Introducing Substitution-Queries in Distributed Real-Time Database Management Systems [*]

Thomas Nolte and Dag Nyström
Mälardalen University
MRTC
Västerås, Sweden
{thomas.nolte, dag.nystrom}@mdh.se

Abstract

This paper introduces query mechanisms that allow automotive control-systems (using a distributed real-time database management system (RTDBMS)) to be queried, monitored and stimulated during run-time without violating its temporal properties. The mechanisms are completely transparent to the control application since they are handled by the RTDBMS. The COMET RTDBMS is extended with ad hoc capabilities to support the introduction of subscription and substitution queries, which are used for monitoring and stimulation. These queries are intended to be used by service and calibration tools to help in the development and maintenance of modern automotive systems. Using these queries could reduce development costs, result in higher quality of the system design and consequently yield higher reliability.

1 Introduction

In recent years, automotive control-systems have evolved from simple single processor systems to complex distributed systems. At the same time, the amount of data that needs to be managed by these systems is increasing dramatically; the data volume managed by automotive systems is predicted to increase 7-10% per year [8]. Current techniques for storing and manipulating data objects in automotive systems are ad hoc in the sense that they normally manipulate data objects as internal data structures. This lack of a structured approach to data management results in a costly development process with respect to design, implementation, and verification of the system [21]. It also makes the system difficult to maintain and develop while preserving consistency with the environment, e.g., maintaining temporal properties of data. As data complexity is growing, the need for a uniform, efficient, and persistent way to store data is becoming increasingly important. One way of handling this complexity is to use a real-time database management system (RTDBMS) as a tightly integrated part of the automotive control-system. RTDBMSs has the potential to solve many of the problems that application designers have to consider with respect to data management, e.g., locking of the data, persistency and deadlock situations. More importantly, incorporating an RTDBMS into an automotive control-system could reduce development costs, result in higher quality of the design of the systems, and consequently yield higher reliability [11]. Moreover, an RTDBMS enables development of advanced diagnosis tools, by providing a uniform interface to access data.

Today, there exists a number of both commercial and research databases suitable for embedded systems. Commercial embedded platforms include Pervasive.SQL [24], Polyhedra [9], Berkeley DB [26], and TimesTen [29]. Research real-time platforms include, DeeDS [2], RODAIN [16], STRIP [1], and BeeHIVE [27]. The general trend among these platforms is that commercial systems focus towards the embedded systems market, i.e., focus on flexibility, adaptability, and efficiency, while the research platforms mainly address real-time requirements. This discrepancy makes neither type of database system suitable for the automotive domain, where the adaptability and efficiency must be combined with maintaining real-time requirements. However, the real-time database management system COMET [20] aims to bridge this gap by providing both a light-weight, adaptable, and reconfigurable design paradigm [28], as well as efficient and predictable RTDBMS mechanisms [19].

This paper shows how the COMET RTDBMS can be extended with distribution mechanisms to allow service tools to query, monitor and stimulate a control system at run-

[*]This work is supported by SSF within the SAVE project, SAfety critical components for VEhicular systems.
Authors in alphabetical order.

time, while still maintaining a high level of abstraction. Three types of database queries, ad hoc queries, subscription queries, and substitution queries are introduced to obtain this behavior. These queries are handled by the RT-DBMSs on each node and are completely transparent to the control application and are not violating the temporal properties of the control system.

The outline of the paper is as follows; in Section 2, a background of automotive systems, current service tools, CAN, and COMET are given. The paper continues in Section 3, in which the extended data distribution is presented. Finally, in Section 4 the paper is summarized and concluded.

2 System model

In this section, typical automotive control-systems are presented together with how these systems distribute data. Furthermore, the COMET RTDBMS is presented in detail, including how it is distributed using CAN. Also, typical service tools used for automotive systems are presented, motivating the contribution of this paper, namely the database queries allowing for querying, monitoring and stimulation of data during run-time of real-time control-systems.

2.1 Automotive control-systems

Typical automotive control-systems are found in chassis and vehicle safety systems, such as Vehicle Dynamics Control (VDC) systems, also known as Electronic Stability Program (ESP). VDC/ESP is designed to assist the driver in over-steering, under-steering and roll-over situations [32]. This, and similar safety systems, such as the Anti-lock Brake System (ABS), all require *feedback control*.

Other safety-systems are air-bag systems [6], that control the operation of air-bags in the vehicle. Typically a vehicle contains several air-bags that are connected to sensors that detect abnormal situations, e.g., sudden acceleration or decelerations of the vehicle. Once an abnormal situation is detected the correct (depending on the type of crash) air-bags are inflated in a matter of half a millisecond.

Body and comfort electronics require both feedback and *discrete control* for subsystems such as climate control, cruise control, locks, window lifts, seat control and HMI, to mention a few. Typically body and comfort electronics rely on driver interaction and are not safety-critical, they involve hundreds of system states and events, and they interface to physical components in the vehicle, e.g., motors and switches.

Other automotive control-systems are powertrain systems and x-by-wire systems. Powertrain is the assembly of gears by which power is transmitted from the engine of the vehicle to the driving axis. Powertrain includes *engine control* which involves the coordination of fuel injection, engine speed, valve control, cam timing etc. X-by-wire is the notation for new subsystems replacing hydraulic and mechanical parts with electronics and computer (feedback) control systems. Examples of x-by-wire systems are steer-by-wire, shift-by-wire, throttle-by-wire and break-by-wire.

During the development and maintenance of these control systems, support through hardware tools is essential. These tools are used to monitor and diagnose both the software control-system and the mechanical systems.

2.2 Architecture

An automotive control-system (subsystem) consists of one or several Electronic Control Units (ECUs). An automotive system, consisting of several subsystems with a total of up to 70 ECUs, has to distribute thousands of variables and signals (data) over several communication networks [17], e.g., CAN networks. This makes a modern automotive system complex.

In this paper, all ECUs are assumed to be equipped with the COMET RTDBMS [20]. Moreover, these ECUs are assumed to be connected using the Controller Area Network (CAN) [25].

2.3 CAN

CAN, or the Controller Area Network [25], is a serial bus that was developed in the beginning of the eighties by Bosch. Today CAN is the most widely used vehicular network in the automotive industry. Over the years several different CAN standards have been developed and used in different applications, where the ISO 11898 [12, 13] is the most commonly used fieldbus in the European automotive industry.

A typical CAN application is any type of embedded system with soft real-time requirements and loop times of 5-50 ms. CAN transmits messages in an event-triggered fashion using frames containing 0 to 8 bytes of data and 4 to 6 bytes of header. These frames can be transmitted at speeds of 10 Kbps up to 1 Mbps.

CAN handles communication faults by retransmission, and there is no error containment or support for higher level of fault tolerance. However, it holds a strong position and will most likely continue to be the most used communication bus in the automotive application domain for a long time.

With CAN, messages are not interrupted while in transmission. Moreover, the CAN message identifier (ID) is representing the priority of the message. Hence, CAN is implementing non-preemptive *Fixed Priority Scheduling (FPS)*, and suitable analysis techniques can be used, like the FPS response-time tests to determine the schedulability of CAN message frames, presented by Tindell *et al.* [30, 31].

Using FPS, priorities are assigned to the messages before execution (offline), by the allocation of message identifiers (IDs). The message with the highest priority among all messages available for transmission is scheduled for transmission.

2.4 Data distribution in automotive control-systems

OSEK/VDX [22] is an effort to standardize and increase portability of automotive software. Among the OSEK/VDX specifications, OSEK/VDX COM [23] is a uniform communication environment for automotive control unit application software. OSEK/VDX COM provides communication services through a well defined API. Moreover, it specifies an Interaction Layer (IL) that provides the communications interface to the application. The application can transmit messages to other applications resident on the same ECU or on other ECUs. If the receiving application is resident on the same ECU, the IL handles the communications internally. If receiving applications are resident on another ECU, the IL packs one or more messages (signals) into Interaction Layer Protocol Data Units (I-PDUs). These I-PDUs are then sent to the Network Layer (NL), either periodically or explicitly initiated by some event. However, OSEK/VDX COM does not specify the NL other than defining some minimum requirements.

AUTOSAR [4], which aims at providing a global standard for software in automotive systems, proposes similar mechanisms for data distribution using a run-time environment to route communications both inter- and intra-ECU.

Apart from OSEK/VDX and AUTOSAR, automotive systems distribute signals over CAN in several ways, e.g., with the usage of Volcano as done by Volvo Car. The Volcano system [8] provides tools for packaging signals into messages, as well as assigning priorities for CAN messages to achieve a high utilization of the bus. Moreover, it is possible to perform timing analysis of the system using the Volcano tools. An offline schedulability test is done to ensure that all deadlines are met.

2.5 The COMET real-time database management system

The COMET RTDBMS [20] is a data management system intended primarily for embedded control-systems, e.g., automotive systems. COMET contains data management concepts that allow hard and soft application tasks to access and share data in a predictable and efficient way [19].

Tasks in the system interact with the RTDBMS through database transactions. Different types of tasks require dif-

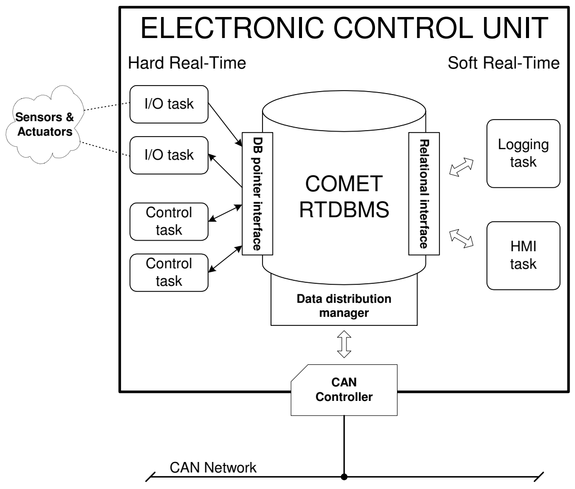

Figure 1. The architecture of an ECU using COMET

ferent kinds of transactions. Therefore, transactions are divided into the following two classes, see Figure 1:

1. **Soft transactions**, either precompiled or ad hoc (formulated and parsed at run-time) reside in soft real-time tasks. These transactions utilize the relational database query interface, such as, SQL [7], for database access. Soft transactions provide a flexible and dynamic access to the data in the database to the system and are especially suited for management tasks, e.g., logging, diagnosis, and, user interface (HMI) tasks, e.g., tasks controlling the dash board.

2. **Hard transactions**, which are precompiled, reside in hard real-time tasks. A hard transaction utilizes the database pointer interface (see Section 2.5.1) [19], providing an efficient and predictable access to individual data elements in the database. A majority of the transactions in a vehicle, i.e., transactions used for vehicle control, would fall into this class.

2.5.1 Database pointers

Database pointers allow individual data elements in an RT-DBMS to be accessed in an efficient and predictable manner [19]. They are intended as a complement to the relational data model, without limiting the expressibility of the relational query processing.

Figure 2 shows an example of an I/O task that periodically reads a sensor and propagates the sensor value to the database using a database pointer, in this case the oil temperature in the `engine` relation. The task consists of two parts, an initialization part (lines 2–4) executed when the

```
1 TASK OilTempReader(void) {
2   int s;
3   DBPointer *ptr;
4   bind(&ptr, "SELECT temperature FROM engine
                  WHERE  subsystem=oil;");
5   while(1){
6     s=read_sensor();
7     write(ptr,s);
8     waitForNextPeriod();
    }
  }
```

Figure 2. An I/O task that uses a database pointer

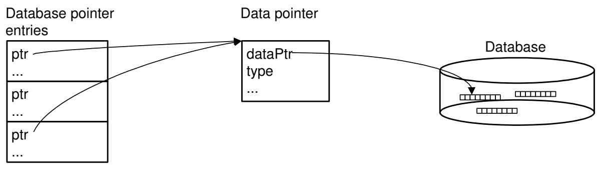

Figure 3. The data structures used by database pointers

system is starting up, and a periodic part (lines 5–8) scanning the sensor. The initialization of the database pointer is first done by declaring the database pointer (line 3) and then binding it to the data element containing the oil temperature in the engine (line 4). When the initialization is completed, the task begins to periodically read the value of the sensor (line 6), then propagates the value to the RTDBMS using the database pointer (line 7), and finally awaits the next invocation of the task (line 8).

Database pointers are implemented using the data structures shown in Figure 3. The binding of a database pointer to a database element is performed in the following steps:

1. A new *database pointer entry* is created in the RTDBMS.
2. The SQL query is executed. It is required that the result of the query is a single data element. If it is the first time the data element is bound to a database pointer, a new *data pointer* is created in the RTDBMS. The data pointer is initialized with the address of the data element and its type.
3. The database pointer entry is set to point at the data pointer.
4. Finally, the pointer to the database pointer entry is returned as a `DBPointer*`.

In addition to the `bind(ptr,q)` operation, the database pointer interface consists of the `remove(ptr)` operation which deallocates a database pointer, the `write(ptr,data)`, and the `read(ptr)` operations which updates, respectively reads the data element.

2.5.2 Concurrency in COMET

For applications that use multiple transactions possibly executed in parallel, some form of concurrency control in the RTDBMS is needed to maintain the consistency of the database. One common way of enforcing concurrency control is to introduce database locks. Before a transaction is allowed to access a data element in the database, the appropriate lock must be obtained. Database locks are similar to semaphores in the sense that they protect a shared resource.

For real-time systems, e.g., automotive control-systems, using locks might introduce unwanted blocking. This is especially true for systems that have both hard tasks executing at high frequencies, and soft tasks that might execute transactions with long execution times.

In COMET, this problem is solved by combining database locks (for soft transactions) with a versioning algorithm (for hard transactions). The concurrency control algorithm, denoted 2-version database pointer concurrency control (2V-DBP) [19], allow hard and soft transactions to share common data elements without interfering with each other.

2.5.3 Data distribution in COMET

To be able to support distributed automotive control-systems, COMET needs to be equipped with a distribution manager that communicates using the Controller Area Network (CAN). The distribution manager supports periodic pre-compiled queries (sporadic queries are treated as periodic based on their minimum inter-arrival time). When queries are distributed between ECUs in the system, data is *mapped* onto periodic CAN frames (messages) with identifiers (IDs) assigned to fulfill timing requirements of the transactions. The mapping of query data onto CAN frames is done similar to what is explained in Section 2.4, i.e., using a tool where signals and data are mapped onto messages. These messages are then periodically sent (multicasted/broadcasted) on the CAN bus. For the remainder of this paper, this periodically sent traffic is defined as the *original system CAN traffic*.

Both hard and soft periodic transactions are mapped between CAN frames and the COMET database using database pointers with 2V-DBP for efficient access. Hence, both packaging of data into messages for transmission, and updating data in the database upon message reception, are fast and simple operations. Moreover, the usage of a database simplifies data access. Using, e.g., the CAN Calibration Protocol (CCP) [3], lists of data elements are used to describe which data elements that are to be mapped into specific messages. In the CCP specification, these lists represent physical memory addresses. However, using the COMET RTDBMS with the database pointer concept, data access is handled on a logical (relational) level. Hence, no direct access to the ECU memory is required, protecting the ECU while providing a clear interface to the ECU's data elements, still allowing fast access to its data elements.

Note that the mapping of data onto a set of periodic CAN frames, together with performing the schedulability test on the set of messages, is done offline hence not requiring resources during runtime. During runtime mapping is done simply based on lookup tables, containing database pointers.

2.6 Service tools for automotive systems

During the development and maintenance of a modern automotive system, support through hardware tools connected to the control system is essential. These tools are mainly used to calibrate, test and diagnose both the software control system and the mechanical systems. For the remainder of this paper, these tools are simply referred to as *service tools*.

Today, a substantial effort is put into calibrating the parameters of an automotive control-system. The aim of this work is, among others, to optimize the performance of the system, and to comply with regulations regarding emissions etc. It is noteworthy that it is not the performance of the control system and its real-time properties, e.g., keeping deadlines and minimizing jitter, that is calibrated, but the performance of the mechanical system being controlled. For an automotive engine, typically several man-years are invested in calibration.

In order to perform the calibration, a *calibration tool* is used. These tools are typically connected to the vehicle via the CAN network, and then the automotive system can be monitored or updated using the tools. Even though commercial calibration tools that support the CCP exist, e.g., CAMEO for Vehicle Use [5], and CANaph Graph [33], it is not uncommon that in-house developed tools are used.

Service tools are also used to detect and diagnose potential system failures, both electrical and mechanical, during service of the vehicle. By providing service stations with powerful service tools, more efficient and accurate service can be performed. Desirable functionalities in such systems include:

- Downloading of warning- and error-logs from the vehicle. These logs contain information on system anomalies detected in the vehicle since its last service. Typical logs might include sporadic failures of sensors and abnormal sensor readings such as temperatures.
- Reading of a set of data elements in the vehicle. Such information can be used to further localize errors. An example of such a reading might be to obtain information on all current sensor values regarding the engine.
- Periodically subscribe to data elements to monitor fluctuations of their values over time, e.g., RPM- or temperature readings. Such information might be used to spot intermittent failures.
- Take control of (stimulate) a subsystem or function in the vehicle. Consider, for example taking over the acceleration pedal to be able to control the RPM of the engine. This functionality is useful for automated tests of the vehicle.

Since service tools communicate with the vehicle through its data, it is natural that the service tools communicate directly with the RTDBMSs in the ECUs, since they are responsible for managing the data in the automotive system. An advantage of this is also that the automotive application itself needs not to be aware of the existence of service tools.

So far COMET has only been discussed in the context of precompiled, periodic (and sporadic) offline scheduled data distribution (the original system CAN traffic). Hence, it must be extended to allow these new event driven ad-hoc activities to be executed. To allow the above mentioned activities, three new types of *distributed* database transactions are introduced:

1. **Soft ad hoc queries** These queries are similar to soft transactions, in the sense that they use a query language, but can now be formulated at run-time. Since it can not be foreseen which data elements an ad hoc query will access, these queries must be allowed to be distributed, i.e., gather information from different ECUs in the system.
2. **Subscription queries** These queries allow a data element, not currently distributed, to be subscribed to by a task on a different ECU or by a service tool. This query type consists of three parts, (i) a `start of subscription` in which the subscriber requests that a subscription is started, (ii) the actual subscription itself, and (iii) an `end of subscription`.
3. **Substitution queries** These queries allow current producers of data (sensors etc.) in the system to be overridden, in order for a service tool to take control of a certain subsystem. A substitution query has, as subscription queries, three parts, namely, (i) `start of substitution`, (ii) the actual substitution itself, in which the substitution data is propagated through the network, and (iii) the `end of substitution`.

3 Extending the COMET data distribution

To be able to incorporate the three query types introduced in Section 2.6, the data distribution in COMET needs to be extended to support *ad hoc CAN traffic*, see Figure 4. From the figure, it can be seen that the ad hoc CAN traffic is added in a lower priorities segment than the original system CAN traffic. In this segment, every node (both ECUs and service tools) are assigned a unique CAN ID to transmit on. This implies that any message collisions among ad hoc messages are handled by the CAN network. Furthermore, since all ad hoc CAN messages are transmitted with lower priorities than the original system CAN traffic, the schedulability is still valid regardless of the amount of ad hoc CAN traffic. However, it must be checked whether the ad hoc CAN traffic, although unlikely, introduce longer blocking times than caused by existing traffic, affecting the timely delivery of messages. It is common to assume the blocking time to be equal to the longest possible CAN-frame, in which case this check is not needed.

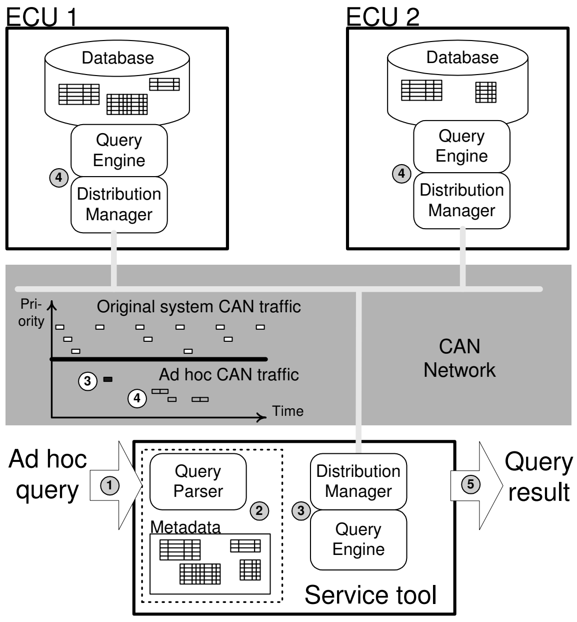

Figure 4. Execution of an ad hoc query

3.1 Ad hoc queries

Ad hoc queries are distributed database queries, formulated at run-time, normally by a service tool. This type of query allows a user to view the current state of the system using a powerful high level query language, i.e., SQL.

It is however noteworthy, that ad hoc queries provide neither transaction nor snapshot semantics, i.e., the result of an ad hoc query cannot be viewed as the state of the system at a single instance in time. However, ad hoc queries follow the COMET consistency properties, namely that data consistency and transaction semantics can be guaranteed locally on each ECU, but due to the fact that freshness typically is more favored than global consistency [14], inconsistencies among nodes can be tolerated. This coincides with the consistency of most automotive systems in practice.

The execution-flow of an ad hoc query is as follows (numbers in Figure 4 correspond to the list below):

1. An ad hoc query is entered to the service tool. These queries follow standard SQL syntax.

2. The query parser in the service tool parses the SQL query, and creates an execution plan. To create and optimize the execution plan, the query parser has access to the metadata, i.e., information such as structure and size of the data elements and relations in the database. The metadata is stored in the service tool.

3. The query engine in the service tool starts to process the execution plan. Usually, the first step in the execution plan is to retrieve data from the database, so therefore the distribution manager sends out a request for data on the CAN network, using the service tool's assigned CAN ID. Typically, such a request is on the form <*DATA_REQ, REL_NAME, COND*>, where all tuples (rows in a relation) for the relation *REL_NAME* which meet the boolean condition *COND* are requested.

4. All distribution managers in the ECUs will then forward this message to its local query engine, which will launch a soft transaction retrieving the requested tuples. It is however noteworthy that only the tuples that each ECU has ownership of (stated in each ECU's metadata) is retrieved. Declaring ownership for each tuple avoids several ECUs to retrieve (and thus return) the same tuple to the service tool. When all tuples are retrieved, each ECU's distribution manager packs them together in CAN messages and transmits them on its respective CAN ID. When a distribution manager is completed (possibly after sending 0 tuples), it acknowledges end of transmission.

5. Finally, the query engine in the service tool completes the execution plan and outputs the query result.

3.2 Subscription queries

Subscription queries are used to monitor individual internal data elements over a period of time. These queries utilize just as the ad hoc queries, low priority CAN traffic to initialize and terminate subscriptions. The level of service, with respect to frequency and Quality of Service (QoS), of the subscription can be specified. QoS is divided into two classes, either the subscription is performed as a background service (soft real-time), using ad hoc traffic, or it is guaranteed (hard real-time). Guaranteed subscriptions undergo an admission control in which a schedulability analysis (as presented in Section 2.3) of the original system CAN traffic together with the added subscription traffic is performed. This analysis determines whether or not to accept (admit) the subscription. It is assumed that the service tool has the full knowledge of the system, in terms of the original system's CAN traffic. If admitted, the subscription will temporarily be treated as a part of the original system CAN traffic. The execution-flow of a subscription query is as follows:

1. A subscription query is entered into the service tool. The query consists of the following:
   - *<NODE, REL_NAME, KEY, ATTRIBUTE>*, which corresponds to the relation name, the key and attribute (row and column) of the tuple located in the ECU pointed out by *NODE* in which the data element to subscribe upon is located. This information is enough to uniquely identify any data element in any database in the system.
   - *<PERIODICITY, QoS>*, which corresponds to the periodicity and the QoS level (soft or hard) of the subscription.
2. The query engine of the service tool first checks in its metadata if the data to view already is distributed (i.e., is already in the original system CAN traffic) with at least the same level of service. In that case the service tool uses that distribution.
3. For queries with the QoS level set to hard, the query engine in the service tool performs an admission control. In the admission control, the following is determined; (i) if the subscription, given its periodicity, can be safely inserted into the original system CAN traffic without violating any system requirements (i.e., schedulability analysis is performed), and (ii) at which priority (CAN id) it can be transmitted.
4. Given that the query is admitted (or if the QoS level is soft) an ad hoc message is sent out with the following format: *<SUB_REC, NODE, REL_NAME, KEY, ATTRIBUTE, PERIODICITY, CANID>*.
5. The ECU being addressed receives the message and acknowledges it.

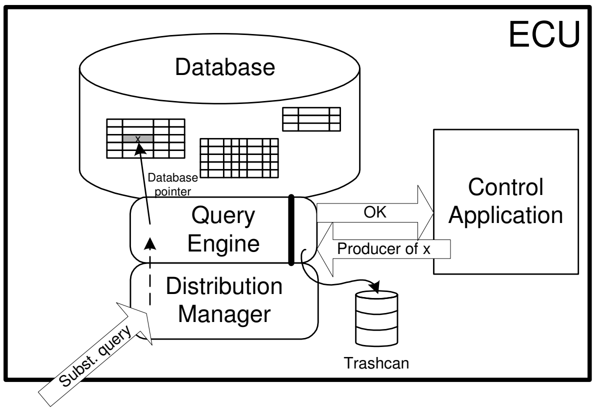

Figure 5. Execution of an substitution query

6. The ECU then creates a new database pointer (if not already existent) and periodically starts to transmit on the assigned CAN ID.
7. Eventually, the service tool transmits an `end of subscription`, and the subscription is terminated.

3.3 Substitution queries

Substitution queries are used to stimulate the system from a service tool or similar. When a substitution query is active for a data element, it overrides the producer of that element. In Figure 5, a substitution query for data element x is active, thus any producer in the control application is overridden. Still however, the producers will receive a normal response (e.g., `query successful` or similar) on their data updates on x. This implies that, from the control applications point of view, a substitution (or subscription) is completely transparent since it is handled by the RTDBMS. The workflow for a substitution query is the same as for a subscription query, except that data packages are sent from the service tool to the control application. Just as for subscription queries, substitution queries can be executed on both a soft and a hard QoS-level.

4 Summary

During the development and maintenance of an automotive system, service tools play an important role in calibration, testing and diagnosis. These tools need an intimate access to system data to be able to monitor the system behavior during run-time.

This paper presents how a real-time database management system can be used to enable this behavior. The COMET RTDBMS is extended with three new query types for querying, monitoring and stimulating data during runtime of the system, without violating the temporal properties of existing control systems. These new queries could help in the development of modern automotive systems, reducing development costs, resulting in higher quality of the system design and consequently yield higher reliability. Furthermore, the approach presented in this paper enables any data residing in the database to be monitored and stimulated during runtime, also data that is not explicitly preconfigured for this data access.

Future work includes extending the distributed RT-DBMS to also support other networks [18], such as Local Interconnect Network (LIN) [15] and Flexray [10].

References

[1] B. Adelberg, B. Kao, and H. Garcia-Molina. Overview of the STanford Real-time Information Processor (STRIP). *SIGMOD Record*, 25(1):34–37, 1996.

[2] S. F. Andler, J. Hansson, J. Eriksson, J. Mellin, M. Berndtsson, and B. Eftring. DeeDS Towards a Distributed and Active Real-Time Database System. *ACM SIGMOD Record*, 25(1):38–40, 1996.

[3] ASAP Standard. CCP - Can Calibration Protocol, Version 2.1, February 1999.

[4] AUTOSAR. Homepage of Automotive Open System Architecture (AUTOSAR). http://www.autosar.org/.

[5] AVL LIST GMBH. Cameo for Vehicle Use. http://www.avl.com.

[6] R. Boys. Safe-by-Wire: The Leading Edge in Airbag Control. In *SAE World Congress*, Detroit, MI, USA, 2003. SAE.

[7] S. Cannan and G. Otten. *SQL - The Standard Handbook*. MacGraw-Hill International, 1993.

[8] L. Casparsson, A. Rajnak, K. Tindell, and P. Malmberg. Volcano - a revolution in on-board communication. *Volvo Technology Report 98-12-10*, 1998.

[9] Enea AB. http://www.enea.se.

[10] FlexRay Communications System - Protocol Specification. Version 2.0, June 2004.

[11] T. Gustafsson and J. Hansson. Data management in real-time systems: a case of on-demand updates in vehicle control systems. In *Proceedings of the Real-Time Application Symposium (RTAS 2004)*. IEEE Computer Society Press, May 2004.

[12] ISO 11898. Road Vehicles - Interchange of Digital Information - Controller Area Network (CAN) for High-Speed Communication. *International Standards Organisation (ISO)*, ISO Standard-11898, Nov 1993.

[13] ISO 11898-1. Road Vehicles - Controller Area Network (CAN) - Part 1: Data link layer and physical signalling. *International Standards Organisation (ISO)*, ISO Standard-11898-1, 2003.

[14] T.-W. Kuo and A. K. Mok. SSP: a Semantics-Based Protocol for Real-Time Data Access. In *Proceedings of 14th IEEE Real-Time Systems Symposium*, pages 76–86. IEEE Computer Society, December 1993.

[15] LIN Consortium. LIN - Local Interconnect Network. http://-www.lin-subbus.org/.

[16] J. Lindstrom, T. Niklander, P. Porkka, and K. Raatikainen. A Distributed Real-Time Main-Memory Database for Telecommunication. In *Proceedings of the Workshop on Databases in Telecommunications*, pages 158–173. Springer, September 1999.

[17] N. Navet, Y. Song, F. Simonot-Lion, and C. Wilwert. Trends in Automotive Communication Systems. *Proceedings of the IEEE*, 93(6), June 2005.

[18] T. Nolte, H. Hansson, and L. Lo Bello. Automotive Communications - Past, Current and Future. In *Proceedings of the* 10^{th} *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'05)*, Catania, Italy, September 2005.

[19] D. Nyström, M. Nolin, A. Tešanović, C. Norström, and J. Hansson. Pessimistic Concurrency Control and Versioning to Support Database Pointers in Real-Time Databases. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, pages 261–270. IEEE Computer Society, June 2004.

[20] D. Nyström, A. Tešanović, M. Nolin, C. Norström, and J. Hansson. COMET: A Component-Based Real-Time Database for Automotive Systems. In *Proceedings of the Workshop on Software Engineering for Automotive Systems*, pages 1–8. The IEE, June 2004.

[21] D. Nyström, A. Tešanović, C. Norström, J. Hansson, and N.-E. Bånkestad. Data Management Issues in Vehicle Control Systems: a Case Study. In *Proceedings of the 14th Euromicro Conference on Real-Time Systems*, pages 249–256. IEEE Computer Society, June 2002.

[22] OSEK/VDX. Open Systems and the Corresponding Interfaces for Automotive Electronics. http://www.osek-vdx.org/.

[23] OSEK/VDX-Communication. Version 3.0.3, July 2004. http://www.osek-vdx.org/mirror/OSEKCOM303.pdf.

[24] Pervasive Software Inc. http://www.pervasive.com.

[25] Robert Bosch GmbH. BOSCH's Controller Area Network. http://www.can.bosch.com/.

[26] Sleepycat Software Inc. http://www.sleepycat.com.

[27] J. A. Stankovic, S. H. Son, and J. Liebeherr. *Real-Time Databases and Information Systems*, chapter BeeHive: Global Multimedia Database Support for Dependable, Real-Time Applications, pages 409–422. Kluwer Academic Publishers, 1997.

[28] A. Tešanović, D. Nyström, J. Hansson, and C. Norström. Aspects and Components in Real-Time System Development: Towards Reconfigurable and Reusable Software. *Journal of Embedded Computing*, February 2004.

[29] TimesTen Performance Software. http://www.timesten.com.

[30] K. W. Tindell, A. Burns, and A. J. Wellings. Calculating Controller Area Network (CAN) Message Response Times. *Control Engineering Practice*, 3(8):1163–1169, 1995.

[31] K. W. Tindell, H. Hansson, and A. J. Wellings. Analysing Real-Time Communications: Controller Area Network (CAN). In *Proceedings of* 15^{th} *IEEE Real-Time Systems Symposium (RTSS'94)*, pages 259–263, San Juan, Puerto Rico, December 1994. IEEE Computer Society.

[32] A. T. van Zanten, R. Erhardt, K. Landesfeind, and G. Pfaff. VDC systems development and perspective. In *SAE World Congress*. SAE, 1998.

[33] Vector-CANtech, Inc. CANape Graph. http://www.vector-cantech.com.