Electronic Preprint for Journal of Information Processing Vol.25

Regular Paper

In-vehicle Distributed Time-critical Data Stream Management System for Advanced Driver Assistance

Akihiro Yamaguchi[1,a)] Yousuke Watanabe[2] Kenya Sato[3,4] Yukikazu Nakamoto[3,5]
Yoshiharu Ishikawa[1] Shinya Honda[1] Hiroaki Takada[3,2]

Received: March 20, 2016, Accepted: May 11, 2016

Abstract: Data stream management systems (DSMSs) are suitable for managing and processing continuous data at high input rates with low latency. For advanced driver assistance including autonomous driving, embedded systems use a variety of onboard sensor data with communications from outside the vehicle. Thus, the software developed for such systems must be able to handle large volumes of data and complex processing. We develop a platform that integrates and manages data in an automotive embedded system using a DSMS. However, because automotive data processing, which is distributed in in-vehicle networks of the embedded system, is time-critical and must be reliable to reduce sensor noise, it is difficult to identify conventional DSMSs that meet these requirements. To address these new challenges, we develop an automotive embedded DSMS (AEDSMS). This AEDSMS precompiles high-level queries into executable query plans when designing automotive systems that demand time-criticality. Data stream processing is distributed in in-vehicle networks appropriately, where real-time scheduling and senor data fusion are also applied to meet deadlines and enhance the reliability of sensor data. The main contributions of this paper are as follows: (1) we establish a clear understanding of the challenges faced when introducing DSMSs into the automotive field; (2) we propose an AEDSMS to tackle these challenges; and (3) we evaluate the AEDSMS during run-time for advanced driver assistance.

Keywords: data stream management system (DSMS), distributed stream processing, real-time scheduling, earliest deadline first (EDF), sensor fusion, automotive system, advanced driver assistance system (ADAS)

1. Introduction

Automotive systems utilize various onboard sensors such as radar, lasers, cameras, and speed sensors, in order to observe the status and surroundings of a vehicle. By using these sensors to provide warnings to the driver or operate the vehicle automatically, automotive applications such as pre-crash safety systems, adaptive cruise control, and lane departure warning systems have been demonstrated and commercialized [1]. Research and develop on autonomous driving systems has increased in popularity. Google and Urban Challenge, which is a competition funded by the Defense Advanced Research Projects Agency, have demonstrated that self-driving is possible and safe in urban areas [2], [3]. In such situations, managing and processing automotive sensor data is quite complicated.

Sensor data processing in these scenarios is time-critical, often imposing deadlines (i.e., a real-time requirement). For data processing with a real-time requirement, missed deadlines may lead to serious accidents. In order to meet deadlines, algorithms for real-time scheduling have been studied in the field of real-time systems. On the other hand, because raw sensor data includes uncertain information, enhanced reliability is a necessity. In order to improve positional accuracy for tracking vehicles and/or pedestrians, algorithms for sensor data fusion are often used to enhance reliability by integrating multiple sensors and the Global Positioning System (GPS).

For advanced driver assistance, developers have been planning to utilize cooperative intelligent transport systems (C-ITSs) to improve transport safety and reliability using vehicle-to-infrastructure [*1] (V2I) and vehicle-to-vehicle (V2V) communications, as well as GPS and onboard sensors. The European Telecommunications Standards Institute (ETSI) has promoted the international standardization of C-ITSs. This standardization process has included communication protocols and data exchange formats employed by the protocols [4], [5]. Because V2V/V2I communications can broadcast to surrounding vehicles in 100 ms cycles [4], the input-data volumes and arrival-time of the communications may change while a vehicle is driving.

The configuration of an automotive system is distributed and complicated. In today's luxury cars, the system comprises approximately 70–100 nodes such as electronic control units

[1] Graduate School of Information Science, Nagoya University, Nagoya, Aichi 464–8601, Japan
[2] Institute of Innovation for Future Society, Nagoya University, Nagoya, Aichi 464–8601, Japan
[3] Center for Embedded Computing Systems, Nagoya University, Nagoya, Aichi 464–8603, Japan
[4] Mobility Research Center, Doshisha University, Kyotanabe, Kyoto 610–0321, Japan
[5] Graduate School of Applied Informatics, University of Hyogo, Kobe, Hyogo 650–0047, Japan
[a)] yamagut.ertl@gmail.com;yamagut@ertl.jp

[*1] Infrastructure has a specific meaning in ITSs, where it comprises the roads, centers, and facilities around vehicles.

(ECUs) connected by in-vehicle networks. Application software and sensors are embedded in a distributed manner in the nodes. Thus, the complexity of software development has increased for distributed systems in addition to the growing scale and complexity of the data. In these situations, automotive manufacturers must address the increased costs of embedded software development and required software platform.

A data stream management system (DSMS) is suitable for processing and managing continuous data. It is relatively easy to design data processing in the form of query descriptions. Moreover, DSMSs can often be applied in distributed systems. DSMSs manage data as a stream, which is a sequence of tuples. For this reason, data stream processing is low-latency and suitable for use with changing data rates. In the past, DSMSs were applied mainly in networking fields (traffic engineering, intrusion detection, and sensor networks) and finance (arbitrage and financial monitoring). We have been researching and developing a data-centric software architecture for automotive embedded systems by DSMSs [6].

However, we have encountered new challenges when applying DSMSs to automotive data processing. In particular, because deadlines are required rather than reduced average latency or improved throughput, a DSMS should have architecture suitable for time-critical purposes and should meet deadlines in distributed stream processing. Furthermore, stream processing is distributed in in-vehicle networks, where previous methods cannot be directly applied. In addition, a DSMS must provide reliable sensor data using sensor fusion, where the arrival-timing of input-data from V2V communications is out-of-order. Previous DSMSs have difficulties handling these challenges.

1.1 Contributions and Roadmap

To address these challenges, we developed an Automotive Embedded Data Stream Management System (AEDSMS) at the Center for Embedded Computing Systems of Nagoya University (NCES), which is an organization that acts as a center for research and education in automotive embedded systems technology based on industrial-university cooperation [*2]. We promoted this research by establishing a consortium with seven companies, including Toyota, Hitachi, and Denso, and three additional organizations, including the National Institute for Land and Infrastructure Management and the Japan Digital Road Map Association.

The main contributions of this paper are as follows:

(A) Clarifying the challenges of applying DSMSs to automotive data processing.

(B) Developing an AEDSMS to overcome these challenges, which is a DSMS with the following features:
  (a) The DSMS architecture precompiles high-level queries into an executable query plan in the design of automotive systems;
  (b) Distributed stream processing for in-vehicle networks;
  (c) Real-time scheduling for distributed stream processing;
  (d) A sensor fusion operator for out-of-order input-streams.

(C) Evaluating the feasibility of the AEDSMS during runtime for advanced driver assistance, according to ETSI specifications.

This paper is the extended version of Ref. [7] and includes additional evaluations and explanations. In particular, regarding contribution (B.c), this paper extends [8], which proposes a method of stream processing on a single node into distributed stream processing.

The remainder of this paper is organized as follows. Section 2 provides an overview of related work. Section 3 presents a brief summary of the research field and our motivation for applying a DSMS to automotive systems. Section 4 considers the important challenges that must be addressed when applying DSMSs to the field of interest. Section 5 introduces the AEDSMS. Section 6 presents an evaluation of the effectiveness of the AEDSMS based on its application to an advanced driver assistance system. Section 7 summarizes how the AEDSMS overcomes various challenges, finally, and our conclusions are provided in Section 8.

2. Related Work

2.1 Automotive Software Platforms without Data Streams

The AUTomotive Open System ARchitecture (AUTOSAR) is a major component-based, automotive control software platform and software development methodology that facilitates software reusability and production [9]. The partners in the consortium include many automotive companies; carmakers use AUTOSAR as the foundation of their commercial automotive systems. However, AUTOSAR does not provide a solution to data management problems, as described in Section 3.2.

The SAFESPOT project developed a platform of automotive systems for managing data obtained from onboard sensors and V2V/V2I communications using a relational database (RDB), where the members of the consortium, including Bosch and Volvo, performed demonstration experiments [10]. In addition, [11], [12] developed an RDB to satisfy the real-time requirements of automotive data management, where deadlines must be met. However, RDBs are not suitable for automotive sensor data because their update frequency is high, and it is necessary to process data continuously at low-latency in driver assistance systems.

2.2 Data Stream Management Systems

A data stream management system is a suitable platform for managing continuous data such as sensor data and network packets at a low-latency. Many DSMSs have been developed for general-purpose stream processing systems, which include STREAM, Aurora, Gigascope, InfoSphere Streams, as well as complex event processing systems such as Oracle CEP and Esper [13]. Recently, platforms for scalable, distributed, and parallel stream processing have been developed using cloud computing, such as MillWheel and Muppet [14], [15]. However, these platforms do not meet the requirements of the automotive field, as described in Section 4, and their software sizes are too large for embedded deployment.

In the sensor networks field, DSMSs have been developed with sufficiently small footprints to install in embedded systems, as well as the automotive field. For example, [16] proposed an on-

[*2] Project URL: http://www.nces.is.nagoya-u.ac.jp/project/DataInteg_2010.pdf (in Japanese)

board stream processing method for engineering testing and diagnosis in vehicle systems. The Vehicle Data Stream Mining System (VEDAS) [17] and Minefleet [18] are distributed data stream mining platforms for use in mobile computing devices or automotive systems. StreamCars is a stream processing platform for driver assistance systems, which shares a similar objective as our research [19]. This method provides sensor fusion operators to enhance the reliability of sensor data. However, the performance and implementation have not been described in detail. In addition, previous studies have not considered either distributed stream processing systems in in-vehicle networks or their real-time requirement, although they are important in automotive data processing, which means that they do not satisfy the requirements described in Section 4.

2.3 Real-time Scheduling of Data Stream Processing

In order to satisfy the real-time requirements, many studies have used real-time scheduling to determine the order of processing to meet deadlines [20]. Some studies have also addressed stream processing in the real-time community and database community. In particular, most of the proposed methods have been based on earliest deadline first (EDF), which is an algorithm for real-time scheduling that can handle cases where the input-data rate changes [21], [22], [23]. In these methods, users specify the End-to-End deadline, which is the maximum allowable end-to-end latency between when the data is read from a sensor as tuples and when it is processed, for each output-stream. However, these methods have limited applicability in stream processing, and it is difficult to apply them directly to automotive data processing.

Zhengyu et al. [21] and Yuan et al. [22] implicitly considered a query as a scheduling task and required a one-to-one relationship between a query and an End-to-End deadline. As a result, they were unable to deal with a multiquery, which shares part of data processing and delivers multiple output-streams. The method proposed by Xin et al. [23] is less limited than other methods [21], [22], and it can deal with multiqueries by dividing the query graph so that each task includes one output-stream. In contrast, there may exist tasks without End-to-End deadlines in distributed stream processing because the query plan is divided and distributed among nodes. Therefore, previous methods cannot be used with distributed stream processing with multiqueries. Based on previous research [22], [23], some studies have addressed Quality of Service (QoS) management and load shedding to meet End-to-End deadlines [24], [25], [26]. However, this method cannot operate in distributed stream processing with multiqueries for the aforementioned reasons.

2.4 Operator Placement of Distributed Stream Processing

The operator placement problem is the key issues of distributed stream processing because it is important for improving performance. It is NP-hard because it has combinatorial complexity where operators and streams are allocated in nodes and networks [27]. For this reason, various heuristic methods of operator placement have been proposed to reduce network usage and balance the load among nodes [28]. A method that formulates the operator placement problem as mixed-integer linear programming (MILP) has also been proposed [29]. However, these previous methods assume overlay (mainly peer-to-peer) networks. There are no existing methods that apply to in-vehicle networks, which have a hybrid network topology of mainly bus networks and gateways.

3. Background and Motivation

In this section, we explain the features of automotive data processing and the reason for introducing DSMSs into automotive embedded systems.

3.1 Automotive Data Processing

The applications of an automotive embedded system often has a lot of data processing because the process flow of the application involves three steps: (1) obtaining input-data from input-devices such as sensors and communication equipment distributed among many nodes, (2) processing and combining the data, and (3) controlling the requisite actuators, which are also distributed among many nodes.

We now describe the features and requirements of automotive data processing.

Real-time Requirement

Data processing has End-to-End deadlines, which is specified by users according to the specifications of applications when designing automotive systems. For example, ETSI specifies the End-to-End deadline for vehicle collision warnings to be 300 ms [30]. For vehicle control, the End-to-End deadlines are approximately 10–100 ms, depending on the application. This feature is required for driver assistance systems, which are safety-critical.

Reliability Requirement

The error between the value observed from an input-device and the true value must be minimal to provide accurate information such as vehicle position. ETSI has defined the data dictionary used in C-ITS communications, which requires that confidence information be attached to sensing data, such as the latitude, longitude, and velocity [5]. Confidence information is generally represented as a symmetric 95% confidence interval [4], [5]. Therefore, it is recommended that each vehicle broadcast its confidence information outside the vehicle, as required by ETSI.

Variation of Input-data Volumes

Automotive embedded systems are equipped with a number of sensors, where the input-data are updated every 10–100 ms. In contrast, V2V/V2I communications can broadcast to surrounding vehicles within a range of 100–500 m in 100 ms cycles [4]. The number of messages received by automotive embedded systems increases at intersections because many vehicles broadcast V2V communications at these points. ETSI specifies that a vehicle can receive a maximum of 1000 vehicle messages per second [30]. This requirement means that it is possible to input information for a maximum of 1000 vehicles per second because one message only includes information for one vehicle in V2V communications [4], [5]. However, it is difficult to process all of this input-data and meet End-to-End deadlines. Therefore, in some situations, the input-data obtained from outside the vehicle must be filtered to meet deadlines unlike input-data from onboard sensors that cannot always be filtered.

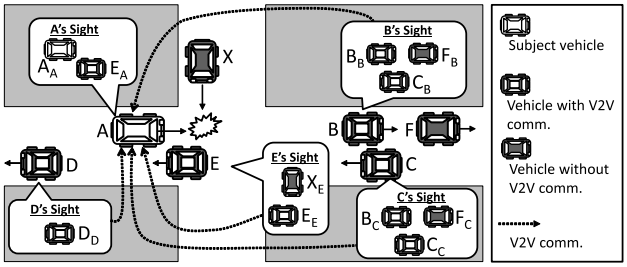


Fig. 1 Example of an intersection with collision warnings.

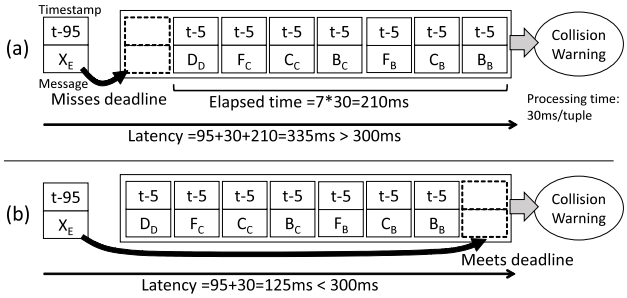


Fig. 2 Order of data processing based on V2V communications.

Out-of-order Arrival

Each vehicle that uses V2V communications broadcasts a message in the interval of 100 ms at the timing of the vehicle. As a result, it is possible for a vehicle to receive messages via V2V communications with a maximum difference of 100 ms between when the data are read by a sensor and when the sensor data are transmitted by the sender, even if communication delays are ignored. In other words, input-data from V2V communications is out-of-order.

3.1.1 Example of Real-time Scheduling

Intersection collisions are one of the most common types of accidents that account for approximately 50% of all reported crashes [31]. In particular, the use of C-ITS is expected to be beneficial at intersections with poor visibility because the use of onboard sensors, such as radar and cameras, on a vehicle cannot accurately detect other vehicles. By using the application scenario, we explain the importance of introducing real-time scheduling.

Figure 1 shows an example of a collision warning system for subject vehicle A at an intersection with poor visibility via V2V communications. Vehicles A and X approach the intersection. A does not know that X is approaching because of poor visibility. However, vehicle E can detect vehicle X using E's onboard sensors. The End-to-End deadline for collision warnings is 300 ms according to ETSI specifications [30]. If vehicle A cannot provide a collision warning for vehicle X within 300 ms after vehicle E detects X using its onboard sensors, A will collide with X. Vehicles A, B, C, D, and E broadcast the information sensed by their onboard sensors to other vehicles using V2V communications. However, vehicles X and F cannot use V2V communications. Y_Z is the message that vehicle Z broadcasts by sensing vehicle Y.

In **Fig. 2**, subject vehicle A receives message X_E regarding vehicle X from vehicle E at the current time t and waits to process the messages broadcast from vehicles B, C, and D, according to the situation shown in Fig. 1. To simplify the discussion in this example, we assume that each vehicle senses the surrounding vehicles at the same time, where E broadcasts information regarding X via V2V communications at 95 ms after sensing it, whereas B, C, and D broadcast their information at 5 ms after sensing. We assume that it takes 30 ms to process a message that has arrived from V2V communications.

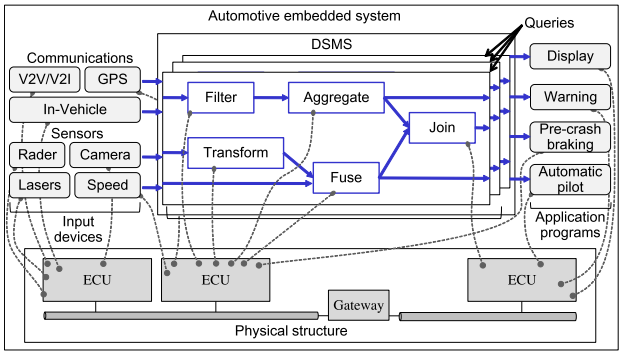


Fig. 3 DSMS in an automotive embedded system.

Figure 2 (a) shows the case where the messages are processed using first in, first out (FIFO), where the latency required to process X_E is $95+30+7\times30=335$ ms because seven messages are waiting to be processed before X_E. As a result, the End-to-End deadline of 300 ms is missed, and the subject vehicle collides with X. This shows that it is important to meet End-to-End deadlines in automotive data processing.

On the other hand, Fig. 2 (b) shows an example using real-time scheduling based on EDF [20], which determines the processing order required to prioritize the input-data with early deadlines. In this case, X_E is processed first because the deadline of X_E is the earliest. As a result, subject vehicle A processes X_E and the remaining messages within the End-to-End deadline and stops before colliding with vehicle X. Therefore, the application of real-time scheduling to automotive data processing is useful for avoiding vehicle crashes.

3.2 Reenergizing the Software Architecture Based on DSMS

In automotive systems that include a variety of sensors, the scale and complexity of the software and the volumes of data are expected to grow [32]. Traditional automotive software imposes a high degree of coupling among application programs, sensors, and their data. Automotive component suppliers provide products that satisfy the following: data processing is embedded in application programs, and the application programs use the related sensors individually. As a result, similar data processing is duplicated, although the processed data is commonly reused. In addition, the physical structure of nodes, in-vehicle networks, and onboard sensors in an automotive embedded system is usually different among vehicle types. If the physical structure is changed during traditional automotive software development, large sections of the software must be modified, which results in increased development costs for automotive embedded software.

The discussions above suggests the necessity of a data-centric software architecture and platform. In our approach, we separate data processing from the application programs. To achieve this, we use a DSMS that is suitable for data processing in automotive embedded systems. **Figure 3** shows the architecture of the DSMS

applied to an automotive embedded system. We describe automotive data processing using queries in the DSMS. The input-data from onboard sensors and communications is inserted into input-streams of the query plan as tuples, and application programs obtain the results of the queries from output-streams. By defining the schema of each stream that is independent of specific sensors and applications, sensors and applications are decoupled in this approach. In addition, it is easy to reduce duplicated processing of similar data from the query graph.

Because automotive data processing is distributed among nodes, the DSMS and query graph are also distributed. The queries of the DSMS are separated from the physical structure in order to improve the reusability of the queries among different vehicle types. By using this approach, the DSMS provides location-transparent data access to users.

The DSMS represents queries as dataflow diagrams in the same manner as Aurora and Borealis, and it follows their basic query specifications [33]. However, other DSMSs, e.g., [34], [35], represent queries using structured declarative languages like SQL. SQL-like declarative languages can simplify query descriptions, but the developers of safety-related automotive applications design data processing using a work-flow representation.

4. Challenges to Automotive Embedded Data Stream Management Systems

This section introduces the important challenges that must be addressed when embedding DSMSs in automotive systems compared with other areas such as sensor networks, financial tickers, traffic management, and click-stream inspection.

4.1 Time-criticality during Runtime (C1)

In other areas, the ability to register, remove, and modify queries as necessary on the fly is advantageous for DSMSs. Existing methods that improve the performance of DSMSs migrate operators dynamically during runtime according to the load status (i.e., input-data volumes), where their performance metrics include the average latency and throughput [28], [33]. Therefore, their query plan, including the query graph and operator placement, changes dynamically during runtime.

In contrast, automotive data processing must satisfy real-time requirements, as described in Section 3.1. For real-time requirements, which differs from the requirements of conventional DSMSs, speed is not important unless a deadline will be missed. This implies that the query plan should be optimized in the worst case scenario.

Time predictability is also a requirement, which is the ability to exactly predict the worst case latency. Changing the query plan dynamically in DSMSs makes it difficult to satisfy the predictability requirement. In automotive embedded systems, hardware configurations and application software are determined at the design stage during automotive development, and the data processing installed in automotive embedded systems is also determined at that time. The maximum input-data volumes that should be processed are known by validating them beforehand. Therefore, the query plan should be optimized in the critical case in which the input-data volumes are maximized under the assumption. Equivalently, the DSMS can determine the query plan at the design stage.

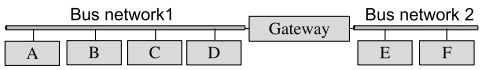


Fig. 4 Examples of in-vehicle networks and nodes.

4.2 Distributed Data Streams in In-vehicle Networks (C2)

The structure of networks and nodes generally differs among vehicle types; thus, distributed stream processing in an automotive system should be location transparent. A key issue is operator placement because it affects the efficiency of data processing and network usage. In other fields, many methods have been proposed for solving the operator placement problem by assuming overlay networks, as described in Section 2.4. These methods estimate network usage accurately because one of their main goals is its reduction [28].

Automotive data processing also requires reducing and estimating in-vehicle network usage. Increasing in-vehicle network usage not only increases the communication delay, but also makes it difficult to predict the communication delay by frequent transmission collisions. However, because in-vehicle networks have a complex hybrid network topology, which connects multiple bus networks with multiple gateways, it is difficult to decide the delivery route of streams through in-vehicle networks with previous operator placement methods. Furthermore, it is also difficult to estimate in-vehicle network usage. The following example indicates the requirements for the DSMS to accurately estimate delivery routes of streams and network usage in in-vehicle networks.
Example: Consider the following case: nodes A, B, C, and D and nodes E and F are connected via a gateway, as shown in **Fig. 4**. If A and B communicate, some streams are routed only through bus network1. In this case, network usage increases between C and D, although the network usage between E and F does not increase. In contrast, if D and E communicate, some streams are routed through bus network1, the gateway, and bus network2. In this case, network usage increases not only between C and D, but also between E and F.

4.3 Applying Real-time Scheduling (C3)

As explained in Section 3.1, our DSMS requires real-time scheduling. The data rate of the input-streams may change dynamically because the input-data rate from V2V communications changes during driving. For this reason, EDF scheduling is suitable as a real-time scheduling algorithm, which is the same approach used by the previous methods of Section 2.3.

In automotive data processing, the typical query plan is multiqueries because multiple applications share information such as vehicle position, as described in Section 3.2. Moreover, because automotive data processing may be distributed, data stream processing may also be distributed. Although scheduling is one of the main research topics in data stream processing, real-time scheduling has not been widely studied, and it does not meet our requirements. As described in Section 2.3, previous methods cannot be applied to distributed stream processing with multiqueries.

4.4 Enhancing Reliability by Sensor Data Fusion (C4)

Driver assistance systems require enhanced reliability of sensor data. Sensor fusion is a typical and effective method that meets this requirement. Kalman filters and particle filters are well-known algorithms employed for sensor fusion, which use previous output-results to improve positional accuracy or for tracking vehicles/pedestrians. However, few previous studies have implemented sensor fusion in a DSMS, and the implementation methods are not clear [19].

To develop various sensor fusion algorithms, it would be helpful if DSMSs could support common functions for sensor fusion. Such sensor fusion algorithms fuse sensor data with the same timestamp with previous output-results. As described in Section 3.1, because input-data may be out of order, tuples with older timestamp may arrive later. This means that it is necessary to manage previous output-results for a certain period of time. Most DSMSs provide windows for operators to manage temporal recent tuples. However, the intention is to manage the tuples before operators process rather than output results after the operators are processed. Therefore, it is necessary to support windows for the output-results of operators as a common function during sensor fusion.

4.5 Reusability of Query Descriptions (C5)

The physical structure and application programs are different by the type of vehicles. Queries that represent data processing are influenced by the type of vehicle because data processing in automotive systems is embedded in the product of the vehicle. Therefore, a reusable query description independent of the vehicle type is required.

5. AEDSMS: Automotive Embedded Data Stream Management System

To overcome the challenges discussed in Section 4, we introduce AEDSMS, which is a DSMS for automotive embedded data processing.

5.1 Design Principles

Figure 5 shows the overview of AEDSMS. AEDSMS provides users with high-level queries (HLQs) that can be location-transparent and modularized and precompiles HLQs into low-level queries (LLQs) using a code generation approach before they are installed in an automotive embedded system. When a vehicle is moving, AEDSMS dynamically processes the tuples to meet real-time and reliability requirements.

For HLQs, users can select a subset of standard Aurora operators; for example, Map, Filter, Union, and Join can be employed [33]. In addition, user-defined operators (UDOs) can be employed that allow users to implement processes using C++ classes. UDOs can be extended using class inheritance in C++. Moreover, AEDSMS is equipped with a mechanism for defining stream processing components (SPCs) by composing multiple operators and stores them in a library (SPC-Library). Users can retrieve them from the SPC-Library to describe queries and use operators and SPCs.

In the same manner as previous time-critical data stream processing on a single node [21], [22], [23], AEDSMS allows users to set End-to-End deadlines for output-streams of queries in distributed stream processing. AEDSMS executes the query plan based on EDF to meet End-to-End deadlines during runtime.

In AEDSMS, a sensor data fusion operator (SDFO) is available, which improves reliability by fusing streams that are ob-

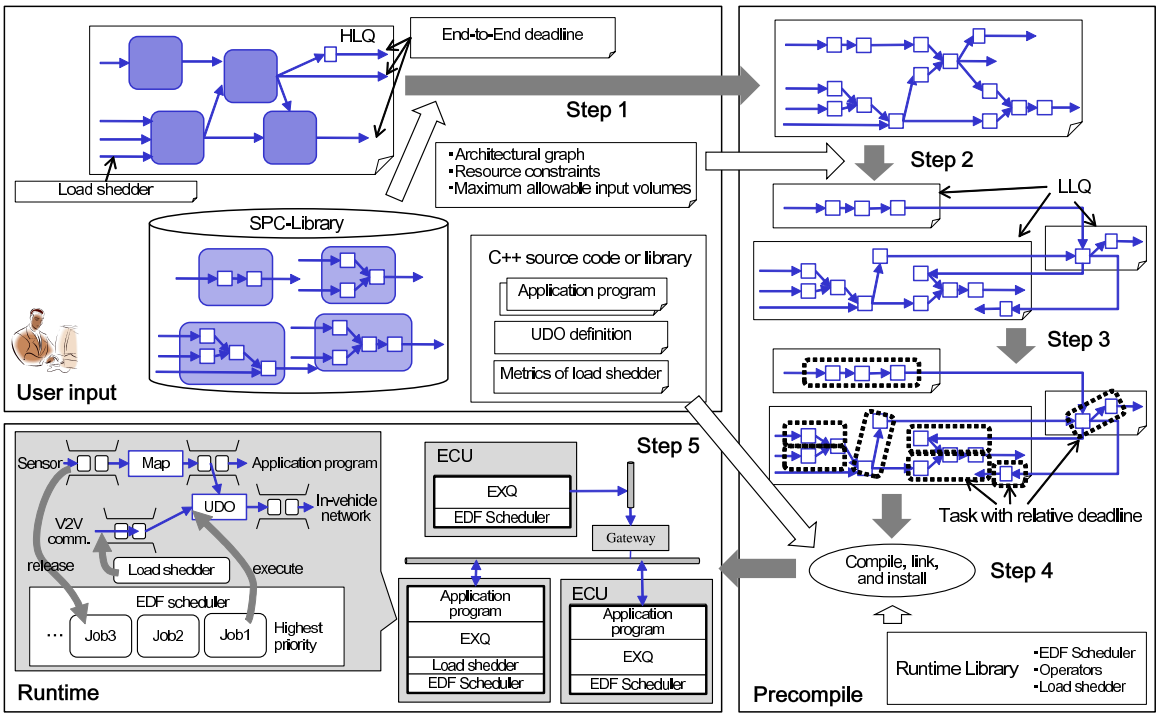


Fig. 5 Overview of AEDSMS.

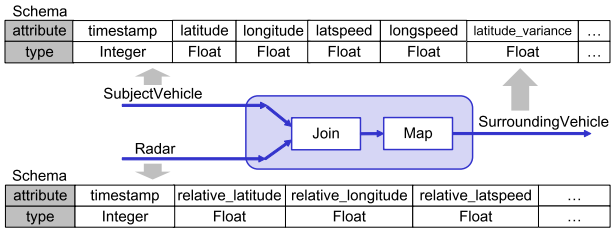


Fig. 6 Example of an SPC description.

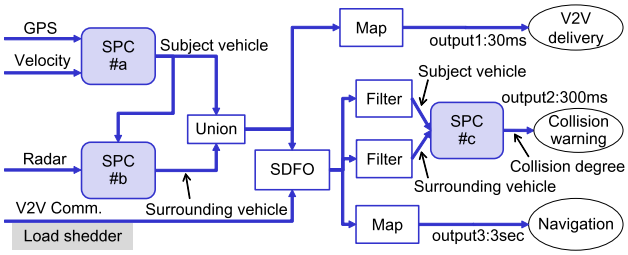


Fig. 7 Example of an HLQ.

tained redundantly from different sensors. Users can implement specific sensor fusion algorithms by inheriting the SDFO C++ class. The process that acquires inputs from onboard sensors is verified in advance at the design stage to ensure that it meets the deadline. However, a process that employs communication outside of the vehicle may miss a deadline due to large fluctuations in the volume of data and communication delays. Processes that use onboard sensors and communicate with the outside are redundant; thus, processes that employ communications are only used when they meet deadlines. Therefore, AEDSMS can provide reliable streams while meeting deadlines even if data obtained from outside of the vehicle is employed.

Queries installed in a target automotive embedded system should be customized for the specific vehicle type, whereas queries described by users should be independent of the physical structure. To address this issue, AEDSMS provides HLQs that are less dependent on the physical structure and converts HLQs into LLQs that are suited to the physical structure. Only query plans that have been generated as binary code are installed in the automotive embedded system. In the following, we describe the order in which the query plan is generated.

I. High-Level Queries (HLQs)

HLQs represent the highest level of abstraction for AEDSMS queries, and they are described by users. HLQs are location transparent, which means that they are independent of networks and nodes. HLQs are described using operators and SPCs.

II. Low-Level Queries (LLQs)

LLQs are customized according to the physical structure of the target, i.e., LLQs depend on networks and nodes. An LLQ does not include SPCs, and it only comprises operators. Each LLQ is generated for the node where operators are placed.

III. Executable Queries (EXQs)

An executable query (EXQ) is a query plan embedded in a target. Each EXQ is generated from an LLQ. Because each EXQ is a binary code, it cannot be easily edited by users.

5.2 Stream Processing Component (SPC)

In addition to defining queries, SPCs are defined as dataflow diagrams that comprise operators and connectivity, as shown in **Fig. 6**. An SPC has the same interface as operators; the inputs are streams, and the outputs are streams.

In the example in Fig. 6, an SPC produces the position and velocity of surrounding vehicles. It converts relative position/velocity from radar into an absolute value using the absolute position/velocity of the subject vehicle. The schema of the stream (Radar) obtained from radar has attributes such as relative position and velocity, confidence [*3], and the timestamp. The schema of the other stream (SubjectVehicle) input into the SPC has attributes such as position, velocity, confidence, and the timestamp. The schema of the stream output from the SPC has the same schema as SubjectVehicle. The SPC comprises two operators: Join and Map.

5.3 Example of an HLQ

An example of an HLQ is shown in **Fig. 7**. SPC#b is the SPC described in Fig. 6. This HLQ is a multiquery that shares part of the data processing and delivers three output-streams (output1, output2, and output3). A load shedder is positioned in the input-stream from V2V communications. In this HLQ, data processing using V2V communications is made redundant by processing with onboard sensors. Because of this, all of the input-data from onboard sensors are processed to meet their deadlines even if the load shedder drops the input-data from V2V communications.

Output1 delivers the stream about subject and surrounding vehicles, which is generated from the onboard sensors of the subject vehicle. This End-to-End deadline is the shortest at 30 ms because the output-stream is used as input for the driver assistance systems of surrounding vehicles. Output2 delivers the stream including the time to collision (TTC) as CollisionDegree, which is calculated from the streams related to the subject vehicle and other vehicles with the same timestamp. The End-to-End deadline is 300 ms according to ETSI because the output-stream is used to produce collision warnings [30]. Because output3 provides the results of sensor fusion from the onboard sensors and V2V communications for car navigation, real-time performance is not required, and the End-to-End deadline is 3 s, which is sufficiently long.

SPC#a outputs SubjectVehicle, which is obtained using GPS and velocity sensors. SPC#b outputs SurroundingVehicle, which is obtained using onboard sensors, as explained in Section 5.2. The Union operator unifies SubjectVehicle and SurroundingVehicle. The SDFO operator fuses the tuples obtained from the onboard sensors and V2V communications. Map operators format the tuples for the purpose of V2V delivery and navigation. SPC#c outputs CollisionDegree based on the outputs of the two Filter operators, which extract streams related to the subject and surrounding vehicles.

5.4 Design Flow of AEDSMS

The design flow of AEDSMS comprises five steps, as outlined

[*3] Confidence is represented as a variance in the same manner as Ref. [10] to simplify the computation.

in Fig. 5. Steps 1–4 statically precompile queries when the software is developed. Step 5 is only processed during runtime on the target automotive embedded system.

Step 1. SPC expansion

This step expands the SPCs in an HLQ using the definition of SPCs in the SPC-Library, and it embeds the operators and connectivity, which define SPCs, into the HLQ. As a result, the expanded HLQ does not include SPCs.

Step 2. Operator placement

This step allocates the expanded HLQ to the nodes and in-vehicle networks of the target. Unlike previous distributed stream processing [28], [29], AEDSMS converts a physical structure into an architectural graph, which is an approach that is widely used to explore the design space of embedded systems [36]. Next, AEDSMS allocates the queries to the architectural graph, as described in Section 5.5. The previous methods of operator placement often use the input-data rate at runtime. In contrast, AEDSMS optimizes operator placement by using the maximum input-data rate in the assumed worst-case; in this case, meeting deadlines is more important than speed. As a result, previous methods of distributed stream processing can be applied in our field, and LLQs can be generated.

Step 3. Task definition

This step defines tasks from the LLQs and assigns the appropriate relative deadline, which is the modified End-to-End deadline, for each task. This method is described in Section 5.6. Because the release-timing of each task is when tuples are inserted into tasks, the information required by EDF scheduling at runtime can be obtained in this step.

Step 4. Binary generation

This step statically converts LLQs into source code and then selects the minimum number of necessary modules from a Runtime Library, which contains functions to run the DSMS in automotive embedded systems. Next, this step links the Runtime Library, EXQ, and application programs (if any). As a result, executable files are generated.

Step 5. Execution on an automotive embedded system

The executable files generated during runtime in Step 4 are installed on nodes that execute stream processing. The instance of each task has an absolute deadline by adding the relative deadline to the timestamp of the processed tuples. On each node, the scheduler executes the operator that inputs the tuple with the earliest absolute deadline at the highest priority, based on the EDF policy. Application programs can use output-streams via function calls or callback functions.

5.5 Operator Placement using an Architectural Graph

Figure 8 outlines how the operator placement problem is solved with an architectural graph. In an architectural graph, a vertex is a node (ECU), network (bus), or gateway. In particular, a vertex that comprises a network or gateway is a virtual vertex. The architectural graph has an edge between vertices if and only if a gateway or an ECU connects with a network.

The following discussion assumes that the physical structure and application programs are set in a manner similar to that in Fig. 8 (1). A query is outlined in Fig. 8 (2). The architectural

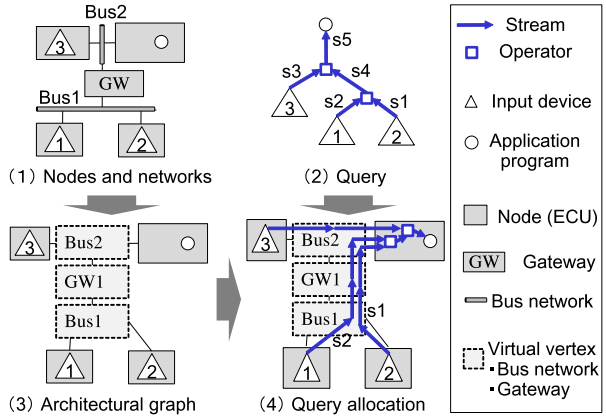


Fig. 8 Query allocation using an architectural graph.

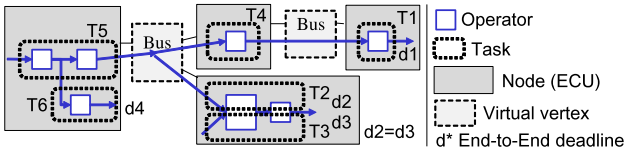


Fig. 9 Allocation method for tasks and their relative deadlines.

graph converted from Fig. 8 (1) is shown in Fig. 8 (3). After conversion into an architectural graph and by satisfying the following constraints, the queries can be allocated using previous methods of operator placement, such as those in [29].

- An operator cannot be mapped to a virtual vertex, which represents a network or a gateway [*4].
- A stream can be mapped between different nodes only if the architectural graph has an edge between the vertices.
- The usage of each network can be estimated as the sum of the data volume in the streams that are input to the virtual vertex corresponding to the network. For example, the network usage of Bus1 can be estimated as the sum of the input-data volume of the two streams (s1 and s2) in Fig. 8 (4).

Using the above method, the delivery route of each stream can be correctly represented in in-vehicle networks and network usage can be accurately estimated.

5.6 Task Definition for Real-time Scheduling of DSMSs

It is necessary to define tasks and specify their relative deadlines to apply EDF to DSMS scheduling. AEDSMS defines each task as a sequence of operators, where the execution order is determined statically at each node, as shown in **Fig. 9**. If there exists a dependency of execution order (i.e., precedence constraints) among tasks, EDF* [37] can be applied to resolve the dependency. EDF* calculates the relative deadline d of a precedence task τ from the computation time c_i and the relative deadline d_i of each subsequent task τ_i; that is,

$$d = \min\{d_i - c_i; i = 1, 2, \cdots, n\}. \quad (1)$$

First, the relative deadlines of tasks without subsequent tasks are initialized as End-to-End deadlines. Next, by recursively applying Eq. (1) from subsequent tasks to a precedence task, it can be shown that all tasks can specify appropriate relative deadlines and

[*4] A stream can be allocated to virtual vertices.

remove precedence constraints [37]. AEDSMS considers communication delays in the same manner as the computation time of an operator; it uses the following equation, which is an extension of Eq. (1):

$$d = \min\{d_i - c_i - c_{\tau\to\tau_i}; i = 1, 2, \cdots, n\}, \quad (2)$$

where $c_{\tau\to\tau_i}$ is the communication delay between τ and τ_i.

The following discussion assumes the example in Fig. 9. After the relative deadline of T4 is derived from the End-to-End deadline of T1 using Eq. (2), the relative deadline of T5 is derived from the relative deadline of T4 and End-to-End deadlines of T2 and T3 by Eq. (2). Therefore, AEDSMS assigns relative deadlines even if a multiquery is distributed among multiple nodes. As a result, real-time scheduling can be applied based on EDF in AEDSMS.

Previous load shedders [24], [25], [26] drop or filter tuples from streams to avoid overload or missing deadlines. Because the input-data from onboard sensors cannot be dropped in safety critical applications, AEDSMS can set load shedders to specific input-streams, which is input-data from outside the vehicle in most cases. Users can write a value function to determine the filtering conditions for each load shedder. The load shedder filters tuples of lower value so that the maximum input volume is not exceeded, which is predetermined at the design stage. Setting the load shedder to the input-stream from V2V communications makes it possible to process all of the input-data from onboard sensors without filtering while meeting deadlines, even if the number of surrounding vehicles increases.

5.7 Sensor Data Fusion Operator (SDFO)

The SDFO is the operator that allows users to implement sensor fusion algorithms efficiently by stream processing and is implemented with a specific window structure via class inheritance from UDO. The window structure of the SDFO is outlined in **Fig. 10**. In the automotive field, sensor fusion requires windows to handle out-of-order inputs, as described in Section 4.4. Therefore, in addition to the front window, where traditional DSMS operators such as Join and Aggregate hold the previously input tuples for a certain period of time, SDFO has a rear window, which holds previous output tuples for a certain period of time.

Most sensor fusion algorithms fuse the same observed objects and do not fuse different observed objects. Therefore, both the front window and rear window are partitioned for observed objects. Moreover, because sensor fusion usually enhances the effects by fusing diverse sensors, the combination of fused sensor data should be distinguished by observing sensors. Therefore, the front window is also partitioned for observing sensors.

The SDFO fuses the tuples held in the front window when all of the inputs arrive. However, the latency is not guaranteed if the SDFO uses input-data obtained from outside the vehicle. In this case, the SDFO imposes a time limit and only processes tuples that have arrived at the front window when the timeout is expired.

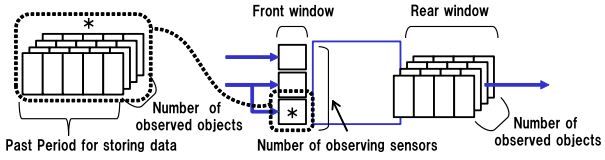


Fig. 10 Structure of the windows employed for SDFO.

6. Feasibility Study

We evaluated the performance of AEDSMS during runtime and confirmed its feasibility.

6.1 Evaluation Method

The application scenario is the example in Section 3.1.1, and the HLQs are the same as those in Section 5.3 (Fig. 7). In the automotive field, the volume of input-data from V2V communications is large and fluctuates. In order to confirm the feasibility of AEDSMS when the input-data volume is maximized, we performed an evaluation where information is received at a maximum rate of approximately 1000 vehicles per second from V2V communications, which matches the ETSI specification [30]. We used the network simulator Scenargie [*5] in this evaluation because it can handle V2V communications from a large number of vehicles, as well as facilitate an accurate simulation of their radio properties, radio field strength, and the mobility of each vehicle. We used roads with a grid layout, specifically, the Manhattan mobility model, which includes obstacles such as buildings, thereby allowing us to evaluate collision warnings at intersections with poor visibility, while still satisfying the maximum input-data volume specified by ETSI.

Figure 11 shows the roads and initial vehicle positions. Each vehicle moves in the direction of the arrow at a speed of 60 km/h, and the distance between all intersections is equal to 100 m. The number of vehicles is maximized at the two intersections marked with circles in Fig. 11. **Figure 12** shows the number of input-data obtained from V2V communications per unit time (1 s). These simulation parameters ensure that the input volume is maximized and satisfies the ETSI specifications at the two intersections.

Both V2V communications and radar were assumed to provide

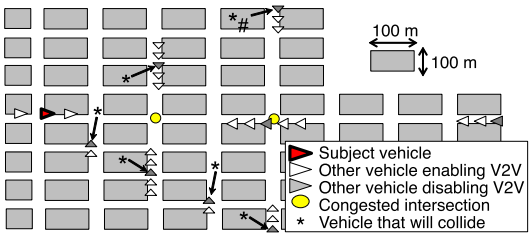


Fig. 11 Initial positions of vehicles in the Manhattan model.

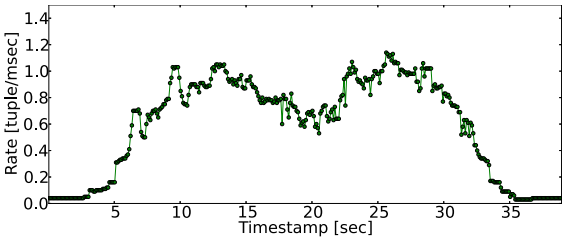
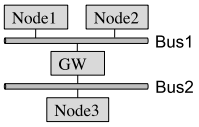


Fig. 12 Input rate from V2V communications.

[*5] Scenargie: https://www.spacetime-eng.com/en/labSimulator.html



Node (ECU)	Input devices	Application Programs
Node1	GPS, Velocity sensor	-
Node2	Radar	-
Node3	V2V communication	all

Fig. 13 Physical structure.

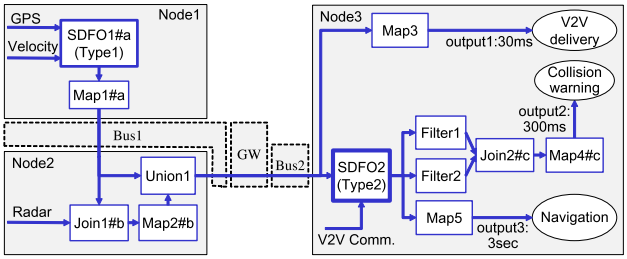


Fig. 14 LLQs and their mappings.

Table 1 Task definition on each node.

Task	Node	Operators	Relative deadline [ms]
T_1	Node1	SDFO1, Map1	$d_1 := \min\{d_2 - c_2, d_3 - c_3\} - c_{\text{Bus1}}$
T_2	Node2	Union1	$d_2 := \min\{d_4 - c_4, d_5 - c_5\} - 2c_{\text{Bus1}} - c_{\text{GW}} - c_{\text{Bus2}}$
T_3	Node2	Join1, Map2, Union1	$d_3 := \min\{d_4 - c_4, d_5 - c_5\} - 2c_{\text{Bus1}} - c_{\text{GW}} - c_{\text{Bus2}}$
T_4	Node3	SDFO2	$d_4 := \min\{d_6 - c_6, d_7 - c_7, d_8 - c_8\}$
T_5	Node3	Map3	$d_5 := 30$
T_6	Node3	Filter1, Join2, Map4	$d_6 := 300$
T_7	Node3	Filter2, Join2, Map4	$d_7 := 300$
T_8	Node3	Map5	$d_8 := 3000$

input-data at 100 ms cycles, and their range was approximately 200 m. Radar was not refracted by obstacles because we used a high frequency of 70 GHz or greater. However, the V2V communications occurred at 700 MHz, the frequency band defined in Japan, which means that it is refracted to some extent even if there are obstacles. Because sensor noise cannot be simulated in Scenargie, we experimentally injected artificial noise based on sensor specifications [*6] [*7].

Figure 13 shows the physical structure. The network is comprised of one gateway and three nodes in which the previous methods of operator placement have difficulties allocating a query plan. The specification for each node is an 800 MHz CPU with 512 MB of RAM running on a Linux OS (Fedora10). **Figure 14** shows the physical structure and allocation of the LLQs. Operators #a–#c correspond to the SPCs to which the operators belong. The bus networks assume Controller Area Networks (CANs), and the communication delay is calculated using formular (1) in [38].

Table 1 shows the task definition on each node. Each task is comprised of the third column (Operators), which corresponds to operators in Fig. 14. For some i, d_i is the relative deadline of task T_i, and c_i is the computation time. c_{Bus1}, c_{Bus2}, and c_{GW} are the communication delays of Bus1, Bus2, and GW, respectively.

6.2 Basic Performance of SDFOs

Before evaluating the application scenario described in Section 6.1, we evaluated the basic performance of a single SDFO. An SDFO is a specific operator that was newly developed for AEDSMS and takes more computation time than the other operators. We implemented two types of Kalman filters as SDFOs, which were the same as the SDFOs in the LLQs shown in Fig. 14.

Type1: Subject vehicle position estimation

This type estimates the subject vehicle's position and velocity by observing GPS and velocity sensors, as described in Ref. [39]. This type of sensor fusion is used frequently in commercial cars.

Type2: Fusion of multiple sensor data from vehicles

This type estimates the subject vehicle's position by fusing the position of the subject and surrounding vehicles based on onboard sensors and V2V communications, in the same way as Ref. [40]. The fusion number varies according to the number of vehicles communicating via V2V and the number of vehicles observed.

Table 2 shows the latency of each SDFO when the fusion number (i.e., the number of sensor fusions) is 1, where the statistics are calculated using over 10,000 samples. The average latency is on the order of ms for Type1 even in the worst case conditions, and the End-to-End deadline is 30 ms for Type1. The latency is on the order of 10 ms for Type2, and the End-to-End deadline is 300 ms for Type2. The standard deviations are also small. Thus, the latency is predictable and acceptable when the fusion number was 1.

Table 2 Latency in the case of single sensor fusion.

Type	Average	Standard deviation
1	1.84 ms	86.2 μs
2	11.0 ms	92.2 μs

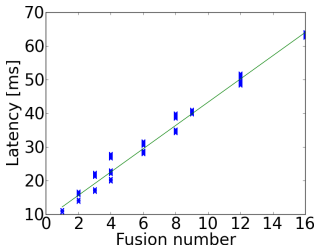


Fig. 15 Relationship between the fusion number and latency.

Figure 15 shows the change in the latency as the fusion number varies with Type2. In addition, Fig. 15 shows the results of the linear regression analysis, where the fusion number is used as an explanatory variable to predict the latency. The results show that the error is sufficiently small because the average relative error between the measured value and the predicted value is 1.64 ms. Therefore, the computation time of each SDFO can be predicted from the fusion number. Furthermore, the computation time is predictable in the worst case because the maximum fusion number can be estimated from the input volume of tuples in the SDFO.

6.3 Effect of Introducing the EDF Scheduler

We evaluated the effectiveness of applying real-time scheduling based on EDF in AEDSMS and compared it with an FIFO scheduler. **Figure 16** shows the changes in the maximum latency for output1, which is the output-stream with the shortest End-to-End deadline. The latency is measured from the time between when the input-data is read from the sensors to when the tuples are inserted into the output-streams. Each plot and error bar are the average and standard deviation of the maximum latency based on 100 trials, respectively.

[*6] DGPS: http://www.u-blox.com/images/downloads/Product_Docs/GLONASS-HW-Design_ApplicationNote_(GPS.G6-CS-10005).pdf
[*7] Radar: http://www.abott-mf.com/images/pdf/IbeoLUX2010.pdf

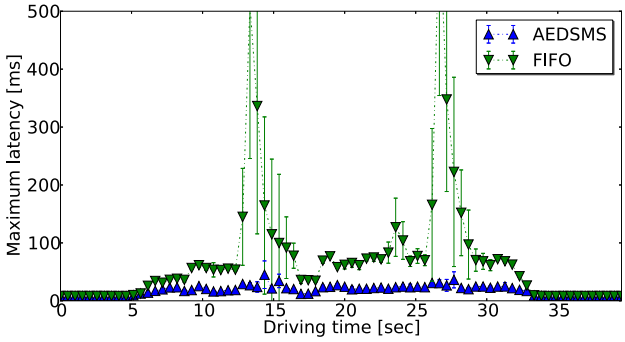


Fig. 16 Changes in maximum latency of output1 with driving time.

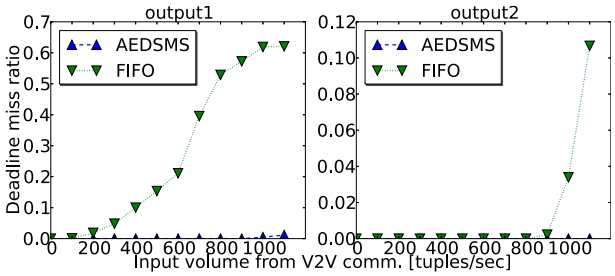


Fig. 17 Deadline miss ratio of output1 and output2.

The proposed method reduces the maximum latency of output1 by 61%. This is because AEDSMS processes the tuples with earlier deadlines first using EDF and postpones the tuples with later deadlines. In particular, because the computation time of SDFO2 is long with large input-data volumes from V2V communications, the difference of the execution order drastically affects the difference in the maximum latency, as shown in Fig. 16.

Figure 17 shows the rates of tuples with missed deadlines (deadline miss ratio) for output-streams of output1 and output2. The deadline miss ratio of output3 is always 0 in each method. The horizontal axis represents the maximum volume of input-data that the load shedder passes per 1 s in the input-stream derived from V2V communications.

Figure 17 shows that real-time scheduling and load shedding both reduce the rates of missed deadlines. However, the effect of real-time scheduling based on EDF is significantly better. In the cases where EDF is applied, it is possible to process 800 sets of input-data per 1 s. In contrast, the cases without EDF (i.e., FIFO) can only process 100 sets of input-data per 1 s. This shows that the proposed method processes approximately eight times as much input-data volume, while still meeting deadlines. This can be attributed to the fact that the maximum latencies of output1, where the End-to-End deadline is the shortest, are reduced by EDF.

With or without EDF, the maximum numbers of the input-data volume that could be processed in the following evaluation were 800 and 100 per 1 s, respectively.

6.4 Effect on Time to Collision

We evaluated the effects of vehicle collisions based on the time to collision (TTC), which is the remaining time when the subject vehicle would have collided with the surrounding vehicles, obtained from output2. In this scenario, without braking, the subject vehicle would have collided with the six vehicles that are marked

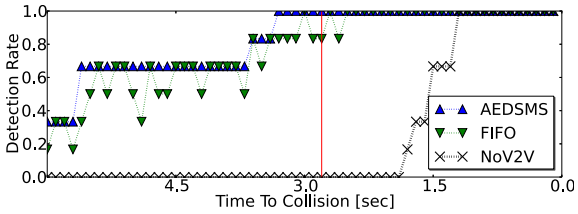


Fig. 18 Changes in the detection rate with TTC from output2 in a trial.

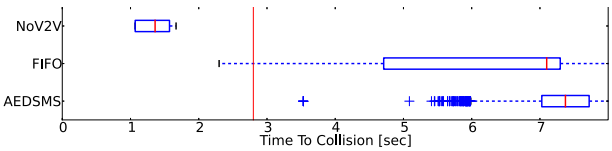


Fig. 19 Box plot of TTC at the first detection of each vehicle∗ from output2.

by asterisks (∗) in Fig. 11. We estimated that the time required to stop to be 2.8 s on dry roads based on the total stopping distance, where the vehicle is traveling at a speed of 60 km/h, based on [41]. We also assumed that the subject vehicle began to stop after detecting the vehicles∗. This means that the subject vehicle collides with each vehicle∗ if and only if the TTC is shorter than 2.8 s for vehicle∗.

First, **Fig. 18** shows how many of the six vehicles∗ could be detected with a specific TTC, as the detection rate, in a trial. A red line is drawn at 2.8 s, which is the threshold of collision avoidance for each vehicle∗. We denote the case where V2V communication is not used by NoV2V.

If V2V communication is not used (i.e., NoV2V), the subject vehicle collides with all of the vehicles∗ because the detection rate is 0 when TTC is greater than 2.8 s. This shows that it is difficult to avoid vehicle crashes using only onboard sensors. However, all of the vehicles∗ were detected before colliding because the detection rate was 1 when TTC was greater than 0. This shows that collision warnings are processed only by the onboard sensors with the time limit feature of the SDFO. Figure 18 also shows that AEDSMS using EDF achieves a better detection rate than that using FIFO when the TTC is long. This is because EDF allows more tuples from V2V communications to be processed while still meeting deadlines.

Next, we estimated the vehicle crash rate as the proportion of cases where the subject vehicle collided with the vehicles∗. In this experiment, trials similar to Fig. 18 were repeated 100 times. In this case, the vehicle crash rate was defined by

$$\frac{\sum_{1,2,\cdots,100} M_t}{6 \times 100}, \tag{3}$$

where M_t is the number of vehicles∗ that collided with the subject vehicle in the t-th trial.

Figure 19 shows the TTC when each vehicle∗ is detected at the first time in all of the trials. A red line is drawn at the threshold of collision avoidance (2.8 s). The shortest TTC (worse case) of AEDSMS is 3.5 s, whereas that of FIFO or NoV2V is 2.3 or 1.1 s, respectively. In particular, AEDSMS avoids vehicle crashes in all of the trials, whereas using FIFO or NoV2V contributed to vehicle crashes. Thus, we found that real-time scheduling of AEDSMS has a positive effect on advance driver assistance systems.

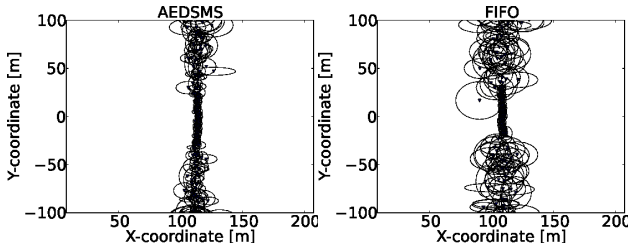


Fig. 20 Tracking of vehicle# in a trial with confidence from output3.

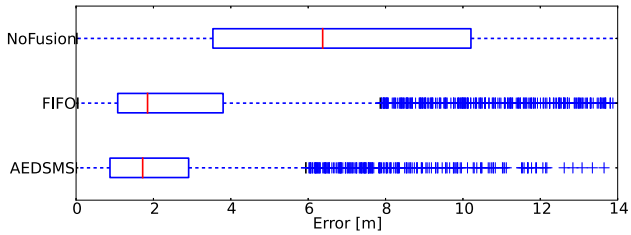


Fig. 21 Box plot of positional error for all of the vehicles from output3.

6.5 Impact on Sensor Data Fusion

First, by focusing on vehicle# in Fig. 11, we confirmed the effect on the positional accuracy of vehicles based on sensor fusion in the application scenario. **Figure 20** shows the vehicle# position and confidence that are included in the surrounding vehicles' information delivered from output3. The circle represents the symmetrical 95% confidence interval, which is used in ETSI specifications [4], [5].

Figure 20 shows that AEDSMS using EDF improves the positional accuracy and confidence about the vehicle#, compared to using FIFO. This is because tuples can be fused for a higher number of vehicles with the SDFO (Type2) when using EDF, which allows more tuples to be processed while still meeting deadlines.

Next, we evaluated the positional accuracy between the true and observed values for all of the vehicular information that was delivered from output3. **Figure 21** shows the error between the exact position and the observed position based on output3 for all of the vehicles in this scenario.

With and without sensor fusion, the average errors were 2.32 and 7.30 m, respectively, i.e., 4.98 m less than when using sensor fusion. Therefore, sensor fusion is effective for improving confidence with AEDSMS. With EDF and FIFO, the average errors were 2.32 and 3.37 m, i.e., 1.05 m less than when applying EDF. Therefore, we demonstrated that real-time scheduling of AEDSMS is effective for improving the positional accuracy of automotive data.

7. Discussion

In this section, we summarize how AEDSMS addresses the challenges described in Section 4.

Time-criticality During Runtime (C1)
AEDSMS achieves time-criticality using the architecture based on precompilation strategies, as shown in Fig. 5. This is because it requires to optimize query plans in the worst-case. The other reason is because the maximum input-data volumes, automotive data processing, and the physical structures are specified during the design stage. Moreover, real-time scheduling based on EDF is applied to achieve the requirement, which is explained later (i.e., C3). In addition, we evaluated the performance of the SDFOs, which require more computation time than standard operators, and confirmed that the worst-case latency for the SDFOs are predictable in Section 6.2.

Distributed Data Streams in In-vehicle Networks (C2)
Previously proposed distributed stream processing systems do not determine the route of streams in in-vehicle networks, which consist of bus networks and gateways, when allocating a query plan. In addition, they do not provide accurate measures of usage in in-vehicle networks. Thus, AEDSMS solves these problems by applying an architectural graph, which is used widely in embedded systems (especially, design space exploration), as described in Section 5.5.

Applying Real-time Scheduling (C3)
Previous methods for real-time scheduling of stream processing could not be applied directly to AEDSMS because they are too limited to handle multiqueries in distributed stream processing. Thus, we proposed the definition of tasks and their relative deadlines using EDF*, which is employed in real-time scheduling. We applied EDF-based real-time scheduling to AEDSMS, as described in Section 5.6. We confirmed that EDF-based real-time scheduling reduced the maximum latency, the deadline miss ratio, and the vehicle crash rate, and improved the positional accuracy of vehicles, as described in Sections 6.3, 6.4, and 6.5.

Enhancing Reliability by Sensor Data Fusion (C4)
AEDSMS enhanced reliability by using the sensor fusion operator, SFDO, as described in Section 5.7. We implemented two types of Kalman filters and confirmed the basic performance and improved positional accuracy of vehicles in Sections 6.2 and 6.5.

Reusability of Query Descriptions (C5)
In AEDSMS, high-level query descriptions are produced using HLQs, which provide a method for modularizing stream processing by SPC and positional transparency; thus, these descriptions require fewer modifications if the physical structure changes.

8. Conclusions

Because advanced driver assistance systems with automatic driving technologies are being utilized in vehicles, software platforms for managing data integration based on DSMSs are becoming increasingly important. In this paper, we introduced AEDSMS, i.e., a DSMS for data processing in automotive embedded systems, and we clarified five important challenges of this study as well as the solutions to these challenges. The feasibility was evaluated by a vehicle collision warning application, which is crucial for supporting driving safety. The evaluation was simulated according to the C-ITS specification where the input-data volume was maximized. As a result, the performance during runtime was measured, and the effectiveness of AEDSMS was confirmed and demonstrated. This integrated technology can also be applied to other application areas such as avionics and robotics, which must address the same challenges. In the future, we will introduce AEDSMS into actual cars.

Acknowledgments We would like to thank the consortium partners for supporting the development of an AEDSMS. These partners included seven companies (Toyota Motor Corporation, Hitachi Ltd., Hitachi Automotive Systems Ltd., Hitachi Solutions

Ltd., NEC Informatec Systems Ltd., Toyota Central R&D Labs, and Denso Corporation) and three organizations (National Institute for Land and Infrastructure Management, Japan Digital Road Map Association, and Chubu Bureau of Economy, Trade, and Industry). This research was supported in part by a Grant-in-Aid for Scientific Research (25240007) and the Strategic Information and Communications R&D Promotion Programme (SCOPE) (12180615) by the Ministry of Internal Affairs and Communications, Japan.

References

[1] Jones, W.D.: Keeping cars from crashing, *IEEE Spectr.*, Vol.38, No.9, pp.40–45 (2001).

[2] Buehler, M., Iagnemma, K. and Singh, S.: *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic*, Springer, 1st edition (2009).

[3] Erico, G.: How Google's Self-Driving Car Works, IEEE Spectrum (online), available from ⟨http://spectrum.ieee.org/automaton/robotics/artificial-intelligence/how-google-self-driving-car-works⟩ (accessed 2014-11-21).

[4] ETSI: Intelligent Transport Systems; Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service (2011).

[5] ETSI: Intelligent Transport Systems; Users and applications requirements; Part 2: Applications and facilities layer common data dictionary (2013).

[6] Yamada, M., Sato, K. and Takada, H.: Implementation and Evaluation of Data Management Methods for Vehicle Control Systems, *Vehicular Technology Conference* (*VTC Fall*), pp.1–5 (2011).

[7] Yamaguchi, A., Nakamoto, Y., Sato, K., Ishikawa, Y., Watanabe, Y., Honda, S. and Takada, H.: AEDSMS: Automotive Embedded Data Stream Management System, *Proc. IEEE 31st International Conference on Data Engineering*, pp.1292–1303 (2015).

[8] Yamaguchi, A., Watanabe, Y., Sato, K., Nakamoto, Y. and Takada, H.: Real-time Scheduling Method for Automotive Embedded Data Stream Processing (in Japanese), *IPSJ Trans. Database*, Vol.8, No.2, pp.1–17 (2015).

[9] Freund, U.: Mulit-level System Integration Based on AUTOSAR, *Proc. 30th International Conference on Software Engineering*, pp.581–582 (2008).

[10] Vivo, G., Dalmasso, P. and Vernacchia, F.: The European Integrated Project SAFESPOT - How ADAS applications co-operate for the driving safety, *Proc. IEEE Conference on Intelligent Transportation Systems*, pp.624–629 (2007).

[11] Nystrom, D., Tesanovic, A., Norstrom, C., Hansson, J. and Bankestad, N.-E.: Data management issues in vehicle control systems: A case study, *Proc. Euromicro Conference on Real-Time Systems*, pp.249–256 (2002).

[12] Nystrom, D., Tesanovic, A., Tesanovic, R., Nolin, M., Norstrom, C. and Hansson, J.: COMET: A Component-Based Real-Time Database for Automotive Systems, *Proc. Workshop on Software Engineering for Automotive Systems*, pp.1–8 (2004).

[13] Cugola, G. and Margara, A.: Processing Flows of Information: From Data Stream to Complex Event Processing, *ACM Comput. Surv.*, Vol.44, No.3, pp.15:1–15:62 (2012).

[14] Akidau, T., Balikov, A., Bekiroğlu, K., Chernyak, S., Haberman, J., Lax, R., McVeety, S., Mills, D., Nordstrom, P. and Whittle, S.: MillWheel: Fault-tolerant Stream Processing at Internet Scale, *Proc. VLDB Endow.*, Vol.6, No.11, pp.1033–1044 (2013).

[15] Lam, W., Liu, L., Prasad, S., Rajaraman, A., Vacheri, Z. and Doan, A.: Muppet: MapReduce-style Processing of Fast Data, *PVLDB*, Vol.5, No.12, pp.1814–1825 (2012).

[16] Schweppe, H., Member, A.Z. and Grill, D.: Flexible On-Board Stream Processing for Automotive Sensor Data, *IEEE Trans. Industrial Informatics*, Vol.6, No.1, pp.81–92 (2010).

[17] Kargupta, H., Bhargava, R., Liu, K., Powers, M., Blair, P., Bushra, S., Dull, J., Sarkar, K., Klein, M., Vasa, M. and Handy, D.: VEDAS: A Mobile and Distributed Data Stream Mining System for Real-Time Vehicle Monitoring, *Proc. SIAM International Conference on Data Mining*, pp.300–311 (2004).

[18] Kargupta, H., Sarkar, K. and Gilligan, M.: MineFleet®: An overview of a widely adopted distributed vehicle performance data mining system, *Proc. 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp.37–46 (2010).

[19] Bolles, A., Appelrath, H., Geesen, D., Grawunder, M., Hannibal, M., Jacobi, J., Koster, F. and Nicklas, D.: StreamCars: A new flexible architecture for driver assistance systems, *Intelligent Vehicles Symposium*, pp.252–257 (2012).

[20] Buttazzo, G.C.: *Hard Real-time Computing Systems: Predictable Scheduling Algorithms And Applications* (*Real-Time Systems Series*), Springer-Verlag, Santa Clara, CA, USA (2004).

[21] Ou, Z., Yu, G., Yu, Y., Wu, S., Yang, X. and Deng, Q.: Tick Scheduling : A Deadline Based Optimal Task Scheduling Approach for Real-Time Data, *Proc. Advances in Web-Age Information Management*, pp.725–730 (2005).

[22] Wei, Y., Son, S. and Stankovic, J.: RTSTREAM: Real-time query processing for data streams, *Proc. 9th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*, pp.141–150 (2006).

[23] Li, X., Jia, Z., Ma, L., Zhang, R. and Wang, H.: Earliest Deadline Scheduling for Continuous Queries over Data Streams, *Proc. International Conference on Embedded Software and Systems*, pp.57–64 (2009).

[24] Wei, Y., Prasad, V., Son, S. and Stankovic, J.: Prediction-Based QoS Management for Real-Time Data Streams, *Proc. IEEE Real-Time Systems Symposium*, pp.344–358 (2006).

[25] Li, X., Ma, L., Li, K., Wang, K. and an Wang, H.: Adaptive Load Management over Real-Time Data Streams, *Proc. 4th International Conference on Fuzzy Systems and Knowledge Discovery*, pp.719–725 (2007).

[26] Li, X., Jia, Z., Ma, L., Qin, Z. and Wang, H.: QoS-Aware Scheduling for Mixed Real-Time Queries over Data Streams, *Proc. 15th IEEE International Conference of the Embedded and Real-Time Computing Systems and Applications*, pp.145–154 (2009).

[27] Srivastava, U., Munagala, K. and Widom, J.: Operator Placement for In-network Stream Query Processing, *Proc. 24th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, ACM, pp.250–258 (2005).

[28] Lakshmanan, G.T., Li, Y. and Strom, R.: Placement Strategies for Internet-Scale Data Stream Systems, *IEEE Internet Computing*, Vol.12, No.6, pp.50–60 (2008).

[29] Kalyvianaki, E., Wiesemann, W., Vu, Q.H., Kuhn, D. and Pietzuch, P.: SQPR: Stream Query Planning with Reuse, *Proc. 2011 IEEE 27th International Conference on Data Engineering*, IEEE Computer Society, pp.840–851 (2011).

[30] ETSI: Intelligent Transport Systems; V2X Applications; Part 3: Longitudinal Collision Risk Warning application requirements specification (2013).

[31] FHWA: Intersection Safety Issue Briefs (2004).

[32] Casparsson, L., Rajnak, A., Tindell, K. and Malmberg, P.: Volcano a revolution in on-board communications (1998).

[33] Abadi, D.J., Ahmad, Y., Balazinska, M., Hwang, J.-h., Lindner, W., Maskey, A.S., Rasin, A., Ryvkina, E., Tatbul, N., Xing, Y. and Zdonik, S.: Aurora: A new model and architecture for data stream management, *The VLDB Journal*, Vol.12, No.2, pp.120–139 (2003).

[34] Arvind, D.P., Arasu, A., Babcock, B., Babu, S., Datar, M., Ito, K., Nishizawa, I., Rosenstein, J. and Widom, J.: Stream: The Stanford stream data manager, *IEEE Data Engineering Bulletin*, Vol.26, pp.19–26 (2003).

[35] Cranor, C., Johnson, T., Spataschek, O. and Shkapenyuk, V.: Gigascope: A Stream Database for Network Applications, *Proc. 2003 ACM SIGMOD International Conference on Management of Data*, ACM, pp.647–651 (2003).

[36] Oliveira, M.F.S., Nascimento, F.A., Mueller, W. and Wagner, F.R.: Design Space Abstraction and Metamodeling for Embedded Systems Design Space Exploration, *Proc. 7th International Workshop on Model-Based Methodologies for Pervasive and Embedded Software*, pp.29–36 (2010).

[37] Chetto, H., Silly, M. and Bouchentouf, T.: Dynamic Scheduling of Real-time Tasks Under Precedence Constraints, *Real-Time Syst.*, Vol.2, No.3, pp.181–194 (1990).

[38] Navet, N. and Simonot-Lion, F.: Trends in Automotive Communication Systems, *Networked Embedded Systems*, CRC Press, p.13 (2009).

[39] Ammoun, S. and Nashashibi, F.: Real time trajectory prediction for collision risk estimation between vehicles, *Proc. IEEE International Conference on Intelligent Computer Communication and Processing*, pp.417–422 (2009).

[40] Smith, R.C. and Cheeseman, P.: On the Representation and Estimation of Spatial Uncertainly, *Int. J. Rob. Res.*, Vol.5, No.4, pp.56–68 (1986).

[41] South Australia: *The Driver's Handbook*, Department for Transport, Energy and Infrastructure (2005).

Akihiro Yamaguchi is currently a student of Nagoya University, and works for Toshiba Corporation. He received his M.E. degree in Kobe University in 2006. From 2006, he joined Toshiba Corporation. From 2011 to 2014, he was a researcher at the Center for Embedded Computing Systems (NCES), Nagoya University. His research interests include data stream processing and real-time scheduling.

Yousuke Watanabe received his M.E. and Dr.E. degrees in University of Tsukuba in 2003 and 2006. In 2014, he joined Institute of Innovation for Future Society, Nagoya University, as a designated associate professor. His research interests include data stream processing and information integration. He is a member of the Database Society of Japan, IEICE, and ACM.

Kenya Sato is a professor of Doshisha University, Kyoto, Japan, where he has been since 2004. He also currently leads the Mobility Research Center of the university, and serves as a designated professor of Nagoya University. He received his BE and ME degree from Osaka University, and also received the Ph.D. degree from Nara Institute of Science and Technology. During 1991-1994 Dr. Sato was a visiting researcher at Computer Science Department, Stanford University, and he was a chief technologies of Automotive Multimedia Interface Collaboration in Michigan, U.S. in 2001-2003. His research interests include network architecture, distributed systems, and ITS. Professor Sato is a member of Japan head of delegation to ISO ITS technical committee, and a member of Advance Safety Vehicle study committee in Japan.

Yukikazu Nakamoto received his M.E. and Ph.D. degrees from Osaka University in 1982 and 2000, respectively. From 1982 to 2004, he worked for NEC Corporation. In 2004, he joined the University of Hyogo and is currently a Professor of Graduate School of Applied Informatics. From 1990 to 1991, he was a Visiting Researcher at Cornell University. From 2006 to 2015, he was a Designated Professor of the Center for Embedded Computing Systems, in the Graduate School of Information Science, Nagoya University. His research interests include real-time systems, distributed systems, mobile systems, and software development environments. He is a member of IEICE and IEEE Computer Society.

Yoshiharu Ishikawa is a professor in Graduate School of Information Science, Nagoya University. His research interests include spatio-temporal databases, mobile databases, sensor databases, data mining, information retrieval, and Web information systems. He is a member of the Database Society of Japan, IEICE, JSAI, ACM, and IEEE.

Shinya Honda received his Ph.D. degree in the Department of Electronic and Information Engineering, Toyohashi University of Technology in 2005. From 2004 to 2006, he was a researcher at the Nagoya University Extension Course for Embedded Software Specialists. In 2006, he joined the Center for Embedded Computing Systems, Nagoya University, as an assistant professor, where he is now an associate professor. His research interests include system-level design automation and real-time operating systems. He received the best paper award from IPSJ in 2003. He is a member of ACM, IEEE, IEICE, and JSSST.

Hiroaki Takada is a professor at Institute of Innovation for Future Society, Nagoya University. He is also a professor and the Executive Director of the Center for Embedded Computing Systems (NCES), the Graduate School of Information Science, Nagoya University. He received his Ph.D. degree in Information Science from University of Tokyo in 1996. He was a Research Associate at University of Tokyo from 1989 to 1997, and was a Lecturer and then an Associate Professor at Toyohashi University of Technology from 1997 to 2003. His research interests include real-time operating systems, real-time scheduling theory, and embedded system design. He is a member of ACM, IEEE, IEICE, JSSST, and JSAE.

(Editor in Charge: *Takanori Takano*)