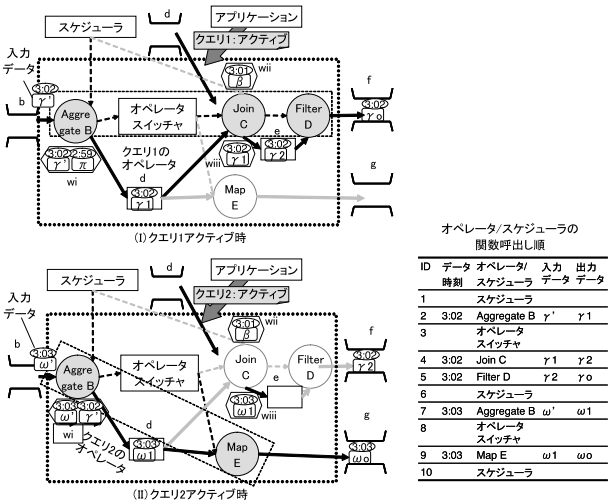

図 8　オペレータの切替え

Fig. 8　Switching of operators.

データメモリのサイズは小さく，オペレータ数が多くないとメモリ使用量の削減効果はさほど見込めないことから，単一データメモリ数の削減については今後の課題とした．

また他の同期実行グループや外部のアプリケーションと接続するストリームについては従来の DSMS と同様にストリームキューが必要となる．たとえば図 6 ではストリーム h，m はストリームキューとして実現する．またオペレータ Join，Aggregate についてもウィンドウのためのメモリ領域が必要となり，ウィンドウのためのメモリ使用量は動的スケジューリング方式と変わらない．たとえば図 6 に示すオペレータ Join I はウィンドウ w1，w2 を持ち，それぞれデータ数分のメモリが必要となる．

4.7　同期実行グループ内のオペレータの切替え

同期実行グループ内でオペレータが動的に切り替わる場合には，オペレータスイッチャにより次に実行するオペレータを決定する．オペレータスイッチャはオペレータ切替え箇所に静的に挿入され，オペレータから関数として呼ばれる．そしてアクティブとなっているクエリに従って次に実行するオペレータを決定し，そのオペレータの関数を呼び出す．なお同期実行グループ内ではオペレータを同期して実行するため，動的スケジューリング方式とは異なり，オペレータを動的に切り替える場合にもそのオペレータの入出力データをストリームキューではなく，単一データメモリに格納する．

例では図 2 に示すようにオペレータ切替え箇所 C1 が生成される場合，**図 8** に示すようにオペレータ Aggregate B からオペレータスイッチャが呼ばれる．そして図 1（1）のクエリ 1 が，アプリケーションからの指定によりアクティブになる場合，オペレータスイッチャはクエリ 1 のオペレータ Join C の関数を呼び出す（図 2（I））．一方，クエリ 2 がアクティブになる場合には，クエリ 2 のオペレータ Map E の関数を呼び出す（図 2（II））．なおオペレータ Aggregate B の出力データはストリームキューでなく単一データメモリ d に格納され，オペレータ Join C，Map E は単一データメモリ d から入力データを読み出す．

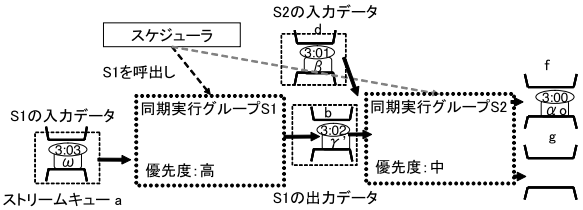

図 9　同期実行グループのスケジューリング

Fig. 9　Scheduling of the synchronous execution groups.

4.8　同期実行グループのスケジューリング

本節では同期実行グループのスケジューリング方法について説明する．4.5.1 項で述べたように，各同期実行グループには入力データを時刻順に与える必要がある．そこでスケジューラでは同期実行グループごとに読み出し対象のストリームキューから最も時刻が古いデータ（最古データ）を抽出する．また同期実行グループの入力データを出力する同期実行グループ（前段同期実行グループ）を抽出し，その前段同期実行グループの出力データの時刻を参照する．そして最古データの時刻が，すべての前段同期実行グループの出力データの時刻よりも古ければ，より時刻が古いデータが新たに読み出し対象のストリームキューに格納されることがないため，最古データを同期実行グループの入力データとする．このようにして各同期実行グループの入力データを算出した後に，スケジューラは，入力データが存在し，かつ優先度が最も高い同期実行グループを呼び出す．

たとえば**図 9** では，図 2 に示すように同期実行グループ S1，S2 を生成した場合の同期実行グループのスケジューリングを示す．図 9 では，時刻 3:01，3:02 にそれぞれデータ β，γ' が到着したが，より時刻が古いデータを処理していたために，データ β，γ' が処理されず時刻 3:03 にデータ ω が到着した場合を示す．この場合，同期実行グループ S1 では，ストリームキュー a にデータ ω のみが存在し，同期実行グループ S1 の前段同期実行グループは存在しないため，データ ω を同期実行グループ S1 の入力データとする．一方，同期実行グループ S2 では，データ β，γ' がそれぞれストリームキュー d，b に存在し，データ β の時刻は 3:01 であるため，データ γ' の時刻 3:02 よりも古い．また同期実行グループ S2 の前段同期実行グループ S1 の出力データも時刻 3:02 である γ' であるため，それよりも時刻が古いデータ β を同期実行グループ S2 の入力データとする．したがって同期実行グループ S1，S2 はそれぞれ入力データ ω，β が存在するため，より優先度が高い同期実行

表 2　評価環境
Table 2　Evaluation environment.

名称	ハードウェア	OS	車載データ処理
運転支援システム	Altera 社 Nios II 開発キット，Cyclone III エディション 3C25（CPU：Nios II/f コア，メモリ：32 MByte）	TOPPERS/ATK2	先行車両認識
カーナビゲーション	ZMP 社 RoboCar R 1/10（CPU：AMD Geode LX800 Processor 500 MHz，メモリ：512 MByte）	Linux（Fedora 10）	周辺車両認識

グループ S1 を呼び出す．

5. 評価

本章では静的スケジューリング方式の評価について述べる．

5.1 評価方法，環境

eDSMS とその静的スケジューリング方式を実装し，動的スケジューリング方式とメモリ使用量，処理レイテンシを比較する．評価環境は**表 2** に示すような運転支援システムおよびカーナビゲーションを想定した環境とする．以下では評価に用いた動的スケジューリング方式および，運転支援システムおよびカーナビゲーションを想定した環境のハードウェア/OS の構成，評価に用いるクエリについて説明する．

5.1.1 動的スケジューリング方式

評価に用いた動的スケジューリング方式では，優先度ごとにオペレータをグループ化し，静的スケジューリング方式と同様に，優先度が高いオペレータのグループを先に実行する．また同一優先度のオペレータのグループに対しては，First-Come-First-Served となっており先着のデータから順番に 1 つずつ最後まで実行していく．たとえば図 1 (5) に示す実行クエリの場合には，ストリームキュー a にデータが存在する限り優先度が高いオペレータ Filter A を実行し続ける．そしてストリームキュー a にデータがなくなった場合に，ストリームキュー b，c から時刻が最も古いデータを抽出し，そのデータに対しオペレータ Aggregate B，Join C，Filter D，Map E を続けて実行し，その後に時刻が次に古いデータを実行する．

5.1.2 運転支援システム

運転支援システムを想定した環境としては，表 2 に示すようにハードウェアは Altera 社の FPGA ボード [17] を活用し，OS として AUTOSAR に準拠した RTOS である TOPPERS/ATK2 を搭載した．アプリケーションは，先行車両を追随走行するアプリケーション（以下，追随走行アプリ）および，先行車両の接近を防止するアプリケーション（以下，接近防止アプリ）を想定する．また eDSMS 上で動作させる車載データ処理は先行車両認識に関するクエリ（**図 10**）である．先行車両認識クエリでは，前方のレーダやカメラ，自車速，ステアリングの情報を入力源とし，**表 3** に示すオペレータにより追随走行/接近防止用の先行車両有無，先行車両の速度，先行車両との相対速度/車間距離を計算し，各アプリケーションに配信する．運転支援システムの動作周期を想定し，各入力源には 10 ms ごとにデータを与え，先行車両認識クエリが実行される．

また接近防止アプリは追随走行アプリよりも緊急度が高いアプリケーションであるため，先行車両認識クエリにおいて，接近防止アプリへの出力データの計算に用いるオペレータの優先度を，追随走行アプリへの出力データのみの計算に用いるオペレータよりも高く設定する．これにより図 10 に示すように，静的スケジューリング方式ではランタイムジェネレータで同期実行グループを生成し，同図に

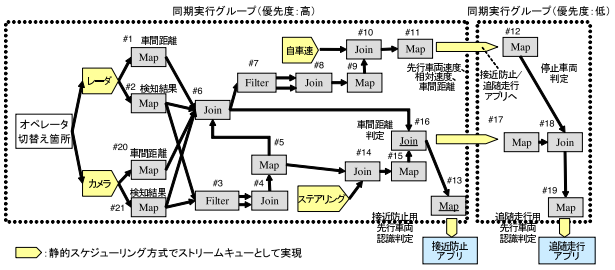

図 10　先行車両認識クエリ
Fig. 10　Query for recognition of forward vehicles.

表 3　先行車両認識クエリの各オペレータの処理内容
Table 3　The operator behaviors in the forward vehicle recognition query.

ID	オペレータ	処理内容
1	Map	レーダの情報から先行車両の車間距離を抽出
2	Map	レーダの情報から先行車両の検知結果を抽出
3	Filter	先行車両の検知結果を，検知，非検知情報に振分け
4	Join	最近の検知開始時刻（ウィンドウサイズ（ws）：1 個）と最新の非検知開始時刻（ws：1 個）を算出
5	Map	検知/非検知の継続時間を算出
6	Join	最新の検知/非検知時間（ws：1 個）と最新の車間距離（ws：1 個）を結合
7	Filter	現在および 1 つ前の車間距離を抽出
8	Join	現在の車間距離（ws：1 個）および 1 つ前の車間距離（ws：1 個）を単一データに結合
9	Map	現在および 1 つ前の車間距離から相対速度を算出
10	Join	最新の相対速度（ws：1 個）と最新の自車速度（ws：1 個）を単一データに結合
11	Map	相対速度と自車速度から先行車両の速度を算出
12	Map	先行車両の速度が 5 m/s 以内なら停止車両と判定
13	Map	車間距離が 30 m 以内の場合，接近防止用先行車両検出と判定
14	Join	最新の検知/非検知時間（ws：1 個）と最新のステアリング情報（ws：1 個）を結合
15	Map	検知時間が 2 s 以上または，非検知時間が 2 s 以内かつステアリング角度 ±20 度以上なら先行車両判定
16	Join	最新の検知/非検知時間（ws：1 個）と最新の車間距離（ws：1 個）を結合
17	Map	車間距離 100 m 以上の場合に先行車両非検出と判定
18	Join	最新の先行車両認識判定（ws：1 個）と最新の停止車両情報（ws：1 個）を単一データに結合
19	Map	停止車両の場合に，先行車両非検出と判定
20	Map	カメラの情報から先行車両の車間距離を抽出
21	Map	カメラの情報から先行車両の検知結果を抽出

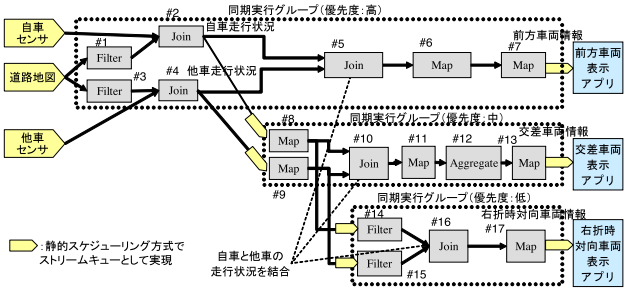

図 11　周辺車両認識クエリ
Fig. 11　Query for recognition of surrounding vehicles.

表 4　周辺車両認識クエリの各オペレータの処理内容
Table 4　The operator behaviors in the surrounding vehicle recognition query.

ID	オペレータ	処理内容
1	Filter	自車の地図情報を抽出
2	Join	自車の最新センサ情報（ウィンドウサイズ（ws）：1 個）と最新地図情報（ws：1 個）を結合
3	Filter	他車の地図情報を抽出
4	Join	他車最大 100 台分のセンサ情報（ws：100 個）と地図情報（ws：100 個）を結合
5	Join	他車最大 100 台分の情報（ws：100 個）から前方車両（同一道路，同一レーンを走行，自車の 1 つ前の車両）を抽出し，最新自車情報（ws：1 個）と結合
6	Map	自車と前方車両の距離，SDA（Stopping Distance Algorithm）の値を算出
7	Map	アプリに配信する前方車両情報の形式に整形
8	Map	自車と交差点との距離を算出
9	Map	他車と交差点との距離を算出
10	Join	他車最大 100 台分の情報（ws：100 個）から交差車両（次交差点が同一かつ交差道路走行）を抽出し，最新自車情報（ws：1 個）と結合
11	Map	自車，交差車両と交差点の位置から勾配の値を算出
12	Aggregate	勾配の移動平均（ws：300 個）を算出
13	Map	アプリに配信する交差車両情報の形式に整形
14	Filter	交差点から 100 m 以内の距離の自車を抽出
15	Filter	交差点から 100 m 以内の距離の他車を抽出
16	Join	他車最大 100 台分の情報（ws：100 個）から右折時対向車両（次交差点が同一かつ対向道路走行かつ右方向指示あり）を抽出し，最新自車情報（ws：1 個）と結合
17	Map	アプリに配信する右折時対向車両情報の形式に整形

示すストリームキュー以外を除去する．また先行車両の検知結果と車間距離を計算するオペレータを，レーダまたはカメラを入力源とするオペレータに切替え可能とするために，図 10 に示すように，ユーザがオペレータ切替え箇所を生成する．

5.1.3　カーナビゲーション

カーナビゲーションを想定した環境としては，表 2 に示すようにハードウェアは ZMP 社の RoboCar [18] を活用し，OS は Linux（Fedora 10）とする．アプリケーションは，前方車両および，交差車両，右折時対向車の走行情報を表示するアプリケーションを想定する．eDSMS 上で動作させる車載データ処理としては周辺車両認識に関するクエリ（**図 11**）である．周辺車両認識クエリでは，自車センサ情報，車々間通信により受け取る他車センサ情報，道路地図情報を入力源とし，**表 4** に示すオペレータにより前方車両および，交差点における交差車両，右折時の対向車両に関する情報を計算し，アプリケーションに配信する．車々間通信を想定し他車センサ情報は 100 ms ごとに 90 個のデータを与え，また自車センサ情報，道路地図情報には 100 ms に 1 個のデータが与えられ，周辺車両認識クエリが実行される．なお道路地図情報のストリームには，道路地図情報を格納する RDB から定期的に自車周辺の道路地図情報を問い合わせ，その問合せ結果が入力される．

周辺車両認識クエリのオペレータの優先度は，前方車両，交差車両，右折時対向車両を表示するアプリケーションの順に，その出力データの計算に用いるオペレータに高い優先度を設定する．したがって静的スケジューリング方式では図 11 に示すように，同期実行グループを生成し，同図に示すストリームキュー以外を除去する．

5.2　メモリ使用量

運転支援システムおよび，カーナビゲーションにおける，ランタイムに割り当てられた静的データ領域のメモリのサイズおよび，クエリ実行時にヒープ領域に確保されるメモリサイズについて評価する．

5.2.1　運転支援システム

運転支援システムに搭載される TOPPERS/ATK2 ではヒープ領域を持たず動的なメモリ確保ができないため，各ストリームキューを固定長のキューとして実現し，静的にメモリを確保する．また先行車両認識クエリでは，動的スケジューリング方式ではストリームと同数の 18 個のストリームキューが必要となるのに対し，静的スケジューリング方式では図 10 に示す 8 個のストリームキューのみを持つ．その結果，静的データ領域のメモリ使用量は**表 5** のようになった．静的スケジューリング方式は動的スケジューリング方式と比較し，静的データ領域のメモリ使用量が 54.7%削減され，固定長のキューの削減効果を確認できた．なお表 5 の結果は，各ストリームキューにデータ 100 個分のメモリを割り当てた場合である．

また文献 [13] に示すように V850E2/Px の RAM サイズは 24〜80 KB である．静的スケジューリング方式では，静的データ領域のメモリ使用量は約 32 KB であり，そのほかに必要となるスタック領域のメモリ使用量は 1 Kbyte 未満であることから，車載組込みシステムの RAM に搭載できる見込みを得た．

なおオペレータ Join（図 10 ID: 4，6，8，10，14，16，18）のウィンドウは静的データ領域に割り当てられ，静的スケジューリング方式，動的スケジューリング方式はともに合計で 440 byte となった．

5.2.2　カーナビゲーション

カーナビゲーションでは Linux を搭載するため，ストリームキューを可変長のキューとして実現し，ストリームキューのデータはヒープ領域に確保する．また周辺車両認

表 5　運転支援システムにおけるメモリ使用量
Table 5　Memory usage in the drive assist system.

	動的スケジューリング	静的スケジューリング
静的データ領域	72,676 byte	32,952 byte
ヒープ領域	-	-

表 6　カーナビゲーションにおけるメモリ使用量
Table 6　Memory usage in the car navigation.

	動的スケジューリング	静的スケジューリング
静的データ領域	74,348 byte	63,712 byte
ヒープ領域	平均 1,167 byte	平均 576 byte
	（157～4,026 byte）	（113～1,309 byte）

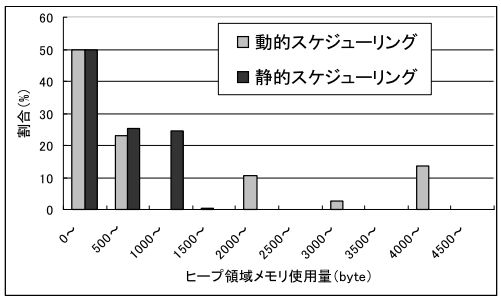

図 12　ヒープ領域のメモリ使用量の分布
Fig. 12　Distribution of the heap memory usage.

識クエリでは，動的スケジューリング方式ではストリームキューがストリームと同数の 20 個となるのに対し，静的スケジューリング方式では図 11 に示す 10 個のストリームキューを持つ．そしてランタイムへの入力データごとにその処理に使用されるヒープ領域のメモリサイズを測定し，その結果を**表 6** に示す．静的スケジューリング方式により平均メモリ使用量が 50.6%削減され，またメモリ使用量の最大値，最小値がそれぞれ 67.5%，28.0%削減された．**図 12** はヒープ領域のメモリ使用量の分布を示す．図 12 に示すように，メモリ使用量が 500 byte 未満および 500～1,000 byte の入力データの割合は両方式ともほぼ変わらない．そして静的スケジューリング方式では全入力データがメモリ使用量 1,500 byte 未満に収まる．一方，動的スケジューリング方式ではメモリ使用量が 2,000 byte 以上の入力データも全体の 26.8%を占め，メモリ使用量の平均値を上げている．

このような動的スケジューリング方式のメモリ使用量の増加は，自車と他車の走行状況を結合するオペレータ Join（図 11）が原因である．このオペレータ Join では，自車の情報と，特定の条件にマッチングした複数台の他車の情報を結合し，他車ごとに出力データを生成するため，1 度に多くの出力データがストリームキューに格納され，メモリ使用量が増加する．たとえば今回の評価では最大 30 個の出力データがストリームキューへ格納されることを確認した．一方，静的スケジューリング方式では上記のストリームキューは除去され，オペレータ Join の出力データは 1 個ずつ単一データメモリに格納されるため，メモリ使用量が増加しない．

なおオペレータ Join（図 11 ID: 2，4，5，16）およびオペレータ Aggregate（図 11 ID: 12）のウィンドウは静的データ領域に割り当てられ，静的スケジューリング方式，動的スケジューリング方式はともに合計で 36,896 byte となった．

表 7　運転支援システムにおける動的スケジューリングの処理レイテンシ
Table 7　Processing latency in the drive assist system by the dynamic scheduling.

	接近防止	追随走行	全アプリ
平均レイテンシ	809 μ 秒	1,040 μ 秒	925 μ 秒
最大レイテンシ	811 μ 秒	1,055 μ 秒	933 μ 秒
最小レイテンシ	792 μ 秒	1,019 μ 秒	906 μ 秒

表 8　運転支援システムにおける静的スケジューリングの処理レイテンシ
Table 8　Processing latency in the drive assist system by the static scheduling.

	接近防止	追随走行	全アプリ
平均レイテンシ	514 μ 秒	683 μ 秒	599 μ 秒
最大レイテンシ	518 μ 秒	689 μ 秒	604 μ 秒
最小レイテンシ	498 μ 秒	662 μ 秒	580 μ 秒

5.3　処理レイテンシ

運転支援システムおよび，カーナビゲーションにおいて，1 つの入力データの処理に要する時間すなわち処理レイテンシを評価する．

5.3.1　運転支援システム

先行車両認識クエリの処理レイテンシを**表 7**，**表 8** に示す．静的スケジューリング方式では動的スケジューリング方式と比較し，全アプリケーションの平均レイテンシが 35.2%，最大レイテンシが 35.6%削減した．このような処理レイテンシの削減は，静的スケジューリング方式によりスケジューラがオペレータを呼び出す回数（以下，オペレータ呼び出し回数）が削減されたためと考えられる．1 つの入力データの処理におけるオペレータ呼び出し回数は，たとえば追随走行アプリの場合，動的スケジューリング方式では，実行するオペレータと同数の 19 回である．一方，静的スケジューリング方式では同期実行グループ数とオペレータ切替え箇所の数の合計 3 回となる．

また運転支援システムのアプリケーションの動作周期はおおよそ 10～100 ms であるのに対し，静的スケジューリング方式および動的スケジューリング方式の処理レイテンシは，追随走行アプリの場合に最大 689 μ 秒，1,055 μ 秒であるため，動作周期に対し処理レイテンシが小さいことも確認できた．

表 9　カーナビゲーションにおける動的スケジューリングの処理レイテンシ

Table 9　Processing latency in the car navigation by the dynamic scheduling.

	前方車両	交差車両	右折時対向車両	全アプリ
平均レイテンシ	115 μ秒	138 μ秒	139 μ秒	125 μ秒
最大レイテンシ	311 μ秒	324 μ秒	318 μ秒	324 μ秒
最小レイテンシ	102 μ秒	120 μ秒	123 μ秒	102 μ秒

表 10　カーナビゲーションにおける静的スケジューリングの処理レイテンシ

Table 10　Processing latency in the car navigation by the static scheduling.

	前方車両	交差車両	右折時対向車両	全アプリ
平均レイテンシ	91.4 μ秒	107 μ秒	108 μ秒	98.4 μ秒
最大レイテンシ	272 μ秒	280 μ秒	271 μ秒	280 μ秒
最小レイテンシ	82.3 μ秒	93 μ秒	96.3 μ秒	82.3 μ秒

なお本評価ではスケジューラの呼び出し回数削減による処理レイテンシ削減の効果を確認するため，外部から入力データを 1 つずつ与え，クエリの処理を実行した．したがって後発の入力データに対し優先度が高いオペレータが先に実行されることがないため，接近防止アプリと追随走行アプリの優先度の違いは処理レイテンシに影響しなかった．優先度による処理レイテンシの影響に関する詳細評価は今後の課題とする．

5.3.2　カーナビゲーション

周辺車両認識クエリの処理レイテンシを**表 9**，**表 10** に示す．静的スケジューリング方式は動的スケジューリングと比較し，全アプリで平均レイテンシが 21.3%，最大レイテンシが 13.6%削減された．処理レイテンシの削減率が運転支援システムよりも小さいのは，静的および動的スケジューリング方式のオペレータ呼び出し回数の差が小さいことが原因と考えられる．1 つの入力データの処理におけるオペレータ呼び出し回数は，たとえば右折時対向車両表示の場合には，動的スケジューリング方式では，実行するオペレータと同数の 10 回である．一方，静的スケジューリング方式では同期実行グループ数と同数の 3 回となる．

また車々間通信データは最大で約 90 台から 100 ms 間隔で受信することを想定している [14]．そこで静的スケジューリング方式，動的スケジューリング方式において 90 台の他車データの合計処理時間を測定した．その結果，静的スケジューリング方式は平均 11,800 us，最大 13,100 us，動的スケジューリング方式は平均 13,900 μ秒，最大 14,600 μ秒となり，車々間通信の周期に対し処理時間が小さい．

なお運転支援システムと同様に外部から入力データを 1 つずつ与え，クエリの処理を実行したため，後発の入力データを優先度が高いオペレータにより先に実行することはなく，実行処理レイテンシに影響しなかった．優先度による処理レイテンシの影響に関する詳細評価は今後の課題とする．

5.4　評価のまとめ

静的スケジューリング方式により，運転支援システムで固定長のストリームキューを減らすことにより，静的データ領域のメモリ使用量を削減した．一方，カーナビゲーションにおいても可変長のストリームキューを減らすことで，ヒープ領域のメモリ使用量を削減した．また運転支援システムおよびカーナビゲーションにおいて，スケジューラのオペレータ呼び出しを減らすことにより処理レイテンシを削減した．さらに eDSMS を車載組込みシステムの RAM に搭載可能であること，車載組込みシステムの処理レイテンシの要件を満たすことを確認した．優先度による処理レイテンシの影響の評価や，実際の車載アプリケーションを用いた環境など様々な車載組込みシステムで評価することが今後の課題となる．

6.　関連研究

Carney らの方式 [19] ではオペレータを superbox 単位でまとめてスケジューリングすることによりスケジューラの処理コストを削減する．Carney らの方式では複数のオペレータを superbox と定義し，スケジューラはまず superbox 単位でスケジューリングを行い，実行する superbox の決定後，superbox に含まれるオペレータに対しスケジューリングを行う．しかし一般的な動的スケジューリング方式と同様に各オペレータはスケジューラにより呼び出し，またオペレータ間のストリームキューも必要となる．一方，静的スケジューリング方式では，同期実行グループ内で各オペレータはスケジューラにより呼び出されず，またオペレータ間のストリームキューも不要となる．

汎用システム向けの DSMS において処理レイテンシを削減する方式として，XStream の Depth-First スケジューリング方式 [20] があげられる．XStream ではオペレータの優先度は考慮せず，実行クエリのオペレータの接続関係のみに従ってオペレータを呼び出すため，動的スケジューリング方式よりもスケジューラの処理レイテンシは小さい．しかしスケジューラによるオペレータの呼び出し回数は変わらず，またオペレータ間のストリームキューも必要となる．

今木らの方式 [21] も，汎用システム向けの DSMS において処理レイテンシを削減する方式である．今木らの方式では複数のオペレータを 1 つのグループとし，そのグループ内では XStream と同様にオペレータの優先度は考慮せず，実行クエリのオペレータの接続関係に従ってスケジューラが各オペレータを呼び出す．またそのグループ内では時刻順に入力データを与えることにより，複数ストリームを入力とするオペレータを実行する際の入力データの時刻順整

列を不要とし，処理レイテンシを削減する．しかし静的スケジューリング方式と異なり，オペレータの動的変更に対応するために，各オペレータをスケジューラで呼び出す．また複数データを出力するオペレータではまとめてデータを出力するため，オペレータ間にストリームキューが必要となる．

組込みシステム向けのDSMSとして，Muellerの方式[22]，Gigascope[23]があげられる．Muellerの方式は無線センサノード向けのDSMSである．クエリを動的に登録し，登録されたクエリから独自の中間言語に変換し仮想マシン上でその中間言語を実行する．一方，Gigascopeは基地局などにあるネットワーク機器向けのDSMSであり，eDSMSと同様に静的に登録されたクエリからC/C++のソースコードを生成する．いずれの方式でも静的スケジューリング方式とは異なり，ストリームキューや，スケジューラのオペレータ呼び出しの削減については検討されていない．

7. おわりに

本稿では，車載組込みシステムにおけるメモリ使用量および処理レイテンシ削減を目的とし，eDSMSの静的スケジューリング方式を提案した．静的スケジューリング方式では，静的に決定したオペレータの優先度に従って複数のオペレータをグループとして抽出し，そのグループ内ではオペレータ間のストリームキューを除去し，ランタイムでは各オペレータを同期して実行する．また静的に決定した実行クエリから，実行するオペレータが動的に切り替わらない箇所を静的に検出し，ランタイムでは検出箇所でスケジューラを介さずに各オペレータが次に実行するオペレータを呼び出す．そして静的スケジューリング方式を実装し，運転支援システム，カーナビゲーションを想定した組込み環境で評価した．その結果，従来方式である動的スケジューリング方式と比較しメモリ使用量，処理レイテンシを削減し優位性を示した．今後の課題としては，実際の車載アプリケーションを用いた環境など，様々な車載組込みシステムにおける評価があげられる．

謝辞　本研究の一部は科研費（22240003）の助成を受けている．

参考文献

[1] 浅沼信吉，加世山秀樹：安全運転支援のための車両予知・予測技術のとりまく状況，国際交通安全学会誌，pp.56–61 (2006).
[2] Jones, W.D.: Keeping Cars from Crashing, *IEEE Spectrum*, Vol.38, No.9, pp.40–45 (2001).
[3] 藤田浩一，宇佐見祐之，山田幸則，所　節夫：衝突危険性のセンシング技術，自動車技術，Vol.61, No.2, pp.62–67 (2007).
[4] 西垣戸貴臣，大塚裕史，坂本博史，大辻信也：予防安全の高度化を実現するセンサーフュージョン技術，日立評論，pp.1–4 (2007).
[5] 佐藤健哉：自動車走行環境認識のためのセンサデータ処理機構，電子情報通信学会技術研究報告，pp.51–56 (2010).
[6] Motwani, R., Widom, J., Arasu, A., Babcock, B., Babu, S., Datar, M., Manku, G., Olston, C., Rosenstein, J. and Varma, R.: Query Processing, Resource Management, and Approximation in a Data Stream Management System, *Conference on Innovative Data Systems Research* (*CIDR*) (2003).
[7] Abadi, D.J., Carney, D., Cetintemel, U., Cherniack, M., Convey, C., Lee, S., Stonebraker, M., Tatbul, N. and Zdonik, S.: Aurora: A new model and architecture for data stream management, *The VLDB Journal*, pp.120–139 (2003).
[8] 渡辺陽介，北川博之：連続的問合せに対する複数問合せ最適化手法，電子情報通信学会論文誌，Vol.J87-D-I, No.10, pp.873–886 (2004).
[9] Babcock, B., Babux, S., Datar, M. and Motwani, R.: Chain: Operator Scheduling for Memory Minimization in Data Stream Systems, *ACM International Conference on Management of Data* (*SIGMOD*), pp.253–264 (2003).
[10] Chandrasekaran, S., Cooper, O., Deshpande, A., Franklin, M.J., Hellerstein, J.M., Hong, W., Krishnamurthy, S., Madden, S., Raman, V., Reiss, F. and Shah, M.: TelegraphCQ: Continuous Dataflow Processing for an Uncertain World, *CIDR* (2003).
[11] 山田真大，鎌田浩典，手嶋茂晴，高田広章，佐藤健哉：データストリーム管理機構を利用した車載データ統合モデルの提案と評価，自動車技術会論文集，Vol.41, No.2, pp.419–424 (2010).
[12] Borealis Distributed Stream Processing Engine, available from ⟨http://www.cs.brown.edu/research/borealis/public/⟩.
[13] ルネサス：マイコン，入手先 ⟨http://japan.renesas.com/products/mpumcu/index.jsp⟩.
[14] 5.8 GHzを用いた車々間通信システムの実験用ガイドライン ITS FORUM RC-005 1.0版，入手先 ⟨http://www.itsforum.gr.jp/Public/J7Database/p32/ITSFORUMRC005V1_0.pdf⟩.
[15] 勝沼　聡，杉本明加，山口晃広，山田真大，金　榮柱，本田晋也，佐藤健哉，高田広章：車載システム向けストリームデータ処理の提案と評価，EMB23, pp.1–8 (2011).
[16] 山口晃広，山田真大，勝沼　聡，本田晋也，佐藤健哉，高田広章：車載DSMSにおける静的クエリ最適化，WebDB Forum, 1G-2-4, pp.1–9 (2011).
[17] Altera: Nios II 3C25 Microprocessor with LCD Controller Data Sheet, available from ⟨http://www.altera.co.jp/literature/ds/ds_nios2_3c25_lcd.pdf⟩.
[18] ZMP: RoboCarR 1/10 / ZMP RC-Z, available from ⟨http://www.zmp.co.jp/e-nuvo/jp/robocar-110.html⟩.
[19] Carney, D., Cetintemel, U., Rasin, A., Zdonik, S., Cherniack, M. and Stonebraker, M.: Operator Scheduling in a Data Stream Manager, *VLDB*, pp.838–849 (2003).
[20] Girod, L., Mei, Y., Rost, S., Thiagarajan, A., Balakrishnan, H. and Madden, S.: XStream: A Signal-Oriented Data Stream Management System, *ICDE*, pp.1180–1189 (2008).
[21] 今木常之，樫山俊彦，西澤　格：遅延演算を利用したストリームデータの再帰処理方法，日本データベース学会論文誌，Vol.8, No.4, pp.7–12 (2010).
[22] Mueller, R.: Data Stream Processing on Embedded Devices, *Degree of Doctor of Sciences*, ETH Zurich, pp.1–292 (2010).
[23] Cranor, C. and Shkapenyuk, V.: Gigascope: A Stream Database for Network Applications, *SIGMOD*, pp.647–651 (2003).

勝沼 聡　（正会員）

（株）日立製作所中央研究所企画員．2008 年東京大学大学院情報理工学系研究科電子情報学専攻修了．同年（株）日立製作所中央研究所入社．2011～2012 年名古屋大学大学院情報科学研究科附属組込みシステム研究センター研究員．データストリーム管理システムの研究に従事．

本田 晋也　（正会員）

2002 年豊橋技術科学大学大学院情報工学専攻修士課程修了．2005 年同大学院電子・情報工学専攻博士課程修了．2005 年名古屋大学情報連携基盤センター名古屋大学組込みソフトウェア技術者人材養成プログラム産学官連携研究員．2006 年名古屋大学大学院情報科学研究科附属組込みシステム研究センター助教．現在，同准教授．リアルタイム OS，ソフトウェア・ハードウェアコデザインの研究に従事．博士（工学）．2002 年度情報処理学会論文賞受賞．ACM，電子情報通信学会，日本ソフトウェア科学会各会員．

佐藤 健哉　（正会員）

同志社大学大学院理工学研究科情報工学専攻教授．1986 年大阪大学大学院工学研究科電子工学専攻修士課程修了．同年住友電気工業情報電子研究所入社．1991～1994 年スタンフォード大学計算機科学科客員研究員．2000 年奈良先端科学技術大学院大学情報科学研究科博士後期課程修了．米国 AMI-C, Inc. チーフテクノロジストを経て，2004 年より現職．同志社大学モビリティ研究センター長および名古屋大学大学院情報科学研究科附属組込みシステム研究センター特任教授兼務．博士（工学）．IEEE-CS，ACM，電子情報通信学会，自動車技術会各会員．

高田 広章　（正会員）

名古屋大学大学院情報科学研究科情報システム学専攻教授．1988 年東京大学大学院理学系研究科情報科学専攻修士課程修了．同専攻助手，豊橋技術科学大学情報工学系助教授等を経て，2003 年より現職．2006 年より大学院情報科学研究科附属組込みシステム研究センター長を兼務．リアルタイム OS，リアルタイムスケジューリング理論，組込みシステム開発技術等の研究に従事．オープンソースの ITRON 仕様 OS 等を開発する TOPPERS プロジェクトを主宰．博士（理学）．IEEE，ACM，電子情報通信学会，日本ソフトウェア科学会，自動車技術会各会員．

（担当編集委員　堀井 洋）