COMET: A Component-Based Real-Time Database for Automotive Systems *

Dag Nyström†, Aleksandra Tešanović⋆, Mikael Nolin†, Christer Norström†, and Jörgen Hansson⋆

†Mälardalen University
Mälardalen Real-Time Research Centre
Västerås, Sweden
{dag.nystrom, mikael.nolin, christer.norstrom}@mdh.se

⋆Linköping University
Dept. of Computer Science
Linköping, Sweden
{alete,jorha}@ida.liu.se

Abstract

With the increase of complexity in automotive control systems, the amount of data that needs to be managed is also increasing. Using a real-time database management system (RTDBMS) as a tightly integrated part of the software development process can give significant benefits with respect to data management. However, the variability of data management requirements in different systems, and the heterogeneousness of the nodes within a system may require a distinct database configuration for each node. In this paper we introduce a software engineering approach for generating RTDBMS configurations suitable for resource-constrained automotive control systems, denoted the COMET development suit. Using software engineering tools to assist developers with design and analysis of the system under development, different database configurations can be generated from pre-fabricated components. Each generated COMET database contains only functionality required by the node it is executing on.

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; data volume managed by automotive systems is predicted to increase 7-10% per year [5]. 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 [17]. 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. Using a real-time database management system (RTDBMS) as a tightly integrated part of an automotive control system 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 can reduce development costs, result in higher quality of the design of the systems, and consequently yield higher reliability [9].

The variability of data management requirements in different automotive control systems requires distinct RTDBMS configurations specially suited for the particular system [27]. Since an automotive control system is heterogeneous, consisting of several nodes (called electronic control units, ECUs), see figure 1, the ability to configure the RTDBMS to suit the requirements of an individual node is crucial. For instance, an automotive system could consist of a small number of resource adequate ECUs responsible for the overall performance of the vehicle, e.g., 32bit CPUs with a few Mb of RAM, and a large number of ECUs responsible for controlling specific subsystems in the vehicle, which are significantly resource-constrained, e.g., an 8bit micro-controller and a few kb of RAM [17]. ECUs with greater amount of resources usually have real-time operating systems support, which is not affordable in small resource-constrained ECUs. Although different in their characteristics and available resources, all nodes in an automotive control system are exchanging, sharing and manipulating data, thereby requiring a uniform way of data

*This work is supported by SSF within the SAVE project, SAfety critical components for VEhicular systems.

management, e.g., via a RTDBMS.

The heterogeneous characteristics of nodes in an automotive control system result in a need to have distinct RTDBMS configurations suited for a particular node [17]. In safety-critical nodes, tasks are often non-preemptive and scheduled off-line, implying that a RTDBMS configuration for that node could be made small in size and provided functionality, since the majority of the RTDBMS's functionality, such as synchronization and concurrency-control, could be handled off-line. Less critical and larger nodes have preemptable tasks, requiring a RTDBMS configuration with run-time concurrency control mechanisms, and support for database queries formulated during run-time (ad-hock queries). A configurable RTDBMS supporting different types of nodes would, from the application's point of view, provide uniform access to the data regardless of the size and characteristics of an ECU.

Today, there exists a number of commercial databases suitable for embedded systems, e.g., Pervasive.SQL [19], Polyhedra [20], Berkeley DB [22], and TimesTen [32]. Although small in size and therefore suitable for resource-constrained automotive control systems, these databases do not incorporate real-time behavior. This in turn implies that their behavior cannot be analyzed, which makes them unsuitable for deployment in an automotive system. Research projects that are building real-time database platforms, such as DeeDS [2], RODAIN [12], STRIP [1], and BeeHIVE [24], mainly address real-time requirements, are monolithic, and targeted towards a larger-scale real-time application, which makes them unsuitable for use in embedded resource-constrained environments.

In this paper we propose a software engineering approach for generating RTDBMS configurations suitable for resource-constrained automotive control systems. This approach is supported by the **COMET development suit**. The suit consists of a set of data management, analysis and configuration tools, as well as a library of pre-defined software artifacts providing specific RTDBMS functionality. The library of artifacts and the possible configurations of the RTDBMS are referred to as the COMET RTDBMS platform. With the COMET development suit we aim at providing software developers an automated way of tailoring and analyzing the data management for a particular automotive control system, or a node in the system. The COMET RTDBMS platform, a part of the COMET development suit, is developed using an approach to aspectual component-based software development (ACCORD) [30]. ACCORD enables us to utilize the benefits of component-based software development (CBSD) [25] by developing components that encapsulate specific real-time database functionalities. ACCORD also enables us to exploit the benefits of aspect-oriented software development (AOSD) [11] by providing a way of encapsulating, managing, and implementing cross-cutting concerns in a RTDBMS in a predictable manner; crosscutting concerns include concurrency control, logging, and recovery. In AOSD, a crosscutting concern is a functionality or non-functional feature that cannot cleanly be encapsulated in a procedure, function, object or a class [11].

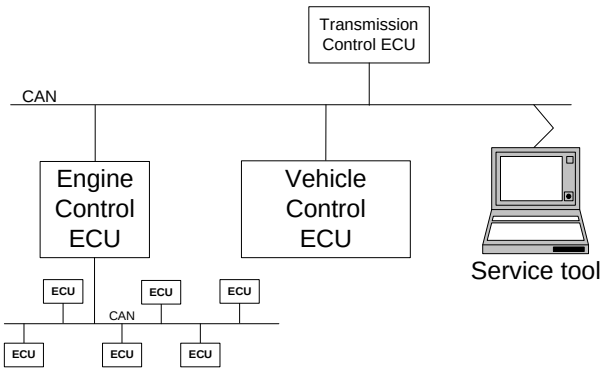

Figure 1. An heterogeneous automotive control system

The paper is organized as follows. In section 2 we present the COMET development suit. We present the key concepts used in the COMET development suit in section 3, including COMET aspects and components and possible COMET RTDBMS configurations. We conclude the paper and discuss our future work in section 4.

2. The COMET development suit

To successfully and efficiently generate systems from a library of pre-defined artifacts, the development process should be supported by appropriate tools. In this section we present our view of the overall development process to obtain system-specific RTDBMS configurations. Figure 2 shows the constituents of this process.

As shown in figure 2, the development of a RTDBMS configuration starts with specifying the requirements of an automotive control system, which are then used as input for making a model of the system. This model consists of the nodes, their interconnections and the individual run-time properties, e.g., the scheduling policy of the node, if the tasks are preemptive or not, available memory, and CPU resources. The goal of making the model of the underlying system is to derive required database functionality for each of the node in the system. Examples of functionality are support for ad-hoc queries (queries dynamically created during run-time), and to enable the data organization to be changed during run-time (i.e., provide a dynamic database schema). Next, a model of the database, i.e., the actual data, and any precompiled queries are derived with the help of

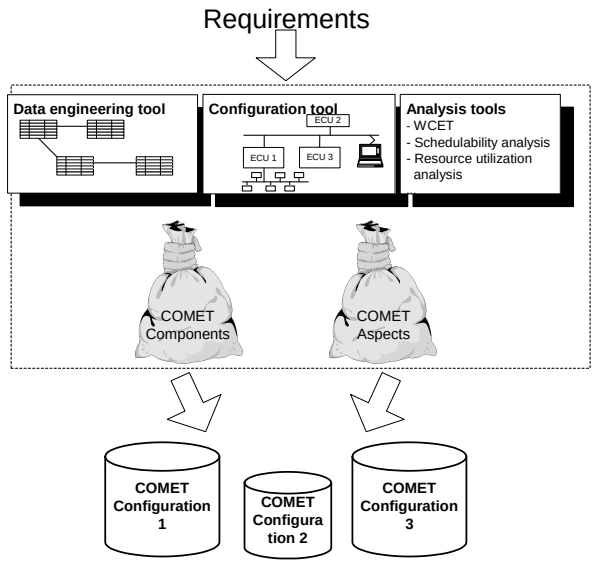

Figure 2. The COMET development suit

the data engineering tool. This step also involves specifying which parts of the database should be available on which node, and the temporal properties of the data, such as temporal consistency [21]. This information, i.e., desired database functionality for each node, data model, and database schema, is then used by the configuration tool to select a set of aspects and components from the library to form a database configuration suitable for each of the nodes (see figure 2). The overall decomposition of the database functionality into aspects and components, and the development of components and aspects, is done according to the ACCORD design principle (see section 3.1). The obtained COMET configurations can then be analyzed with respect to run-time properties, e.g., worst case execution time, memory requirements, and response time analysis, by the analysis tools. If the analysis indicates that the configuration is unfeasible, the configuration step and analysis step could be further iterated until an acceptable solution is found.

The resulting RTDBMSs are configured to contain no more than the needed functionality, thus reducing both computational costs and memory requirements.

3. The COMET key concepts

As mentioned, different nodes in the automotive control system may require distinct RTDBMS configurations. Component-based databases [4, 7, 8, 10, 13, 18, 22] using the component-based software development paradigm [25] can be partially or completely assembled from a pre-defined set of components with well-defined interfaces. Therefore, these are suited for tailoring a database system towards an application. However, there are aspects of database systems that are difficult to encapsulate into components with well-defined interfaces; typical examples include synchronization, transaction models, and database policies such as concurrency control [3]. These aspects are crosscutting concerns that permeate the whole system and affect multiple components. Hence, using traditional component-based approach is necessary but not sufficient to enable efficient development of configurable RTDBMSs. Therefore, in COMET we use an approach to aspectual component-based real-time system development (ACCORD) [30, 31] (discussed in section 3.1) that provides a notion of a reconfigurable component, and thereby enables both encapsulation of RTDBMS functionality into components and efficient handling and implementation of crosscutting concerns via aspects.

Using the ACCORD approach, different COMET components and aspects can be developed, and then used for assembling COMET configurations suitable for a specific automotive control system. Existing COMET components and aspects are discussed in section 3.2. We illustrate the COMET concepts introduced in this section with an example of the COMET configuration suitable for a particular node in the automotive control system in section 3.3.

3.1. Aspects and components in RTDBMSs

ACCORD utilizes notions from both component-based and aspect-oriented software development, integrating them into real-time system development. While CBSD traditionally use black box as an abstraction metaphor for the components, AOSD utilizes the white box metaphor to emphasize that all details of the implementation should be revealed. ACCORD supports the notion of a reconfigurable real-time component model (RTCOM) [26, 30, 31]. Components built using RTCOM are grey boxes as they are encapsulated in interfaces but changes to their behavior can be performed in a predictable way using aspects. Aspects are allowed to modify the code of the components in predefined, explicitly declared, reconfiguration points. In this section we briefly review RTCOM and its configurability via aspects, while detailed descriptions of ACCORD and RTCOM can be found in [26, 30, 31].

Aspects are programming-language level constructs encapsulating crosscutting concerns that invasively change the code of the component and correspond to the traditional aspects in existing aspect languages. The main constituents of aspects are: (i) components, written in a component language, e.g., C, C++, and Java; (ii) aspects, written in a corresponding aspect language, e.g., AspectC [6], AspectC++ [23], and AspectJ [33]; and (iii) an aspect weaver, which is a preprocessor that inserts code from the aspects into the

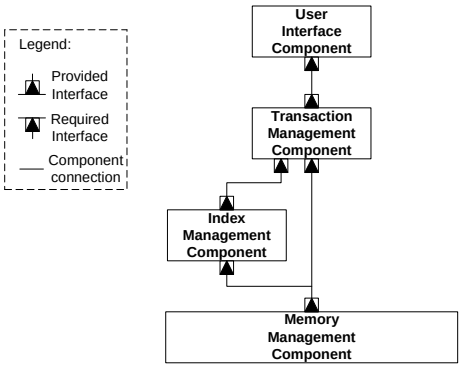

Figure 3. The basic architecture of COMET

reconfiguration points of the components.

An aspect in an aspect language consists of pointcuts and advices. Next we give a brief review of a typical syntax and semantics used in an aspect language; figure 5 shows a concrete example of an aspect woven into a component. A *pointcut* in an aspect language consists of one or more join points, and it is described by a pointcut expression. A *join point* refers to a point in the component code where aspects should be woven, e.g., a method, a type (struct or union). In RTCOM join points are explicitly declared in the component interfaces as reconfiguration points, and these are declared such that temporally predictable weaving in the component code can be done. An *advice* is a declaration used to specify the code that should run when the join points, specified by a pointcut expression, are reached. Different kinds of advices can be declared, such as: (i) *before advice* code is executed before the join point, (ii) *after advice* code is executed immediately after the join point, and (iii) *around advice* code is executed in place of the join point.

3.2. The COMET RTDBMS platform

A central goal with COMET is to enable configurability so that it can handle a variety of different application requirements; COMET has an architecture that allows this [14]. Following the ACCORD design method described in section 3.1, the architecture of COMET consists of a number of components and a number of aspects. Each component provides a well-defined service through operations that are defined in a component's interface. Aspects and components that together provide a specific functionality are denoted as aspect packages.

The foundation of COMET consists of a basic architecture in which components can be instantiated (see figure 3). A fully instantiated basic architecture is referred to as a basic configuration. The basic configuration builds a fully functional RTDBMS capable of storing, manipulating and querying data using some high level database query language. Even though a basic configuration is considered to be a RTDBMS, it has limited functionality, e.g., it cannot handle concurrent transactions, and it has no database crash recovery mechanisms. A basic COMET configuration consists of the following four components:

1. **The user interface component** (UIC) provides a database interface to the application. This interface consists of a data manipulation language, in which the user (application) can query and manipulate data elements. Furthermore, the interface consists, if configured so, of a data definition language which enables the user to manipulate the database schema, e.g., creating and dropping relations (tables). Application requests are parsed by the UIC, and are then converted into an execution-plan.

2. **The transaction management component** (TMC) is responsible for executing incoming execution-plans, thereby performing the actual manipulation of the data in the database.

3. **The index management component** (IMC) is responsible for maintaining an index of all tuples in the database. This is normally done through hash-tables or index-trees. The IMC is capable of transforming a database key into the memory address of the tuple correspondent to the database key. Furthermore, the IMC maintains the database schema in its index of metadata.

4. **The memory management component** (MMC) is responsible for memory allocation of tuples, metadata, and database indexes.

By selecting versions of these components, different basic COMET configurations can be derived.

In addition to these, mandatory, components, it is possible to add optional components to the architecture, such as the scheduling management component (SMC), which is responsible for scheduling transactions. This is useful when the application is preemptive and multiple transactions can be issued simultaneously. However, noteworthy is that a basic configuration of COMET cannot execute multiple transactions concurrently. In this case the SMC maintains the list of transactions in a ready queue and releases the next transaction when the previous is completed.

The services described above are all well defined and their activities are to a high degree isolated, i.e., it would be possible to exchange each one of these services with a different implementation, as long as they interact with other services in the same way. This makes these services suitable for encapsulation into components.

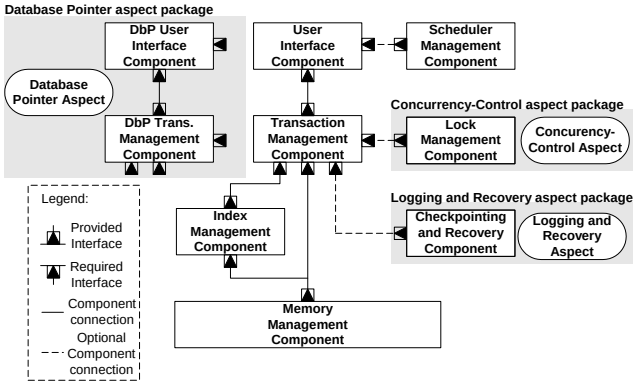

Figure 4. The architecture of COMET with aspect packages

However in a RTDBMS there are concerns which cannot be divided into isolated activities, but rather crosscut multiple components in the system. These crosscutting concerns are, in COMET, encapsulated into aspect packages, which can contain both aspects and components. In figure 4, three such aspect packages can be seen, namely:

1. **The concurrency control aspect package** (CCA) allows multiple transactions to be executed concurrently. Managing concurrent transactions requires some form of concurrency control. The CCA consists of a locking management component (LMC) and a concurrency control aspect. The LMC allows transactions to obtain read- and write-locks on data elements. The concurrency control aspect contains the code for obtaining and releasing the locks, as well as a transaction conflict resolution method. The code of the concurrency control aspect is woven into (i) the TMC to force transactions to obtain locks before accessing data elements, and to release them when finished, (ii) the SMC to enable it to handle graceful termination of aborted transactions, and (iii) the LMC to adapt its behavior according to the conflict resolution policy used.

2. **The database pointer aspect package** (DBPA) enables the application to access individual data elements within the database in an efficient and predictable way. A database pointer [16] is a pointer that is first bound to a specific data element, which then can be read and written with a minimum overhead. Database pointers are used together with the relational data model, and they do not place limitations on the RTDBMS with respect to reorganizing the database schema during run-time. The database pointer concept is developed with automotive control in mind and is a fundamental part when integrating a RTDBMS into an automotive control system. The DBPA consists of two components, namely the database pointer user interface component that provides the application with the database pointer user interface, and the database pointer transaction management component that executes the database pointer operations. Furthermore, the DBPA consists of a database pointer aspect which is woven into the TMC and the IMC, adapting them to co-exist with database pointers.

3. **The logging and recovery aspect package** (LRA) ensures that the database is consistent after a system crash. Logging and recovery is performed through periodic checkpoints, where an image of the database is saved to a persistent storage and all intermediate changes to the database are logged. The aspect package consists of one component, the checkpointing and recovery component (CRC), which contains methods defining how to checkpoint and log changes to the database, and one aspect, the logging and recovery aspect (LRA). The LRA is woven into the MMC, the TMC, and the CRC.

Hence, the COMET RTDBMS platform contains a set of components and aspect packages that are suitable for automotive control systems.

COMET components discussed in this section are suitable for configuring RTDBMSs for use in ECUs requiring the relational data model that can be manipulated using ad-hoc queries. Currently these components support the most common database query commands, namely the `select`, `insert`, `update`, `delete`, `create table` and `drop table`. However, there are also COMET components that only allow static database schemas and precompiled queries, which are suitable for nodes that cannot afford, or do not require, ad-hoc queries. In a configuration generated from such components most of the functionality is handled off-line using the COMET tools.

3.3. A configuration example

To illustrate how the COMET RTDBMS can be configured to suit a particular ECU we present the following example in which we create a suitable COMET configuration based on a number of requirements. Note that the given requirements are typical data management requirements that can be found in an engine ECU of a modern car [9, 17]. Consider the engine ECU with the following data management requirements:

R1: The application performs computations using data obtained from sensors. Sensor data should reflect the state of the controlled environment and, hence, are associated with hard real-time temporal requirements and data freshness requirements.

R2: The application performs diagnostics on the system in order to analyze the system behavior. The diagnostics should be performed both in the steady state of the vehicle and in the transient states, in order to get the full spectrum of data and be able to analyze the vehicle behavior under all situations. Diagnostic operations performed on the system are not critical to the operational safety of the vehicle and, therefore, are associated with soft real-time temporal requirements.

R3: The set of data in the system is fixed at compile time and is never changed during run-time.

R4: The ECU uses preemptive fixed-priority scheduling, implying that multiple tasks can execute concurrently. This in turn implies that the same data items can be read and written by different tasks (which could result in inconsistent data values in the system).

R5: It should be possible to connect a service tool to retrieve system data.

When configuring a RTDBMS for such a system, we begin by modeling the ECU based on the requirements. The configuration tool is then used to provide a suitable basic COMET configuration.

In this case, the basic configuration could be based on the components providing a relational data model, since creating views and complex queries is a required feature of the RTDBMS configuration for this ECU (requirement **R2**). Furthermore, the relational data model provides support for ad-hoc queries (requirement **R5**). However, since a dynamic database schema is not necessary (requirement **R3**), the MMC from the basic COMET configuration in figure 3 can be replaced with components providing static data management and database indexing.

When a suitable basic configuration has been selected, the configuration tool proceeds by adding suitable aspect packages. First, the database pointer aspect package is selected to provide fast access of individual data elements in the ECU (requirement **R1**). Then, in order to fulfill requirement **R4** and enable concurrent execution of transactions such that data values in the database are kept consistent, the concurrency control aspect package is selected. In this case, a concurrency control algorithm 2V-DBP [15] could be selected. 2V-DBP combines locking with two-versions of selected parts of the database to enable hard database pointer transactions to execute without being blocked by soft relational transactions. Using 2V-DBP enables the application to support both hard control tasks and soft diagnostics task in the ECU. Now, all the requirements are fulfilled and the RTDBMS configuration is complete.

The next step is to enter the database schema into the data engineering tool. Since we have chosen a static database schema, the necessary data structures for the database are created. In this step, precompiled queries can also be created and optimized.

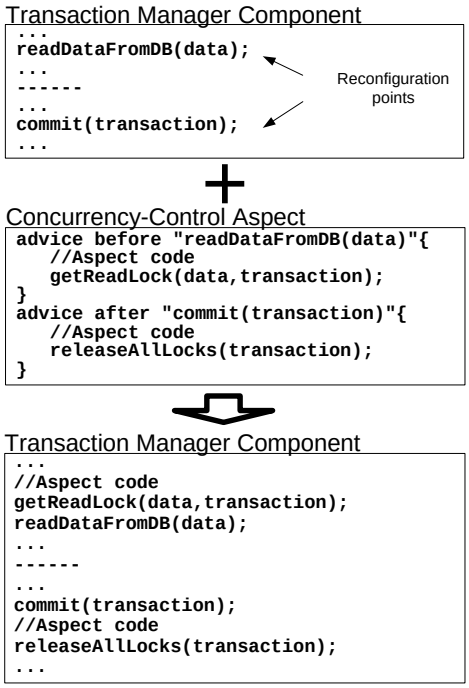

Figure 5. A simplified example of the weaving process

Finally, the analysis tools are used to check if the generated configuration of the RTDBMS is feasible. The analysis tool is used to determine the worst case execution time of the RTDBMS so that schedulability analysis can be performed. Furthermore, the memory requirements of the RTDBMS and physical storage needed for storing data in the system are analyzed.

Given that the configuration and analysis can be done on the models of the components, aspects, and the RTDBMS configuration, the actual weaving and component composition process can be performed after the database configuration is found feasible by the tools. Figure 5 illustrates the weaving process and its constituents for the TMC and the concurrency control aspect. The concurrency control aspect contains two advices: (1) an advice of type before that defines the code that should be inserted into TMC reconfiguration point `readDataFromDB()` to ensure that a transaction obtains the lock on a data item, and (2) an advice of type after that defines the code to be inserted immediately after the transaction commits to ensure that the transaction releases all the locks it has obtained while executing in the database. The result of the weaving is the TMC modified at the reconfiguration points, such that every read of the data

item is now preceded by locking, while every commit of the transaction is followed by unlocking (see figure 5). Note that this example is simplified to show main constituents of the aspects, components, and their possible interaction. However, in the actual implementation, the concurrency control aspect is more complex and contains advices that crosscut the behavior of the SMC and LMC components as well. Moreover, the concurrency control aspect code provides an efficient way of handling possible deadlocks in the database. The overall benefit of having aspects for tailoring database components is the ability to use reconfiguration points in the syntax of the pointcuts of advices. This in turn enables us to identify not only the places in the code of the program that have the same signature as the reconfiguration points (used in the presented example), but also define pointcuts that refer to the execution of the reconfiguration points (i.e., after the call has been made and a function started to execute), and to match any reconfiguration point that has values of a specified type. Also, operators &&, ||, and ! can be used to logically combine or negate pointcuts. Furthermore, separation of concerns into aspects enables us to have both components without aspects, and reconfigured components with aspects as aspect weaving results in a new component weaved with aspect code, but leaving the code of the original component unchanged and available for future reuse, i.e., now we can also reuse already reconfigured components or use original components with different aspects in other reuse contexts. For extensive discussion on benefits of having aspects for tailoring components in the RTDBMSs we refer interested readers to [26, 31].

Finally, if the obtained final configuration of the RTDBMS is found to meet the original requirements, as well as the timing requirements, the RTDBMS is compiled and made ready for deployment into the application.

By this simple example we have shown how a set of requirements can be used when configuring a COMET RTDBMS in order to get a RTDBMS specially suited for a particular ECU in an automotive control system.

4. Conclusions

In this paper we have presented a configurable real-time database platform, called COMET, which provides support for efficient data management in heterogeneous automotive systems. The COMET platform consists of a library of components and aspects, and is supported by a tool suite. The COMET tool suit assists system designers in configuring and analyzing different COMET configurations based on the specific requirements of the targeting automotive system and its nodes. While components encapsulate distinct functionalities of a database system, aspects allow efficient tailoring of the components and the database system based on the requirements of the underlying automotive system or its node. Our approach in providing different COMET configurations by using components in the library together with aspects can also be viewed as efficient product-line architectures of real-time database systems in the automotive domain.

We have showed the differences of the provided properties of the commercially available embedded databases, as well as real-time databases, compared with the needs of automotive control systems. To the best of our knowledge, no previous work exists that takes a holistic approach to data management in automotive systems. Given the increase of data complexity in automotive systems it is our experience that a more structured form of data management will be necessary in a near future, in order to keep time to market as well as development and maintenance costs down.

Although we have presented and discussed distinct configurations of COMET suitable for different nodes in the automotive systems, these were not developed using the full automated support of the tool suite. Rather, the automation done in the development process of COMET configurations so far has been focused on the analysis tools, where we developed the tool for analyzing different configurations of aspects and components with respect to their temporal properties [28, 29]. The remaining part of the COMET tool suit is currently under development. Further work on integrating the database into the component framework, to allow components to be easily distributed over multiple nodes is also planned.

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] D. Batory and S. O'Malley. The design and implementation of hierarchical software systems with reusable components. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 1(4):355–398, 1992.

[4] M. J. Carey, L. M. Haas, J. Kleewein, and B. Reinwald. Data access interoperability in the IBM database family. *IEEE Quarterly Bulletin on Data Engineering; Special Issue on Interoperability*, 21(3):4–11, 1998.

[5] L. Casparsson, A. Rajnak, K. Tindell, and P. Malmberg. Volcano - a Revolution in On-Board Communications. Technical report, Volvo Technology Report, 1998.

[6] Y. Coady, G. Kiczales, M. Feeley, and G. Smolyn. Using AspectC to improve the modularity of path-specific customization in operating system code. In *Proceedings of the Joint European Software Engineering Conference (ESEC) and 9th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE-9)*, 2002.

[7] K. R. Dittrich and A. Geppert. *Component Database Systems*, chapter Component Database Systems: Introduction, Foundations, and Overview. Morgan Kaufmann Publishers, 2000.

[8] A. Geppert, S. Scherrer, and K. R. Dittrich. KIDS: Construction of database management systems based on reuse. Technical Report ifi-97.01, Department of Computer Science, University of Zurich, September 1997.

[9] 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.

[10] Developing DataBlade modules for Informix-Universal Server. Informix DataBlade Technology. Informix Corporation, 22 March 2001. Available at http://www.informix.com/datablades/.

[11] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *Proceedings of the ECOOP*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242. Springer-Verlag, 1997.

[12] 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.

[13] Universal data access through OLE DB. OLE DB Technical Materials. OLE DB White Papers, 12 April 2001. Available at http://www.microsoft.com/data/techmat.htm.

[14] D. Nyström. COMET: A Component-Based Real-Time Database for Vehicle Control-Systems. Licentiate Thesis ISBN 91-88834-46-8, Department of Computer Science and Engineering, Mälardalen University, Sweden, May 2003.

[15] D. Nyström, A. Tešanović, M. Nolin, 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*, June 2004.

[16] D. Nyström, A. Tešanović, C. Norström, and J. Hansson. Database Pointers: a Predictable Way of Manipulating Hot Data in Hard Real-Time Systems. In *Proceedings of the 9th International Conference on Real-Time and Embedded Computing Systems and Applications*, pages 623–634, February 2003.

[17] 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.

[18] All your data: The Oracle extensibility architecture. Oracle Technical White Paper. Oracle Corporation. Redwood Shores, CA, February 1999.

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

[20] Polyhedra Plc. http://www.polyhedra.com.

[21] K. Ramamritham. Real-Time Databases. *International Journal of distributed and Parallel Databases*, 1(2):199–226, 1993.

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

[23] O. Spinczyk, A. Gal, and W. Schröder-Preikschat. AspectC++: an aspect-oriented extension to C++. In *Proceedings of the 40th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS Pacific 2002)*, Sydney, Australia, February 2002. Australian Computer Society.

[24] 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.

[25] C. Szyperski. *Component Software - Beyond Object-Oriented Programming*. Addison-Wesley, 1999.

[26] A. Tešanović. Towards aspectual component-based real-time system development. Technical report, Department of Computer Science, Linkping University, June 2003. Licentiate Thesis, ISBN 91-7373-681-3.

[27] A. Tešanović, D. Nyström, J. Hansson, and C. Norström. Embedded Databases for Embedded Real-Time Systems: A Component-Based Approach. Technical Report MRTC Report ISSN 1404-3041 ISRN MDH-MRTC-43/2002-1-SE, Dept. of Computer Engineering, Mälardalen University, January 2002.

[28] A. Tešanović, D. Nyström, J. Hansson, and C. Norström. Integrating symbolic worst-case execution time analysis into aspect-oriented software development. OOPSLA 2002 Workshop on Tools for Aspect-Oriented Software Development, November 2002.

[29] A. Tešanović, D. Nyström, J. Hansson, and C. Norström. Aspect-level WCET analyzer: a tool for automated WCET analysis of a real-time software composed using aspects and components. In *Proceedings of the 3rd International Workshop on Worst-Case Execution Time Analysis (WCET 2003)*, Porto, Portugal, July 2003.

[30] A. Tešanović, D. Nyström, J. Hansson, and C. Norström. Towards aspectual component-based real-time systems development. In *Proceedings of the 9th International Conference on Real-Time and Embedded Computing Systems and Applications (RTCSA'03)*, February 2003.

[31] 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.

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

[33] Xerox Corporation. *The AspectJ Programming Guide*, September 2002. Available at: http://aspectj.org/doc/dist/progguide/index.html.