Vol.2012-SLDM-155 No.18
Vol.2012-EMB-24 No.18
2012/3/3

Cloudia：車載データ統合プラットフォーム −基本コンセプト−

佐藤健哉[†1,†2] 勝沼聡[†2] 山口晃広[†2]
島田秀輝[†1] 本田晋也[†2]
中本幸一[†3,†2] 高田広章[†2]

近年，車両に搭載されたセンサにより走行環境を認識し，ドライバへの警告や自動で危険を回避する安全運転支援システムが登場してきた．しかし，車載センサとそのデータの種類・量の増加に伴い，アプリケーションのデータ依存による再利用性や代替性が低下し，システム全体の設計・開発の複雑化が問題となっている．この問題へのアプローチとして本研究では，ストリームデータ処理技術を適応した車載データ統合アーキテクチャに基づく組込みシステム実現のためのソフトウエアプラットフォーム (Cloudia) の構築を行い，その有効性，実現性の検討を行う．

Cloudia : A Platform for Automotive Data Integration Architecture –Basic Concept–

Kenya Sato,[†1,†2] Satoshi Katsunuma,[†2]
Akihiro Yamaguchi,[†2] Hideki Shimada,[†1]
Shinya Honda,[†2] Yukikazu Nakamoto[†3,†2]
and Hiroaki Takada[†2]

Safety driving support systems to recognize vehicle driving environment with onboard sensors are appeared recently, which is to provide safe driving and danger warning to drivers. However, due to the systems composed of many kinds of sensors and need to handle sensor data with onboard complicated application situations, it has become difficult to design and develop the whole sophisticate systems. In this study, to approach the problems we have developed a software platform "Cloudia" based on the automotive data integration architecture with data stream processing technology, and verify its availability and realization through feasibility study.

1. はじめに

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安全運転支援システムが近年登場している[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．また，車両追従システム，レーン逸脱警告システム，自動駐車システムなどもある．安全運転支援システムにおいて，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，カメラを始め車輪速センサ，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．一方で，車載システムの構成も発展してきた．電子制御ユニット (ECU) などの各ノードが単体で動作するスタンドアロンから，通信バス経由で情報を交換するデータ通信，そして，ネットワークを利用して複数ノードがリソースの共有を行う形態へと発展し，現在は，AUTOSAR[2)] に代表されるように，車載ソフトウエアの規模と複雑性の増大に対処するため，ソフトウエアプラットフォームを利用したアプリケーション連携の時代になりつつある．しかし，多様なセンサが多数搭載されたシステムにおいては，今後，ソフトウエアに加えて，データ自体の規模と複雑性の増大が予想される．特に，異なるセンサ情報を統合して利用したり，新規のセンサやアルゴリズムを追加する場合は，アプリケーションプログラムを大きく変更する必要が生じ，それに伴いシステム全体の設計・開発も困難となる．

これらの問題に対処するため，本研究では，AUTOSAR の仮想機能バス (Virtual Function Bus)[3)] のようにデータ通信路の確立によるアプリケーション連携のアプローチではなく，システム全体からセンサ部を切り離し，センサから得られたデータに対して，ストリームデータ処理技術を利用し統合管理を行う車載データ統合アーキテクチャ(**図 1**) のアプローチに基づく組込みソフトウエア開発プラットフォームの構築を行う．これにより，複数の安全運転支援システムにおいて利用されているそれぞれのセンサを統一したインタフェースの共通利用でき，センサやアプリケーションの追加・削除が容易に行え，また，アプリケー

†1 同志社大学モビリティ研究センター
Mobility Research Center, Doshisha University
†2 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
Center for Embedded Systems , Nagoya University
†3 兵庫県立大学大学院応用情報科学研究科
Graduate School of Applied Informatics, University of Hyogo

Vol.2012-SLDM-155 No.18
Vol.2012-EMB-24 No.18
2012/3/3

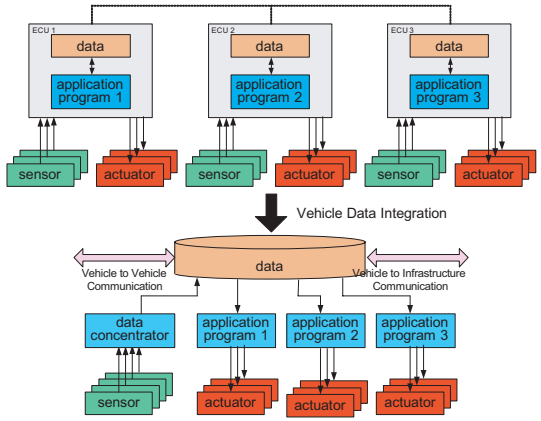

図 1 車載データ統合アーキテクチャ
Fig. 1 Vehicle data integration architecture.

ションソフトウエアの再利用性が高まり，システム開発のコスト削減にもつながる．

2. Cloudia: 車載データ統合プラットフォーム

2.1 コンセプト

本研究で提案する車載データ統合アーキテクチャを実現するための組込みソフトウエアプラットフォーム (Cloudia: Cloud technology for Data Integration Architecture) のシステム構成を**図 2** に示す．複数のアプリケーションにおいてそれぞれ管理されていたデータを物理的あるいは仮想的にデータ空間として統合管理し，センサを切り離す構成を採る．現行のシステム構成では，1つあるいは複数の ECU においてアプリケーションプログラムがデータ処理を行い，その結果，アクチュエータの操作を行う．そのため，センサの構成や設置状況の違いにより，アプリケーションプログラムを変更する必要が生じ，また，複数アプリケーションの設計や開発を統合することが困難となる．

一方，車載データ統合アーキテクチャでは，周辺監視のためのレーダやカメラなどのセンサ情報から得られる物体の存在情報を確率として合算し，車速/車輪速センサやステアリングセンサ，加速度/角速度センサなどから得られる車両の運動状況を示すデータを統合化することで，複数のアプリケーションから共通に利用することが可能となる．

2.2 ストリームデータ処理の適応

リアルタイム性に関する要求の高いデータに対して有効であるストリーム処理技術をセンサデータに適応することで，様々なデータ処理の共有化によるコスト削減しつつ，そのデータ処理結果の高速な配信を実現できる．そこで車載データ統合アーキテクチャのデータ空間において，ストリーム処理技術を利用したデータ管理を適応したデータストリーム管理システム (DSMS)[4] を適応する．DSMS は従来の蓄積型データベース管理システム (DBMS) と異なり，時々刻々センサから送られてくる一過性のデータを逐一データベースに格納した後に処理するのではなく，CQL (Continuous Query Language)[5] を用いて，Filter, Union, Map, Join, Aggregate などのオペレータを用いて処理を行う．クエリは，各オペレータ間のデータの流れを指定することで実現され，そのオペレータを置換，追加することにより，処理内容を容易に統合，分散することが可能となる．

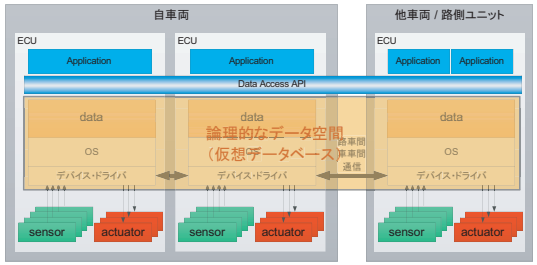

図 2 Cloudia: 車載データ統合プラットフォーム
Fig. 2 Cloudia : A platform for automotive aata integration architecture.

2.3 プラットフォーム構成要素

車載データ統合プラットフォーム Cloudia は，eDSMS(組込みシステム向け DSMS)，および，コンフィグレータの構成要素からなる．

2.3.1 eDBMS

図 3 に eDSMS の構成を示す．eDSMS のランタイムは，ストリーム管理部 (固定長キュー/可変長キュー)，オペレータ実行部 (Filter/Map/Union/Join/Aggregate)，ウインドウ管理部 (個数/時間/キー毎ウインドウ)，集計関数 (Count/Sum/Average/Max/Min)，スケジューラ (入力順/時刻順/優先度付)，データ通信部 (TCP/UDP/CAN/FlexRay/Lin) のコンポーネントグループ (コンポーネント) で構成される．

eDSMS は従来の汎用システム向け DSMS と異なり，設計時に静的に処理内容が決定，不要ストリームおよびスケジューラの抽出，クエリのソースコードの最適化，パーサからのランタイム分離，起動時のパーサ不要，コンパイル対象のコンポーネントの限定といった要因から，小容量，高速・低遅延，カスタマイズ可能が特徴である．

Vol.2012-SLDM-155 No.18
Vol.2012-EMB-24 No.18
2012/3/3

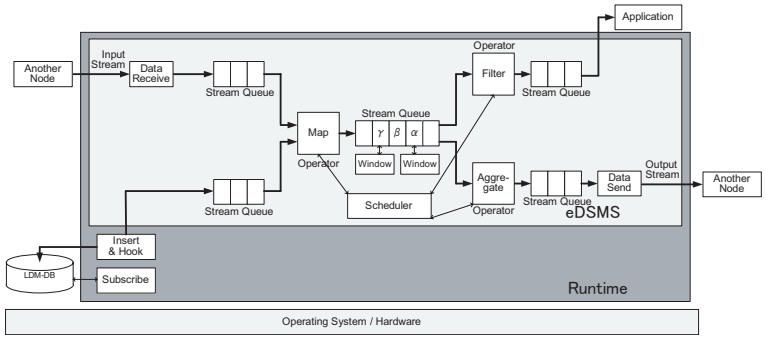

図 3 eDSMS: 組込みシステム向け DSMS
Fig. 3 eDSMS: embedded Data Stream Management System.

2.3.2 コンフィグレータ

車載データ統合プラットフォームを実現するためのコンフィグレータはオプティマイザとジェネレータから構成される．オプティマイザを実施するために，サブクエリを XML パース前にインラインで展開し，XML パース後はクエリ記述として扱うことが可能となる．サブクエリが階層構造になっている場合でも，階層構造を展開することで，最終的に通常のクエリ記述となる．サブクエリ展開後は，個々のオペレータとして構成されているので，クエリ最適化を実施しやすく，ユーザ定義のオペレータよりも処理を効率化することが可能となる．ジェネレータは，XML で記述されたクエリおよび配置情報からランタイム起動部のソースコードを生成する．その後，ランタイム本体を記述したソースコードと合わせてコンパイルすることで，ランタイムのバイナリを生成する．

2.4 アプリケーション適応

本研究において，車載データ統合アーキテクチャを LDM (Local Dynamic Map) へ適応したシステム (Stream-LDM) を開発している．LDM とは，車々間・路車間通信を利用し情報交換を行う協調 ITS で利用することを目的に，地理的情報，周辺車両・道路状態・交通状況などの関連情報を階層的に管理・保持している概念的なデータの集合体である．車載センサおよび路車間・車々間通信を通じて周辺状況の走行車両や歩行者，障害物などのデータを集約し，高精度地図上に重畳し，安全運転支援や交通管理の実現が可能となる．たとえば，交差点において直進車両と右折車両の衝突を検知するクエリを**図 4** に示す．

ストリーム LDM の概要を**図 5** に示す．車載センサデータあるいは路車間・車々間通信より得られるデータをストリームとして，ストリーム処理モジュールで演算するとともに，

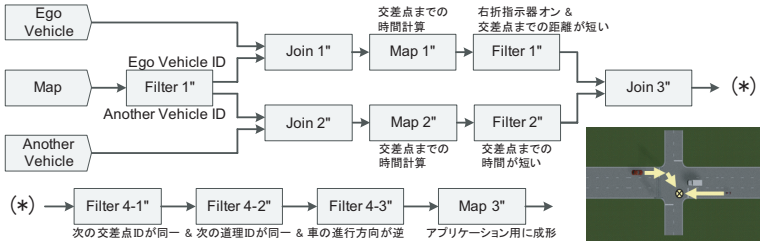

図 4 交差点において直進車両と右折車両の衝突を検知するクエリの例
Fig. 4 A query example for intersection collision warning system.

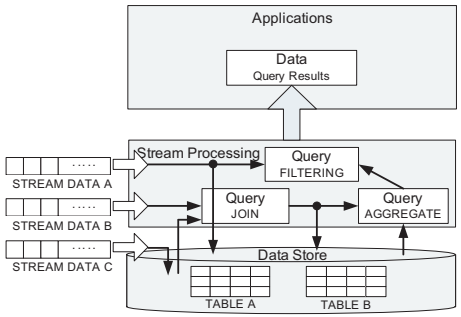

図 5 ストリーム LDM 構成
Fig. 5 Stream LDM architecture.

データストア部へも保持する．アプリケーションは，各オペレータのパラメータを設定することで処理ブロックを定義し，その処理ブロック間のデータフローを指定することにより，出力すべきデータが出現した場合に自動的にアプリケーションに対して応答を返すため，高速で処理が可能となる．

3. ソフトウエアプラットフォーム

3.1 開 発 手 順

一般的に車載システムにおいては，安全性向上とリアルタイム性保証のため，設計時にシステムの解析を行いソフトウエアモジュールの機能や配置を静的に構成する．本プラットフォームにおいても，設計時に静的に定義したクエリに従って，必要最小限となるようにコンポーネントを含むコードを生成し，CPU，メモリ，ネットワークの物理的資源制約を満

Vol.2012-SLDM-155 No.18
Vol.2012-EMB-24 No.18
2012/3/3

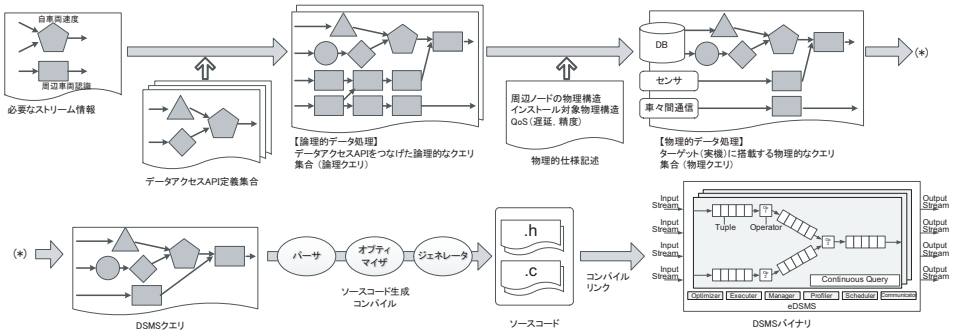

図 6 Cloudia によるソフトウエア開発手順
Fig. 6 Software development process with Cloudia.

たす手法を採る．**図 6** に Cloudia のソフトウエア開発手順を示す．具体的には，(1) 論理的データ処理，(2) 静的クエリ構成，(3) 物理的データ処理，(4) ソースコードコンパイル，(5) バイナリ生成の手順となる．

3.1.1 論理的データ処理

アプリケーションの一部を実現するための意味的な階層モデル (データアクセス API の定義集合) 構成のクエリ記述を，入力ストリーム (センサ情報) まで遡って生成する過程を論理的データ処理と呼び，この過程で生成される論理的なクエリ集合を論理クエリと定義する．ここでデータアクセス API とは，データの意味とスキーマを定義したインタフェースである．たとえば，クエリ記述が「前方車の位置」の場合，自車位置と前方車との相対距離が判明すれば前方車の位置を導き出すことが可能である．この際，具体的なデバイスや情報源の位置を意識せずに生成したクエリプランが論理クエリとなる．

3.1.2 論理クエリ構成

アプリケーションに求められる QoS やハードウエア資源の制約に基づいて，論理クエリからターゲットとなるシステムに搭載するクエリ作成の過程を論理クエリ構成と呼び，この段階でデータアクセス API 単位で必要のないオペレータを削除することでソフトウエア設計の最適化を行うことが可能となる．また，アプリケーション全体を実現するために内部がブラックボックスのノード (たとえば，他社が開発した ECU) を利用する場合，このブラックボックスのノードの出力を情報源とした論理クエリとみなすことが可能となる．この論理クエリを利用してアプリケーションの動作を記述することで，システムの相違による具体的な実装の違いを隠ぺいすることが可能となる．

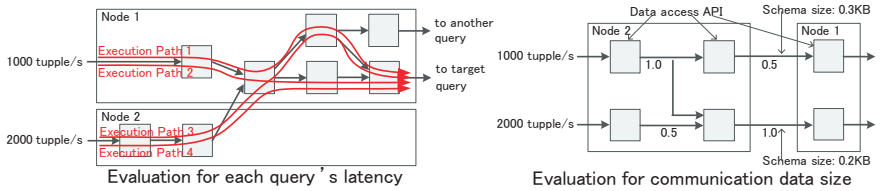

図 7 物理的データ処理例
Fig. 7 An example of physical data execution.

3.1.3 物理的データ処理

各クエリのレイテンシ，全クエリのサイズ，および，ノード間の通信量を指標として，物理的なクエリの集合を生成することを物理的データ処理と呼ぶ．各クエリのレイテンシは，**図 7** に示すように，クエリの入力ストリームに入力してから出力ストリームにデータが達するまでの実行パスの平均時間を算出する．この際に，ストリームの転送時間とオペレータの処理時間を考慮するが，全体のレイテンシにおいて支配的なのは，ネットワークを介した転送時間と処理負荷の高いオペレータの処理時間となる．全クエリのサイズは，各オペレータのサイズおよびストリームのサイズを合算して見積もる．各スキーマのサイズ，オペレータのウインドウサイズ，ストリームのキューサイズは，XML 形式の物理的仕様記述から概算する．ノード間の通信量は，図 7 において，入力ストリームレート (tupple/s) と，スキーマのサイズを利用し，各データアクセス API の出力ストリームのデータレートを入力側から順に見積もる．

3.1.4 ソースコード生成

プラットフォームを実現する C 言語ソースコード生成のために，パーサにより XML 形式で定義された物理クエリをパース表現を生成する．これは XML 形式をグラフ構造に変換したものであり，クエリを構成するオペレータのノードがそのオペレータを転送するストリームのノードで接続されたグラフとして表現する．パース表現に対して，オプティマイザによりクエリ処理最適化を行う．状況によってストリームへのアクセスやスケジューラ呼び出しを除去し，クエリの実行表現を生成する．最後にジェネレータにより，クエリの実行表現からランタイムのクエリ処理部のソースコードと初期化部を生成する．

3.1.5 バイナリ生成

パーサ，オプティマイザ，ジェネレータにより生成されたソースコードをコンパイルし，ランタイムの必要コンポーネント，アプリケーションとリンクを行い，最終的にランタイムのバイナリコードを生成する．

Vol.2012-SLDM-155 No.18
Vol.2012-EMB-24 No.18
2012/3/3

```
<output callback="callbackForApp1" schema="ForwardVehicleRecognition"
                                        stream="forwardVehicleRecognition"/>
    <output callback="callbackForApp1" schema="Intersection" stream="intersection"/>
    <query name="queryForApp1">
        <box AbstractHighLevelAPI="forwardVehicleRecognition">...</box>
        <box AbstractHighLevelAPI="intersection">...</box>
    </query>
```

図 8 XML によるクエリ記述の具体例
Fig. 8 An example of query description.

3.2 アプリケーション開発具体例

ここではアプリケーション開発の具体例を示す．たとえば，前方を走行する車両を考慮しつつ交差点付近に接近したらその周辺地図を取得したいといった交差点案内アプリケーションを設計する場合を考える．まずは XML を利用してクエリ記述を行う．具体的なクエリ記述の XML ソースコードは**図 8** のようになる．アプリケーション開発において設計者はこの XML ソースコードを記述するだけである．この XML ソースコードから，複数の過程を経て物理クエリ設計を行う．この過程を**図 9** に示す．

図 9(1) のクエリ記述を基盤とし，今回の設計においては，前方車との距離を判定する手段は，レーザレーダ，ミリ波レーダ，カメラの 3 通りが選択可能であるとする．ステアリング，勾配検知のセンサの設置を前提とすると，今回の場合，論理クエリの総数は 12 通りとなる．この状態を図 9(2) に示す．そこで，処理ノードのデータアクセス API と当該ノードの情報源を考慮すると，図 9(3) の論理クエリ構成の状態となる．

次に**図 10** に示すアプリケーションの QoS および計算資源の制約と統計情報の物理的仕様記述から，図 9(4) の物理クエリ設計を実施する．ここでは，各指標の見積もりを次のように検討する．

- forwardVehicleRecognition の精度 $= (0.8 * 0.7 + 0.5 * 0.9)/2 = 0.505$
- intersection の精度 $= 1.0$
- intersection にかかる時間 $= 1 * (500/800) = 0.6ms$
- forwardVehicleRecognition にかかる時間 $= [\{10 * (50/100) + 1.0 * (100/100) + (100 * 0.5/1)\} + \{2 + (3 * 1.2/1)\}]/2 \sim 6ms$
- クエリ全体のサイズ $= 1 + 0.5 = 1.5KB$

各クエリの処理時間やクエリ全体のサイズは、正確にはオペレータ単位で見積もるが，計算を簡略するため，ここではデータアクセス API 単位で見積もっている．また，ノード間の

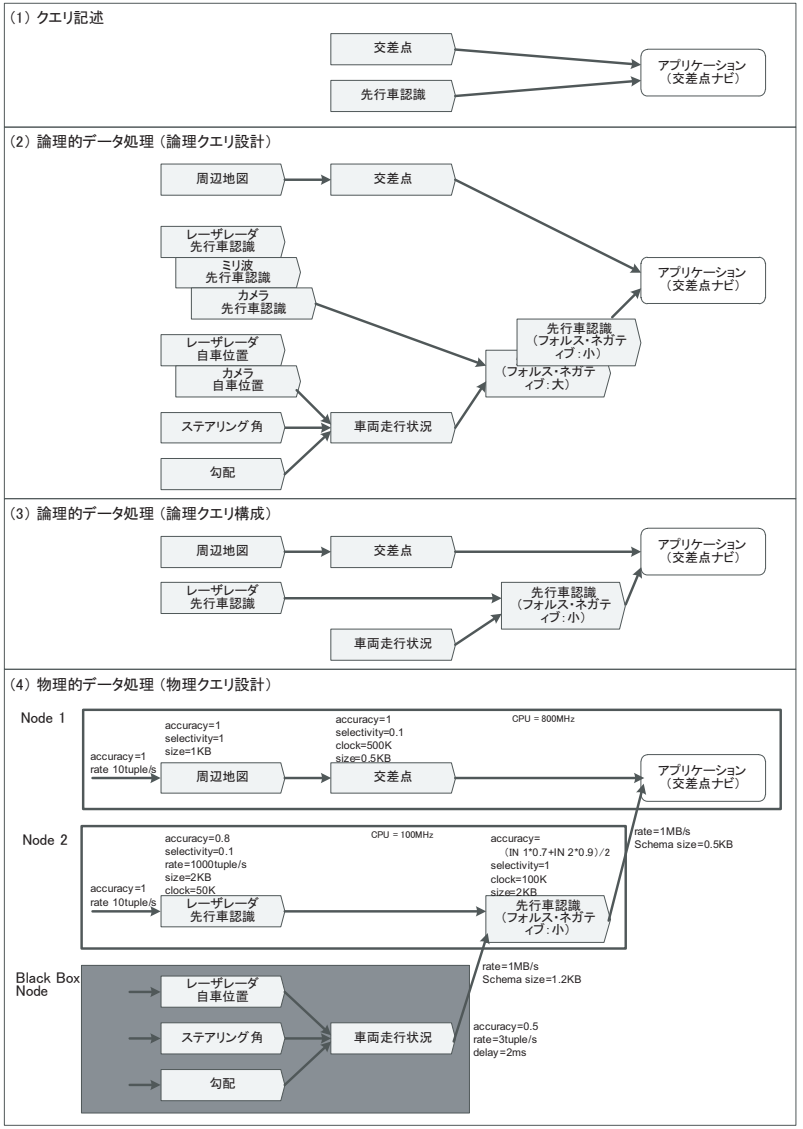

図 9 物理クエリ生成例
Fig. 9 An example of physical query generation.

Vol.2012-SLDM-155 No.18
Vol.2012-EMB-24 No.18
2012/3/3

```
<requirement>
    <qos>
        <accuracy stream="forwardVehicleRecognition" weight="0.5" constraint="0.5" />
        <accuracy stream="intersection" weight="0.8" constraint="0.8" />
        <latency stream="forwardVehicleRecognition" weight="0.5" constraint="20" />
        <latency stream="intersection" weight="0.5" constraint="20" />
        <size node="target node ID" constraint="2000" />
    </qos>
    <constraint>
        <accuracy stream="forwardVehicleRecognition" value="0.5" />
        <accuracy stream="intersection" value="0.8" />
        <delay stream="forwardVehicleRecognition" value="20" />
        <delay stream="intersection" value="20" />
        <size value="2000" /><!--byte-->
    </constraint>
</requirement>
```

図 10 XML による物理的仕様記述の具体例
Fig. 10 An example XML code for physical specification description.

ネットワークの通信量は省略している．QoS の各値は設計時にテンプレート等で代表値を選ぶものとする．また，今回の物理クエリ設計においては，車両走行状況は他社開発 ECU が出力するとして，内部がブラックボックスノードと設定する例を示す．

4. 実装と評価

4.1 実装環境

情報系 ECU および制御系 ECU への適応を前提に，次の 2 種類の実行環境を利用して Cloudia を実装し，基本的な動作を確認した．

(1) ZMP 社 RoboCar® 1/10 モデル RC-Z（情報系 ECU 相当）
- CPU: AMD Geode® LX800 Processor 500MHz
- メモリ: 512MByte
- OS: Linux（Fedore 10）

(2) Altera 社 Nios II Cyclone III 3C25 FPGA ボード（制御系 ECU 相当）
- CPU: Nios II/f processor core
- メモリ: DDR SDRAM 133 MHz, 32 MBytes
- OS: TOPPERS/ATK2（AUTOSAR 準拠）

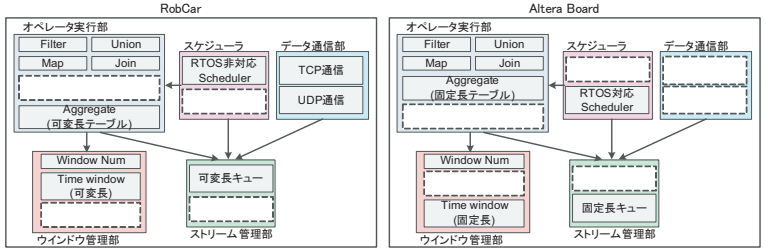

図 11 RoboCar と Altera ボードでの実装評価
Fig. 11 Implementation evaluation for RoboCar and Altera board.

4.2 評価結果

RoboCar および Altera ボードにおいて，**図 11** に示すストリームデータ処理のコンポーネントを選択することで動作を確認した．RoboCar では PC 上と同様のコンポーネントで動作することが可能であった．一方、Altera ボードでは Robocar とは異なり動的なメモリ割当て (ヒープ) に OS が対応していないため，ストリームのキュー，ウインドウ，Aggregate オペレータに固定長のメモリ領域を活用する構成とした．コンポーネント置換え時に，他コンポーネントの変更が発生しないことが確認できた．具体的には，双方の環境において，Filter，Map，Union，Join オペレータ，及び，個数ウインドウは共通のコンポーネントを活用することができ，置換えた他コンポーネントの影響を受けないことを検証した．

謝辞 本研究の一部は科研費 (21500084) の助成を受けたものである．

参 考 文 献

1) 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,Vol.89, No.8, pp.72–75 (2007).
2) Fürst, S. et al.: AUTOSAR – A Worldwide Standard is on the Road, *14th International VDI Congress Electronic Systems for Vehicles* (2009).
3) Schreiner, D., Goschka, K.: A Component Model for the AUTOSAR Virtual Function Bus, *31st Annual International Computer Software and Applications Conference*, Vol. 2, pp.635-641 (2007).
4) Rajeev, M. et al.: Query Processing, Resource Management, and Approximation ina Data Stream Management System, *Conference on Innovative Data Systems Research*, (2003).
5) Arasu, A. et al.: The CQL Continuous Query Language: Semantic Foundations and Query Execution, The VLDB Journal, Vol.15, Issue 2, pp.121–142 (2006).