RD-002

車載システム向けデータストリーム管理システムにおけるクエリ自動構築手法
Automatic Query Construction Method in DSMS for Vehicle System

山口 晃広 † 本田 晋也 † 佐藤 健哉 ‡ 高田 広章 †
Akihiro Yamaguchi Shinya Honda Kenya Sato Hiroaki Takada

1. はじめに

近年，車載システムには，プリクラッシュセーフティといった走行状況や周辺状況を認識し，ドライバへの警告や自動制御により運転の支援を行うアプリケーション（安全運転支援システム）が搭載されるようになってきている．安全運転支援システムを実現する為に，車載システムには様々なセンサが搭載され，車車間通信や路車間通信等も加わり，多種多様な情報源が存在する．またその膨大な組み合わせにより計算できる情報やその精度も異なる．これらの組み合わせや計算資源といった物理構造は，自動車の種類や年式（車種）によって様々に異なる．現状の車載システムの開発では，車種ごとにデータ処理を設計/実装する必要があり，開発工数が増大している．

我々は，車載システムのデータ処理部分をアプリケーションから切り離し，データストリーム管理システム（DSMS）で管理することで，センサデータのような更新頻度の高いデータを低遅延で処理しつつ，アプリケーション開発を容易にするアーキテクチャを研究開発している [3][4][5]．しかし現状のアーキテクチャでは物理構造に依存し，車種ごとにクエリを作り直す必要がある．また，センサ周りの制御処理といったアプリケーション開発者が関心の無い低レベルなデータ処理まで意識してクエリを作る必要がある．

一方，サービス指向アーキテクチャ（SOA）が，Web サービス分野を中心に普及している．Web サービスでは，Web 上に散在するサービスをシステムが柔軟に組み合わせ，ユーザの要求を満たす一連のアプリケーションを実現する．

車載システムのデータには，自車両の位置情報や先行車認識情報といった様々なアプリケーションから利用される汎用的なデータ（車載データ）が存在する [8][3]．提案手法では，車載データや情報源を入力とし，他の車載データを出力するデータ処理を Data Stream Component（DSC）として定義し，これを SOA におけるサービスの単位とする．そして，図 1 のように，DSC をオペレータのように指定してクエリを記述する．DSC の接続は省略することが可能で，省略された場合，予め定義された DSC を物理構造に適合するよう接続していく．なお，ANONYMOUS な DSC（ADSC）を指定することで，具体的な DSC の選択をシステムに任せることもできる．その際，クエリの精度といった QoS を指定することで，物理構造に加え QoS にも適合するようにクエリを補完/構築する．このように，サービスの接続/選択をシステムに任せるアイデアは，SOA と類似する．これにより，安全運転支援システムの開発に以下のようなメリットをもたらす．

- 車種の違い依らないデータ処理の開発
  物理構造に依存せずクエリを記述できるアーキテクチャを実現し，車種の違いに依らないデータ処理の開発を実現する．
- 低レベルなデータ処理を省略
  センサ周りの制御処理など，アプリケーション開発者が関心の無い低レベルなデータ処理を物理構造や QoS に適合するよう，システムが構築する．

更に，DSC を接続/選択して補完されたクエリは，最終的には DSC を含まない通常のクエリとなり，既存の DSMS で利用可能となる．その為，動的なメモリ割当や遅延時間のオーバーヘッドが許されないような組込み環境にも適用できる．また，SOA の類似手法により生成されるクエリよりもクエリ最適化を効率的に適用できることで，安全運転支援システムにおいて重要な最大メモリ使用量や遅延時間をより削減できる．

以降では，2 章で車載システム向け DSMS のクエリに SOA のアイデアを適用する経緯を説明し，3 章で提案手法を説明する．その後 4 章でその評価を述べ，最後に 5 章で全体の考察と今後の課題を述べる．

2. 車載システム向け DSMS への SOA の適用

本章では，車載システム向け DSMS のクエリに SOA のアイデアを適用するに至った過程を説明する．

2.1. 車載システムへの DSMS の適用

車載システムには，様々なセンサが搭載され，車車間/路車間通信等も加わり，様々な情報源が存在する．そして，それらの情報源を結合や加工して得られる様々な車載データを運転支援システムといった多くのアプリケーションが利用している．我々は，車載のデータ処理をアプリケーションから切り離し，統合管理する研究開発を行っている [2][4][5]．

車載システムでは，センサデータのように高頻度で流れるデータを継続して処理する必要があり，車車間通信によりデータを受信する場合等，データレートの増減も起こる為，車載データの統合管理には DSMS が適している．

これまでに DSMS は，STREAM[11] や Borealis[10] 等が開発されてきたが，主に汎用システムを対象としていた．多くの車載システムでは，動的なメモリ確保が困難であり，メモリ使用量や CPU といった計算資源が稀少な組込みシステムである．また，車載システムでは，設計時にどのようなクエリが必要か確定しており，システムを安定して稼動させる必要もあることから，我々は，車載システムに適した組込みシステム向け DSMS（eDSMS）を開発している [5][4]．eDSMS で

†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
‡同志社大学モビリティ研究センター

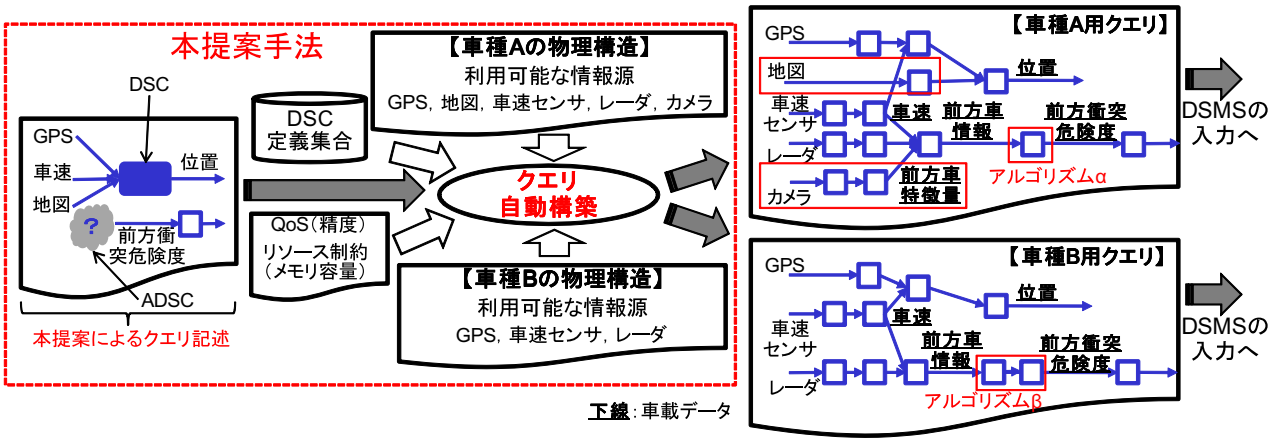

図 1: 提案手法によるクエリ記述

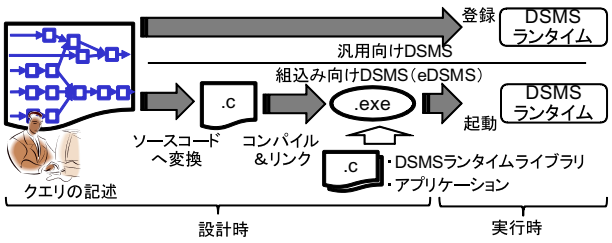

図 2: 汎用向け DSMS と eDSMS との違い

は，図 2 のように，クエリを予めソースコードに変換して DSMS ランタイムライブラリとコンパイル，リンクする特徴を持つ．

2.2. 現状の DSMS の問題点

自動車の種類や年式といった車種が異なると，搭載されるセンサの数や種類が変わるとともに，カーナビゲーションや通信機器の搭載の有無によって地図情報や車車間通信による情報の利用可否も異なる．また，車種により，メモリ容量や CPU 等も異なる．このように，車種が異なれば，情報源の組み合わせや計算資源といった物理構造が一般には異なる．

我々は，車載データを抽象度に応じて DSMS で階層的に管理し，複数のアプリケーションから効率的に車載データを利用する先行研究を行ったが，車種に依存しないクエリの設計/実装は，実現できていなかった [3]．例えば，車種が異なると，車載データの計算方法は入力となるストリームの種類やその計算アルゴリズムが一般に複数存在する為，クエリの候補は多数存在するが，その中からどのように適切なクエリを選択/構築するか検討されておらず，車種毎に階層的なデータの管理方法を見直す必要があった．

以上のように，従来の DSMS では，車載データのクエリを設計/実装する場合，物理構造に依存してしまい，車種ごとにクエリを設計/実装する必要がある．自動車の種類は様々であり，バージョンアップが起こるたびに，クエリの設計/実装を修正することは，車載システムの開発工数の増大を招く．また，従来の DSMS では，車載データ処理のクエリの中で本来意識する必要の無いセンサ周りの制御処理等，アプリケーション開発者にとって低レベルなデータ処理まで意識して開発する必要がある．このような低レベルなデータ処理については，物理構造や QoS をもとに，システムにその処理の構築を任せる方が好ましい．

2.3.SOA による解決

主に，ネットワーク上に散在するソフトウェアの機能をサービスとして定義し，それらを柔軟に連携させて大規模なアプリケーションを効率的に開発するサービス指向アーキテクチャ（SOA）が Web サービス分野を中心に研究/展開されている．SOA では，サービスの結合度を緩め，多種多様なサービスの選択や接続をシステムに任せるというアイデアがある．

提案手法では，2.2 で上げた問題を解決する為に，SOA のアイデアを DSMS のクエリの構築に適用する．しかし，従来の SOA では，Web サービスといったネットワークを跨いだ粒度の大きいプロセスを豊富な計算資源を使い連携する状況で主に検討されてきた [7]．その為に，動的なメモリ割当が不可能であり，数百 μ 秒程度のデータ処理が対象となる等の厳しいリソース制約がある車載システムへ現状の SOA のアーキテクチャを適用することは難しい．また，DSMS のクエリの構築に SOA を適用した研究事例も無く，車載システムの要件に合うクエリを構築する仕組みが必要となる．

なお，SOA では，QoS 等のユーザの要求に合わせてサービスの結合/選択を行う方法が知られている．提案手法では，車載システムの要件に合うリソース制約と QoS とを指定することで，その条件に合致するクエリを構築する．

2.4. 車載システムにおける QoS とリソース制約

車載システムのような組込み環境では，メモリ容量が稀少であるが，DSMS ではデータ処理をストリームで繋ぎ全てメモリ上で処理することからメモリ使用量が大きくなる．車載システムのような組込み環境では，利用可能なメモリ容量の上限が定められている場合が多く，なるべく最大メモリ使用量を減らし，その制約を満たすことが重要な課題となる．次に，安全運転支

援システムでは，様々なセンサデータや場合によっては車車間通信や地図情報を利用するケースが考えられるが，利用する情報源の組み合わせや計算アルゴリズムによって結果の精度が異なる．このようなクエリでは，精度は一定以上に保つ必要がある．また，遅延時間もなるべく削減するとともに予め定めた制約を満たすよう一定値以下に抑えることも重要である．

以上の考察から，以下の 3 つの指標を考慮する．

（1）クエリ全体の最大メモリ使用量
（2）クエリの各出力に対する精度
（3）クエリの各出力に対する遅延時間

eDSMS では，クエリは設計時に確定しソースコードに変換される為，設計時にこれらの指標を静的に見積もる必要がある．指標 3 の遅延時間については，各オペレータの選択率や単位タプルあたりの処理時間といった実行時の統計情報を使って見積もる方法が提案されているが [6]，これらの統計情報を設計時に知ることは難しく指標 3 を直接見積もることは難しい．そこで，指標 1 と 2 の要求を満たすクエリを構築し，指標 3 については実際にクエリを動作させてその性能が要求を満たすか確かめる．

2.5. サービスの単位

欧州の標準化組織である欧州電気通信標準化機構（ETSI）から提案されている車載データ管理の仕様である Local Dynamic Map（LDM）[8] では，自車両や周辺車両の位置や車速，地図とセンサデータから計算される情報等，汎用的なデータ（車載データ）を RDBMS のテーブルで管理し，その名前とスキーマを仕様として公開している．我々は，これらの車載データを RDBMS ではなく，DSMS で管理する方法を研究開発している．

提案手法では，先行研究 [3] のように，車載データや情報源から，別の車載データが生成されることに着目し，そのデータ処理を SOA におけるサービスと定義する．そして，LDM が車載データの仕様のみを管理し，それらの計算方法や取得方法については管理しないことにも着目し，具体的なサービスの接続/選択をシステムに任せることで，提案手法を実現する．

2.6. 提案手法の要件

以上の考察から，提案手法が解決する要件としては，以下があげられる．

（a）クエリの記述が情報源に依存しない
（b）低レベルなデータ処理を省略しクエリ記述できる
（c）構築したクエリがリソース制約や QoS を満たす
（d）組込み環境に適用できる

(a)〜(c) が従来の DSMS のクエリに欠けている点であり，SOA のアイデアを適用することで解決する．また，(d) が既存の SOA の問題点であり，提案手法ではサービス間の接続にオーバーヘッドが無く，生成されたクエリを eDSMS で実行することにより解決する．

3. 提案手法

本提案手法では，車載データの具体的な計算方法や取得方法を SOA のサービスとして定義し，システムにサービスの選択/結合を任せる．なお，サービスの選択/結合の基準は，QoS や物理構造に適合するよう実施する．サービスの選択/結合後は，DSMS の通常のクエリに変換し，既存のクエリ最適化によりデータ処理を効率化する．

表 1: 対象とするオペレータ

オペレータ	説明
Filter	入力に対して，条件にマッチしたデータを出力
Map	入力に対して，演算を実行した結果を出力
Join	2 つの入力の過去一定量のデータをウインドウに保持し，条件に合致したデータを結合し出力
Aggregate	入力データを一定量ウインドウに保持し，保持したデータに集計関数を実行した結果を出力

まず，3.1 節で提案手法を説明する為の前準備を行う．次に，3.2 節で提案手法のサービスを定義し，3.3 節でアプリケーション開発者（ユーザ）が提案手法を使ってどのようにクエリを記述するかを説明する．そして，3.4 節で提案手法の全体の処理の流れを説明し，その中で別途説明が必要な処理を 3.5 節〜3.7 節で述べる．

3.1. 前準備

提案手法の実装/評価では，DSMS として，車載システムに適した eDSMS を使用する．eDSMS では，クエリの記述は Borealis のようなデータフローで表し XML 形式で記述される．本評価で使用するオペレータは，表 1 で紹介する Filter, Map, Join, Aggregate であり，Borealis のオペレータのサブセットである [12]．

3.2. サービスの定義

2.5 で説明したサービスの単位を提案手法では，Data Stream Component（DSC）と呼ぶ．DSC では，入力ストリームが車載データや情報源で，出力ストリームが車載データとなり，その入力から出力へのデータ処理を DSMS のクエリとして定義する．

同じ車載データを出力する DSC であっても，その定義は複数存在する場合が多い．例えば，図 3(左) のように，自車両の位置情報を出力する DSC には，GPS のみを入力とし計算する場合や，GPS と車速を入力としセンサフュージョンにより計算した結果を出力する場合や，更に地図情報を使ってマップマッチングする場合等の様々な定義が存在することが考えられる．また，図 3(右) のように，入出力となる車載データが同じ DSC であっても，計算アルゴリズムが異なることも考えられ，様々な定義が存在する可能性がある．提案手法では，システムが同じ車載データを出力する DSC を図 3 のようにグルーピングしておき，複数定義が存在する DSC の選択をシステムに任せることもできる．これを Anonymous な DSC（ADSC）と呼ぶ．

DSC の定義と ADSC は，ライブラリとしてアプリケーション開発者に公開し，アプリケーション開発者はそれらを指定して利用する．利用したい車載データが LDM に存在すればそれを利用し，存在しなければ名前とスキーマにより新たな車載データとして同様の管理方法で定義し，アプリケーション開発者に公開する．

3.3. クエリの記述

ユーザは，DSMS の通常のオペレータと DSC，ADSC を利用し，それらの接続関係をユーザが決めたい場合はストリームで繋ぐことで，クエリを記述する．

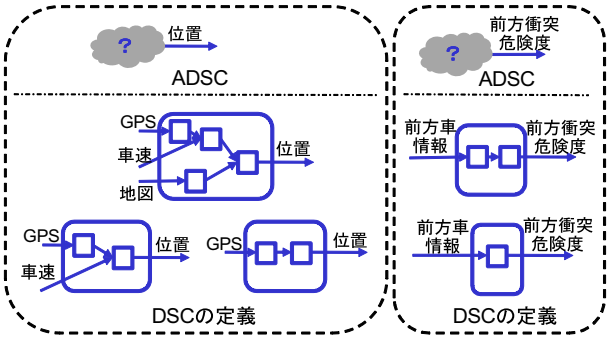

図 3: DSC の定義とそのグルーピング，ADSC

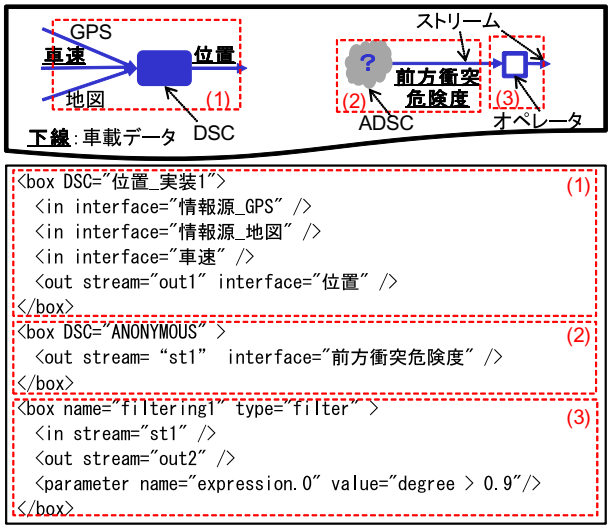

図 4: ユーザによるクエリの記述

DSC を利用する場合は，図 4(1) のように，その名前を DSC 属性に記述し，入出力ストリームの車載データの名前を interface 属性に各々記述する．ADSC を利用する場合は，図 4(2) のように，DSC 属性を “ANONYMOUS” とし，出力ストリームの interface 属性に車載データの名前を記述する．

DSC を利用する際は，interface 属性が存在することを除いて，図 4(3) のように通常のオペレータを利用する場合と同様のインターフェースをユーザに提供する．DSC，ADSC を除いたクエリの記述は，Borealis と同様である [12]．

従来の DSMS では，全てのクエリをユーザが記述する必要があったが，本提案では，以上のようにユーザが車載データの処理を省略し，システムにそのデータ処理の構築を任せることができる．図 4 の例では，ユーザは，自車位置を計算する際に必要となる車速の取得方法や，前方衝突危険度の具体的な計算方法には，関心が無く，それらの記述を省略している．

3.4. 処理の流れ

提案手法の処理の流れを図 5 に示す．まず，アプリケーション開発者は 3.3 で説明したようにクエリを記述する．これが提案手法の入力となる．

（1）クエリ補完

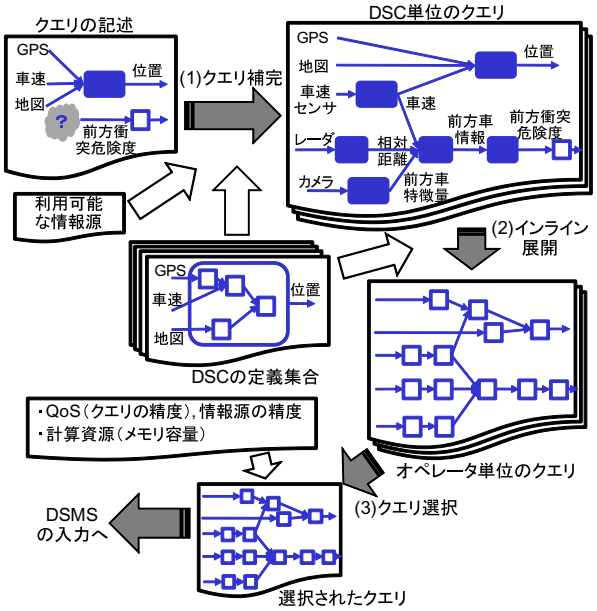

図 5: 提案手法の処理の流れ

アプリケーション開発者が記述したクエリを入力として，利用されている DSC の中で，入力ストリームが接続されていない場合，DSC の定義集合を参照して DSC を接続していく．なお，情報源と接続されているかどうかは，利用可能な情報源を参照して判断する．図 3 のように同じグループに属する DSC が複数がある場合は接続可能なパターンが複数表れることがあり，全ての接続可能なパターンに対してクエリの候補を構築していく．また，ADSC が利用されている場合は，置き換え可能な DSC に置換し，同様に処理する．ここで，構築されるクエリの候補は一般に複数存在する．詳しいアルゴリズムは 3.5 で説明する．

（2）インライン展開

DSC 単位のクエリの中で利用している DSC をそのデータ処理を表すクエリでインライン展開する．そして，インライン展開された DSC の入出力ストリームを DSC 単位のクエリに基づき接続する．インライン展開後のクエリは，通常の DSMS で利用されるクエリと同じものになる．この処理は，各候補に対して実施するため，オペレータ単位のクエリも同数存在する．

（3）クエリの選択

クエリの各候補に対して，物理構造と QoS に適合したクエリをシステムが選択する．2.4 で説明したように，車載システムで重要な指標であるクエリの精度と最大メモリ使用量を見積もり，その値が QoS やリソース制約を満たすクエリの候補を選択する．複数のクエリが候補として存在する場合は，精度の降順や最大メモリ使用量の昇順，またはそれらの重み付けでソートする．3.6 と 3.7 で，クエリの精度とメモリ使用量の見積もり方を説明する．なお，これらはクエリを絞り込む為の大凡の見積もりであり，厳密なものではない．また，本稿はアーキテクチャ全体の提案であり，見積もりの詳細な議論については別報する．

3.5. クエリ補完

クエリの記述を入力として，クエリの中で省略された部分をシステムが補完し構築する方法を説明する．

まず，クエリの中に ADSC を含まない場合で説明する．基本的なアルゴリズムの方針は，補完の必要なストリームに対して，DSC の定義集合から接続可能な出力ストリームを持つ DSC を接続していく．図 3 のように同じ車載データを出力する DSC は一般に複数存在する．複数存在する場合は，接続可能な全てのパターンに対して DSC の接続を実施する．これを接続が必要なストリームがなくなるまで繰り返す．その際，利用可能な情報源を参照する．接続が必要なストリームが存在し，接続可能な DSC が存在しない場合は，構築不可能なクエリとして候補から除外される．接続が必要なストリームとは，情報源とも，オペレータや DSC の出力ストリームとも接続していない DSC の入力ストリームである．

3.2 で説明したように，本提案では車載データはその名前により定義し管理する．つまり，DSC が接続可能かどうかは，車載データの名前が単純に一致するかどうかで判断する．RDBMS のテーブル名と同様，車載データの名前が一致する場合は，そのスキーマも必ず一致する．そのため接続の際に，スキーマを変換する必要は無く，直接接続される．なお，クエリを出力側に辿った時に同じ DSC が見つかった場合は，無限ループを避ける為，その DSC は接続不可能とする．

ADSC を含む場合は，同じグループに属する DSC の定義に置き換えた全てのパターンを入力として，上述の手順を実行すれば良い．

以上のアルゴリズムにより情報源まで接続したクエリが生成されるが，マルチクエリの場合，これだけだと本来共有できるはずの DSC が重複して存在する場合がある．例えば図 5 の場合だと，DSC 単位のクエリに車速を出力する DSC が重複して二つ生成されてしまう．そこで，同一のストリームまたは情報源を入力とし，DSC の種類が同一であれば，それらを共有し重複を取り除く．

3.6. 精度の見積もり

物体認識等のアルゴリズムでは認識誤りが起こり，センサから得られる位置情報等にはノイズがのる為に，車載データには不確かさが存在する．本提案では，データの確からしさを精度という尺度で表し，最も確実な場合を 1 とし最も不確実な場合を 0 とする．精度は，計算アルゴリズムや入力データに加え様々な条件に依存する為，一般に機械的に見積もることは非常に難しい．

そこで，DSC の開発者が，開発時にその精度の見積もり式（精度計算式）も定義する．eDSMS では，図 5 の (4) クエリの選択は，クエリの設計時に行う為に，実行時の情報は利用できない．よって，精度計算式は入力ストリームの精度を変数とする関数として定義できるようにする．また，センサ等の情報源に対しても精度の値を予め調べ付加しておく．すると，クエリの精度は，図 5 の DSC 単位のクエリに対して，情報源から DSC を順に辿り各ストリームの精度を計算していくことで，最終的に計算できる．

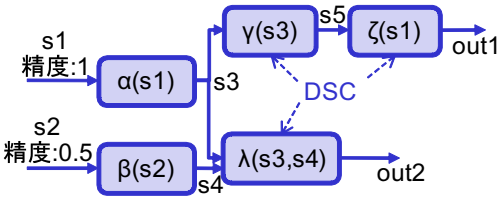

図 6: 精度の見積もり

例えば，図 6 の場合だと，$\alpha, \beta, \gamma, \lambda, \zeta$ が精度計算式で，出力ストリーム out1 と出力ストリーム out2 の精度は，各々$\zeta(\gamma(\alpha(1)))$，$\lambda(\alpha(1), \beta(0.5))$ と見積もれる．

3.7. 最大メモリ使用量の見積もり

eDSMS は，メモリの動的な割り当てが不可能な組込みシステムで動作させることが可能であり，その場合，クエリを構成するオペレータのウインドウとストリームのキューには，データが予め詰まった状態で，メモリが静的に確保される．これらがクエリの最大メモリ使用量の中で支配的であることから，オペレータのウインドウとストリームのキューのメモリサイズの合計として，クエリ全体の最大メモリ使用量を見積もる．

本評価で使用するクエリ最適化は，クエリ実行前に実施するものであり，最大メモリ使用量にも影響を与える為に，クエリ最適化を施した後のクエリから最大メモリ使用量を簡易的に見積もる．

4. 評価

先行車の減速による前方衝突，交差点での出会い頭による衝突，右折時の対向車との衝突は，車両相互事故において，統計上 37 %，32 %，16 %と多く，これらの衝突を検知し警告するクエリ（衝突検知クエリ）を開発することは安全運転支援において価値が高い [13]．本評価では，この 3 つの衝突検知クエリ（前方衝突検知クエリ [14]，交差衝突検知クエリ [15]，右折衝突検知クエリ）を DSMS のクエリとして実装して評価する．

類似手法として，センサネットワーク分野で，ユーザがセンサ周りの低レベルな処理を意識せずにセンサを利用した複数のアプリケーションを容易に開発できるよう，まとまりのある処理単位をサービスとして定義し，それらを自動的に接続する手法が提案されている [1]．類似手法で述べられているように，複数のクエリ（マルチクエリ）を登録すると重複したサービスが現われやすく，これらのサービスを共有化することが効率的なデータ処理に有効である．これは，車載システムでマルチクエリを登録する場合も同様である [4]．しかし，類似手法では共有化できるのはサービスが同一の場合のみである．一方，提案手法では，サービスの中身を複数のオペレータで構成し，インライン展開後はサービスの繋ぎ目が無くなり，通常の DSMS のクエリになる．そのため，DSMS の既存のクエリ最適化をオペレータ単位で効果的に適用でき，従来手法よりも更に処理を効率化できる．

4.1. 評価項目

本章では以下の 3 点を評価する．

表 2: 物理構造

種類	利用可能な情報源	メモリ容量
車載システム A	レーダ，車速センサ	200KB
車載システム B	自車情報，他車情報，道路車両結合情報	1000KB

表 3: サービス品質

精度	遅延時間
0.5 以上	100ms 以下

1. 実アプリケーションへの適用による確認
   2.6 であげた要件 (a)〜(d) を満たすことを前方衝突検知クエリを使って確認する．
2. データ処理の効率化における評価
   前方衝突検知クエリに交差衝突検知クエリと右折衝突検知クエリとを加え，マルチクエリのクエリ最適化が従来手法 [1] より効果的に働き，衝突検知クエリで重要となる最大メモリ使用量と遅延時間が効率的に削減されることを確認する．
3. 開発工数の評価
   提案手法により，従来の DSMS におけるクエリの開発よりも開発工数を削減できることをクエリを記述した XML の行数により示す．

4.2. 評価環境

表 2 に上げた 2 つの車載システムを想定して評価する．車載システム A では，レーダや車速センサが搭載されており，それらを情報源として直接利用できる．車載システム B では，センサから得られる情報を集約した自車情報，車車間通信で取得する他車情報，道路と車両 ID との結合情報（道路車両結合情報）[§] とを情報源として利用できる．全ての情報源の精度の値は 1 とする．表 2 の右列は，クエリ全体で使用できる大よそのメモリ容量である．

車載システム A に対して，Altera 社の組込みボード（Nios II 3C25）を使用し AUTOSAR 準拠のリアルタイム OS である TOPPERS/ATK2 上で評価する [16]. CPU は Nios II/f processor core であり，ヒープメモリは存在しない．車載システム B に対して，実車 1/10 のモデルカー（RoboCar）[17] 上で評価する．RoboCar の CPU は AMD Geode LX800 Processor 500MHz であり，メモリ容量は 512MByte であり．OS は Linux（Fedora 10）である．

QoS とリソース制約として，各衝突検知クエリの精度の下限と遅延時間の上限を表 3 のように全クエリ一律に設定する．

4.3. 評価で使用する DSC 定義集合

提案手法でクエリを構築する為には，DSC の定義集合を参照する．ここでは，紙面の都合上，本評価に関連する DSC の定義集合の基本情報と DSC の一部の定義に絞り，表 4 と図 7 に示す．

[§]LDM の alongroadelement テーブルに相当する車載データである．

表 4: DSC の基本情報

ID	出力ストリーム	入力ストリーム	精度計算式
1	前方衝突警告	前方衝突危険度	$0.9 \cdot \$1$
2	交差衝突警告	交差衝突危険度	$0.9 \cdot \$1$
3	右折衝突警告	右折衝突危険度	$0.9 \cdot \$1$
4	前方衝突危険度	先行車情報, 車速	$0.9 \cdot \$1$
5	交差衝突危険度	交差点車両情報	$0.9 \cdot \$1$
6	右折衝突危険度	交差点車両情報	$0.9 \cdot \$1$
7	先行車情報	自車情報, 道路車両結合情報, 他車情報	$(\$1 \cdot \$2 \cdot \$3)^{1/3}$
8	先行車情報	相対距離, 車速	$0.8\sqrt{\$1 \cdot \$2}$
9	交差点車両情報	自車情報, 道路車両結合情報, 他車情報	$(\$1 \cdot \$2 \cdot \$3)^{1/3}$
10	相対距離	レーダ	$0.7 \cdot \$1$
11	相対距離	レーダ	$0.8 \cdot \$1$
12	相対距離	レーダ, カメラ	$0.8\sqrt{\$1 \cdot \$2}$
13	車速センサ	車速	$0.8 \cdot \$1$

各 DSC の定義には ID を振り，表 4 と図 7 とで対応が取れている．入力ストリームは，複数存在する場合にカンマで区切っている．精度計算式では，入力ストリームの精度を $\$1, \$2, \$3, \ldots$ という記号で参照している．

4.4. 実アプリケーションへの適用による確認

表 2 の 2 つの車載システムに対して前方衝突検知クエリを構築することで，提案手法が要件を満たすことを確認する．ユーザが設計するクエリとしては前方衝突警告を出力する ADSC を一つ指定し，物理構造と QoS を表 2，表 3 のように指定し，提案手法を適用する．最終的に構築されるクエリは意味のある単位が崩れ理解しにくい為，提案手法の途中経過である DSC 単位のクエリを図 8 に示す．一般に複数のクエリの候補が選択される可能性があるが，今回の評価では，車載システム A と車載システム B で各々一通りに決まる．

車載システム A と車載システム B に対して，情報源の違いに応じたクエリが自動構築されており，(a) 情報源に依存しないクエリが設計できた．また，ユーザが設計したクエリには先行車情報や前方衝突危険度の計算といった低レベルなデータ処理が含まれていないことから，(b) 低レベルなデータ処理をクエリ記述から省略できている．次に，これらのクエリを eDSMS で動作させた結果を確認する．車載システム A に構築されたクエリの最大メモリ使用量は 95.67KB，精度は 0.55，平均遅延時間は 350us であり，車載システム B に構築されたクエリの最大メモリ使用量は 122.5KB，精度は 0.81，平均遅延時間は 112.574us であり，いずれも表 2 の物理構造の制約と表 3 の QoS の制約を満たしていることが分かり，(c) 物理構造や QoS に合わせたクエリが構築されている．また，Altera ボード上で QoS を満たすクエリを実行できることも分かり，(d) 組込み環境に提案手法が適用できる．

4.5. データ処理の効率化における評価

前方衝突検知クエリに交差衝突検知クエリと右折衝突検知クエリとを加え，マルチクエリを構築する場合に，SOA の類似手法 [1] より効果的にクエリを最適化

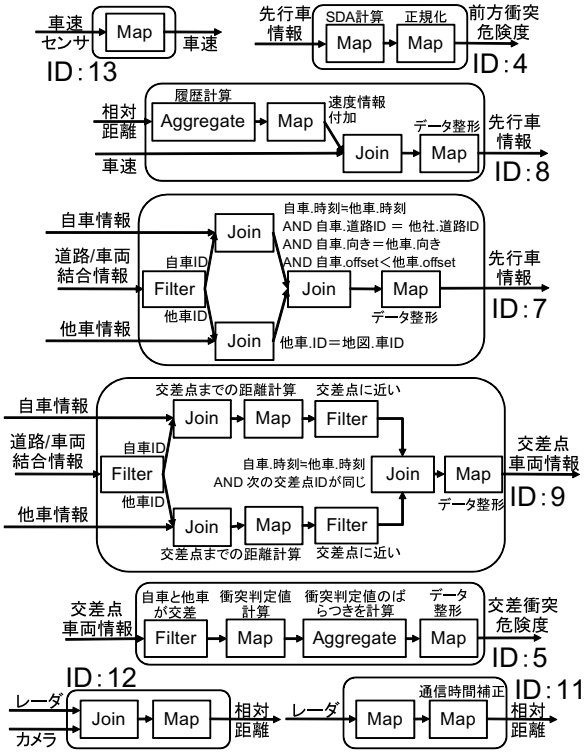

図 7: DSC の定義の例

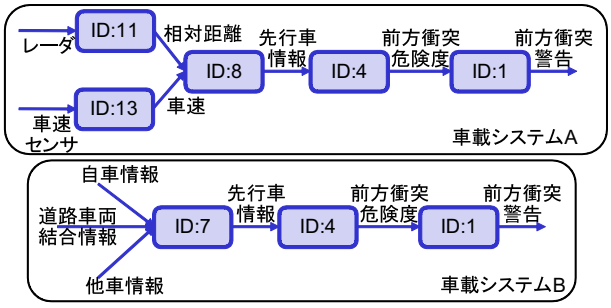

図 8: 構築された DSC 単位のクエリ

できることを示す．対象とする車載システムは，車車間通信が可能な表 2 の車載システム B 上で評価する．ユーザは，前方衝突検知，交差衝突検知，右折衝突検知をそれぞれ出力する ADSC を 3 つ記述し，QoS 等は表 3 のように指定し，提案手法を適用する．

提案手法では，eDSMS を使う為，最終的なクエリは C 言語のソースコードに変換され低遅延/高速に処理される．一方，類似手法 [1] では，論理プログラミングにより実装/評価している．ここでは，実装環境の違いに依らず，クエリ最適化の効果を比較評価する為に類似手法の評価でも eDSMS を使用する．類似手法では，サービスが完全に一致する場合のみ共有化が可能である為，サービス単位でクエリを最適化する．また，比較の為に最適化しなかった場合も評価する．なお，最適化しなかった場合の評価と類似手法の評価では，サービスの内部が複数のオペレータより構成される場合，なるべく単一のオペレータとなるように，各サービスに対して個別にクエリ最適化を行っておく．

表 5: クエリ最適化の評価

項目	最大メモリ使用量	平均遅延時間
最適化無	473.4KB	450.9us
類似手法	377.3KB	344.6us
提案手法	198.0 KB	252.5us

表 6: クエリの行数

項目	行数
従来 DSMS のクエリ記述	959 行
提案手法・アプリ開発者	26 行
提案手法・DSC 定義含む	725 行

提案手法と類似手法との比較結果を表 5 に示す．クエリ最適化により，提案手法では最大メモリ使用量が 58 %削減，平均遅延時間が 44 %削減され，類似手法では最大メモリ使用量が 20 %削減，平均遅延時間が 24 %削減され，提案手法でも類似手法でも最大メモリ使用量と遅延時間の削減に効果があることが分かる．特に，提案手法の方が類似手法よりも，最大メモリ使用量が 48 %削減，平均遅延時間が 27 %削減されており，更に効果が高いことが分かる．

これは，図 7 の ID:7 と ID:9 の DSC のように，サービスの単位である DSC が完全に一致しない場合でも DSC の一部を共有化できることや，DSC を構成するオペレータ間で結合ルールが適用され，多くのオペレータが結合した為と考えられる．つまり，提案手法ではオペレータの数が大幅に削減することで，クエリの最大メモリ使用量の中で支配的であるストリームの数も大幅に削減し，DSMS ランタイムのオペレータ呼び出し等のオーバーヘッドも減ることから，最大メモリ使用量と遅延時間も削減できたと考えられる．

4.6. 開発工数の評価

提案手法により，アプリケーション開発者のクエリの開発工数が削減されることは，以上より明らかであるが，ここでは，クエリを記述した XML の行数により定量的に比較する．車載システム A に前方衝突検知クエリを構築し，車載システム B に前方衝突検知クエリと交差衝突検知クエリと右折衝突検知クエリのマルチクエリを構築する場合で比較した結果が表 6 である．物理構造や QoS は，これまで同様に表 2，表 3 とする．

従来の DSMS でアプリケーション開発者がクエリを記述する場合（従来 DSMS のクエリ記述）では，各物理構造と QoS に合わせて，全てのパターンに対してクエリを作り込んだ行数を測定した．次に，提案手法を使いクエリを自動構築する場合（提案手法・アプリ開発者）のアプリケーション開発者のクエリの行数を測定した．最後に，提案手法・アプリ開発者の行数に，実際に使用した DSC を定義する行数を加えた結果（提案手法・DSC 定義含む）も測定した．これは，従来 DSMS のクエリ記述では，DSC を定義する必要が無く，その分を考慮した測定である．

提案手法により，アプリケーション開発者がクエリを記述する行数は，従来の DSMS でクエリを作り込む

より，97 %削減された．また，実際に使用した DSC 定義集合に関する行数を加えても 24 %削減できることが分かった．

アプリケーション開発者が記述する行数が劇的に減少する理由は，提案手法では，基本的に出力ストリームを 3 つ記述しただけで済み，DSC の接続関係や実際の定義は，一切記述していない為である．また，実際に使用する DSC の定義集合の行数を加えても行数が削減される理由は，図 7 の ID:9 や ID:4 等の DSC が複数回利用されることで，個別に作り込んだ場合よりも行数が減った為と考えられる．なお，既存手法 [1] との比較については，開発工数では提案手法と基本的には変わらない．

5. おわりに

本稿では，車載システム向けのデータストリーム管理システムに対して，車載データを処理するクエリをサービスと定義し，車載データ処理間の接続に SOA のアイデアを適用することでクエリを自動構築する手法を提案した．これにより，実機の物理構造や QoS に適合したクエリを自動構築することを可能とし，車種に依存しないクエリの設計/実装を実現し，低レベルな車載データの処理をクエリから省略可能にする仕組みも実現した．また，サービスの選択/接続後は，サービスを構成するクエリをインライン展開することで，DSMS のクエリ最適化を効果的に適用できることを確認した．なお，組込みボード上で，衝突検知クエリが動作することを確認し，SOA という観点では，組込み環境にも適用できる手法を提案した．

今後の課題としては，車載システムを構成する複数の ECU にクエリを分散配置することも含めたアルゴリズムの提案や，規模の大きなクエリに対する計算量の削減，車載データの精度の扱いや，通信の切断等による情報源の動的な切り替え等がある．

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] J. Liu and F. Zhao：Composing semantic services in open sensor-rich environments, IEEE Network, pp.44-49, (2008).

[2] 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. DE, データ工学, (2010).

[3] 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).

[4] 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, 第 4 回 Web とデータベースに関するフォーラム論文集, 1G-2-4, pp.1-9, (2011).

[5] 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, 情報処理学会研究報告, Vol.2011-EMB-23, No.1, pp.1-8, (2011).

[6] Stratis D. Viglas, Jeffrey F. Naughton：Rate-based query optimization for streaming information sources, Proceedings of the 2002 ACM SIGMOD international conference on Management of data (2002).

[7] Qi Yu, Xumin Liu, Athman Bouguettaya, Brahim Medjahed：Deploying and managing Web services: issues, solutions, and directions, The VLDB Journal, Volume 17 Issue 3, (2008).

[8] ETSI TR 102 863 (V1.1.1), Intelligent Transport Systems (ITS): Vehicular Communications; Basic Set of Applications; Local Dynamic Map (LDM) Rationale for and Guidance on Standardization, (2011).

[9] D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik.：Monitoring streams–a new class of data management applications. Technical Report CS-02-04, Brown Computer Science (2007).

[10] Borealis Distributed Stream Processing Engine, http://www.cs.brown.edu/research/borealis/public/

[11] Stanford Stream Data Manager, http://infolab.stanford.edu/stream/

[12] Borealis Application Programmer's Guide, http://www.cs.brown.edu/research/borealis/public/publications/borealis_application_guide.pdf

[13] 交通事故総合分析センター 事故統計（平成 20 年自動車事故）.

[14] LeBlanc, D., Kiefer, R., Deering, R., Shulman, M., Palmer, M. and Salinger, J.：Forward collision warning: preliminary requirements for crash alert timing. Society of Automotive Engineers (2001).

[15] Zhengrong Yang, Takashi Kobayashi, and Tsuyoshi Katayama：Development of an Intersection Collision Warning System Using DGPS, SAE World Congress (2000).

[16] Altera: Nios II 3C25 Microprocessor with LCD Controller Data Sheet, http://www.altera.co.jp/literature/ds/ds_nios2_3c25_lcd.pdf

[17] RoboCar ZMP RC-Z, http://www.zmp.co.jp/e-nuvo/pdf/catalog/robocar-110.pdf