COMET: a COMponent-based Embedded real-Time database*

Aleksandra Tešanović
Jörgen Hansson
Linköping University
Department of Computer Science
Linköping, Sweden
{alete,jorha}@ida.liu.se

Dag Nyström
Christer Norström
Mälardalen University
Department of Computer Engineering
Västerås, Sweden
{dag.nystrom,christer.norstrom}@mdh.se

Abstract

In this paper we introduce the concept of the tailorable embedded real-time database called COMET. Designing a database to be tailorable, and at the same time satisfy real-time (temporal) requirements and minimal resource (space) requirements of an embedded and a real-time system, require exploiting the concept of components and aspects in embedded real-time database development.

1. Introduction

In the last years the deployment of embedded and real-time systems has increased dramatically [13]. The amount of data that needs to be managed by the real-time systems is increasing, thus requiring an efficient and structured data management. Hence, database functionality is needed to provide support for storage and manipulation of data in real-time and embedded systems, but it must also fulfill requirements both from embedded systems and from real-time systems[1].

Real-time systems are typically constructed out of concurrent programs, called tasks. The correctness of a real-time system depends both on the logical result of the computation, and the time when the results are produced, expressed explicitly as temporal constraints [12]. The most common type of temporal constraint that a real-time system must satisfy is the completion of task deadlines. Moreover, the data in the system is normally associated with temporal constraints, e.g., absolute and temporal validity intervals [10]. Thus, a real-time database must ensure that the data in the database is both logically and temporally consistent.

*This work is supported by ARTES (A network for Real-time and graduate education in Sweden).

[1]We distinguish between embedded and real-time systems, since there are some embedded systems that do not enforce real-time behavior, and there are real-time systems that are not embedded.

In contrast to an application-embedded database hidden inside an application, a device-embedded database is a database that resides in an embedded system. We focus on device-embedded databases, and refer to those as embedded databases. The main objectives for an embedded database are low memory usage, i.e., small memory footprint, portability to different operating system platforms, efficient resource management , e.g., minimization of the CPU usage, and ability to run for long periods of time without administration [9].

Two dimensions are of interest when designing a database for real-time and embedded systems: time and space. This can be further refined to include: (i) functionality vs size trade-offs, (ii) effects of the temporal requirements on the functionality, and (iii) the cost of the production.

Hence, to cope with these challenges we propose a real-time database platform based on the concept of components and aspects: COMET (COMponent-based Embedded real-Time database system). To reduce the cost of production, reuse is the key word, and reuse of components from a component library is the form of reuse that is gaining momentum, e.g., COM, CORBA and JavaBeans. Having aspects in addition to components implies incorporating basic ideas of aspect-oriented programming (AOP) [6] into the database development. Thus, in COMET we distinguish between *functional and aspectual decomposition* of the database system. Functional decomposition is a way of decomposing a database system into components. *Components* are functional units that contain the primary structure of the database system and carry the core functionality of the system. Aspectual decomposition is a way of separating cross-cutting concerns in the system, e.g., code that cannot be encapsulated within one functional unit but is tangled over the entire system. Hence, *aspects* are non-functional units that contain the secondary structure of the database and refer to components and other aspects.

The paper is organized as follows. Motivation for com-

ponentization of an embedded real-time database system is given in section 2. Section 3 presents major challenges for the development of a component-based embedded real-time database system. Related work is discussed in section 4. The paper finishes with summary containing main conclusions and directions to our future research.

2. Motivation

2.1. Embedded and Real-Time Databases

Existing commercial embedded database systems, e.g., Polyhedra, RDM, Velocis, Pervasive.SQL, Berkeley DB, and TimesTen, have different characteristics and are designed with specific applications in mind. They support different data models, e.g., relational vs object-relational model, and different operating system platforms. Moreover, they have different memory requirements and provide different types of interfaces for users to access data in the database. Application developers must carefully choose the embedded database their application requires, and find the balance between required and offered database functionality. Hence, finding the right embedded database is a time consuming, costly and difficult process, often with a lot of compromises. Additionally, the designer is faced with the problem of database evolution, i.e., the database must be able to evolve during the life-time of an embedded system, with respect to new functionality. However, traditional database systems are hard to modify or extend with new required functionality, mainly because of their monolithic structure and the fact that adding functionality results in additional system complexity.

Although a significant amount of research in real-time databases has been done in the past years, it has mainly focussed on various schemes for concurrency control, transaction scheduling, and logging and recovery, and less on configurability of software architectures. Research projects that are building real-time database platforms, such as ART-RTDB [7], BeeHive [14], DeeDS [1] and RODAIN [8], have monolithic structure, and are built for a particular real-time application. Hence, the issue of how to enable development of an embedded database system that can be tailored for different embedded and real-time applications arises.

2.2. Customization by Composition

Having embedded real-time database systems that would allow adding or replacing functionality in its architecture in a component-based manner would be beneficial for several reasons: (i) complexity of the database system and maintenance cost would be reduced; (ii) applications do not have to pay performance and cost penalty for using unneeded functionality, since unnecessary components do not have to be added to a system; and (iii) evolution of a system would be simplified, since new components with new required functionality could be plugged into the system.

Although some major database vendors (e.g., Oracle, Informix, Sybase, and Microsoft) have recognized that component-based development offers significant benefits, their component-based solutions are limited in terms of tailorability with, in most cases, no support for analysis of the composed system. To the best of our knowledge, the only existing research aimed to build completely configurable database management system (DBMS) is KIDS [4]. KIDS introduces a configurable DBMS composed out of components (DBMS subsystems), e.g., object management and transaction management. Customization of this system is improved by having reusable architectures, as well as components stored in a library. KIDS has a well-defined development process, available configuration support and optional analysis tools (which are missing in other component-based databases solutions, e.g., Garlic [3], Navajo [2]). From a real-time point of view all approaches discussed do not enforce real-time behavior. Issues related to embedded systems such as low-resource consumption are not addressed at all.

Tools to support the designer in the composition and analysis of the composed system are essential, and most of the component-based systems discussed do not provide adequate configuration and especially analysis support (vital for real-time systems).

3. COMET

3.1. Methodology

The COMET platform consists of two parts. The first part is a component library, which holds a set of components and aspects. The second part are tools that, based on the application requirements, support the designer when building an embedded database using components and aspects from the library. Our approach consists of extracting relevant database features by studying a number of application case studies in the first phase, followed by an implementation in the next phase, and evaluation of the design tools and database architecture on number of case studies in the final phase.

3.2. Challenges

A starting point towards the platform is to design a component-based database. In this respect, we are faced with the following challenges.

1. How do we encapsulate different database functionalities/services into components suitable for embedded

and real-time systems? This also includes: (i) defining the database component model appropriate for real-time and embedded systems; (ii) specifying component's real-time attributes as aspects such that the composed system is predictable; and (iii) defining rules for connecting components, such that composed system is reliable and fulfills the system's requirements.

2. How do we ensure predictability of the composed database system, with respect to response time, memory usage and CPU utilization? Furthermore, the challenge is to ensure easy integration of the composed database system into the run-time environment of the real-time system ensuring temporal behavior of the system.

Coping with these challenges is a difficult task, for several reasons. First, the COMET database should be suitable for low resource systems, primarily systems with constrained memory and power consumption. Thus, appropriate decomposition of the database functionality should be made, such that the functionality most likely (not) to be needed by most embedded systems is identified and encapsulated into components. Second, the COMET database should ensure predictable transaction execution. Additionally, most embedded systems are implemented in main-memory, thus requiring the COMET database to be a main-memory database.

We illustrate these challenges with an example of data management issues in a class of automotive control applications. Discussion in the example is primarily based on a case study within the project performed at Volvo Construction Equipment Components AB, Sweden, where we analyzed data management in existing real-time systems used to control wheel loaders and articulated haulers.

Example *Vehicle Control Systems*

Typically, control systems in the automotive industry are hard real-time safety-critical systems consisting of several distributed nodes. Each node implements specific functionality and can be viewed as a stand-alone real-time system, e.g., nodes can implement transmission, engine, or instrumental functions. The size of the nodes can vary significantly, from very small nodes, e.g., 32 Kb RAM, to larger nodes, e.g., 64 Kb RAM and 512 Kb Flash. Depending on the functionality of a node and the available memory, different database implementations are needed. For example, in safety-critical nodes tasks are often non-preemptive and scheduled off-line, avoiding concurrency by allowing only one task to be active at any given time. This, in turn, influences functionality of a database in a given node with respect to concurrency control. At this point we need to clarify the relationship between the task and the transaction. In control applications, such as this one, tasks are performing all updates on data. Therefore, from a database point of view, a task performing an update can be treated as a database transaction, as it encapsulates (one or more) transactions. Less critical nodes, having preemptable tasks, would require concurrency control mechanisms. Furthermore, some nodes require critical data to be logged, e.g., warning and errors, and require backups on startup and shutdown, while other nodes only have RAM (main-memory), and do not require non-volatile backup facilities from the database.

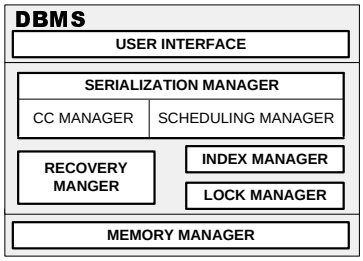

Figure 1. COMET functional decomposition

The discussion on different functionality needed from a database by different nodes could be continued further, but our goal with this short discussion is to show that introducing a component-based database has premise, since it would allow the database to be tailored to suit the needs of a real-time application in each node with respect to memory consumption, concurrency control, recovery, different scheduling techniques, transaction and storage models. This helps to optimize memory consumption in every node and allows databases to be integrated more easily with the run-time environment.

3.3. Functional and Aspectual Decomposition

In order to meet the demands for customization and fine-tuning of the COMET database for specific real-time and embedded applications, the decomposition of a database system must be done both on a functional level (functional decomposition), and on an aspect level (aspectual decomposition). Using aspects as separation of concerns, allows designing functional components that are cross-cut with real-time-specific and database-specific aspects, e.g., real-time aspect and transaction aspect, respectively. For example, a component that controls access to the data in the memory will not only have to manage retrieval and storage of data in a particular memory type (memory type could be viewed as an aspect), but also manage real-time properties of the data objects stored in memory (real-time constraints on data could also be viewed as an aspect).

The functional and aspectual decompositions of COMET are shown in figure 1 and 2, respectively. As-

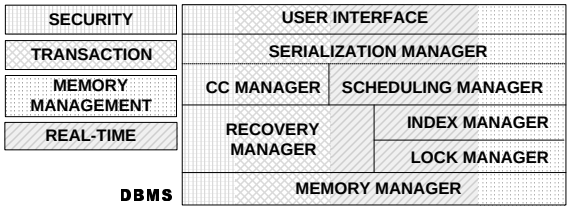

Figure 2. COMET aspectual decomposition on an application level (application aspects)

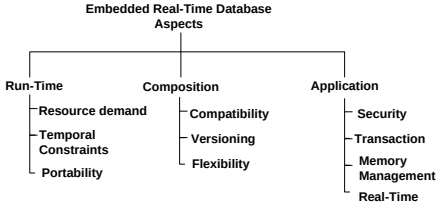

Figure 3. Classification of aspects in an embedded real-time database system

pectual decomposition also implies that components used for system composition are no longer traditional black box components[2], rather they are grey in that we can modify their internal behavior by applying different aspects.

Since aspects are considered to be a property of a system that affects its performance or semantics, and that cross-cuts the system's functionality [6], aspects that can be identified in an embedded real-time database system could be numerous. Thus, we must carefully define both components implementing certain functionality and different aspects that cross-cut these components.

3.4. COMET Functional Decomposition

In the initial design of the COMET we have identified six components (see figure 1): (i) user interface, a component that enables user to access data in the database and is doing query processing; (ii) serialization manager, a component that is in charge of ensuring serialization of transactions, and performs scheduling and concurrency control (CC); (iii) locking manager, a component that deals with locking of data; (iv) index manager, a component that deals with the indexing of the data; (v) recovery manager, a component that is in charged of recovery and logging of data in the database; and (vi) memory manager, a component that allows access to data possibly stored in different memory media.

The principle that lead us to this functional decomposition of the COMET database is primarily the need to have functionally exchangeable units that are loosely coupled, but with strong cohesion, i.e., internal strength.

Somewhat natural is the choice to have a user interface as a component, since different applications may require different ways of accessing data within the system.

In the control applications discussed previously, there are scenarios not requiring sophisticated transaction scheduling and concurrency control, e.g., the hard real-time system allows only one transaction to access the database at a time. Therefore, it would be helpful to have this functionality as a flexible part of the database architecture. Given that scheduling and concurrency control are database functionality that are tightly coupled, we choose to encapsulate the two into one component, a serialization manager. On the other hand, we must decouple these two functionality into two distinct sub-components within the serialization manager to allow flexible exchange of scheduling mechanisms and/or concurrency control mechanisms when applicable.

As COMET stores data in main-memory, there is a need for different recovery and logging techniques, depending on the type of the storage, e.g., non-volatile EEPROM or Flash. Thus, enabling the exchange of different recovery strategies through the recovery manager component would be beneficial for different applications. Similarly, memory management is required to be a flexible part of the architecture since there is a need for a component that would take care of the access to different types of memory. Same reasoning could be applied to decisions on index manager and locking manager, since indexing and locking are performed by the database fairly independent of, for example, serialization and recovery.

3.5. COMET Aspectual Decomposition

In order to classify the complexity of requirements in the database design, and address them in the design of the COMET in a systematic way, we classify aspects as follows (see figure 3):

- application aspects, which are aspects of the database towards the application,
- run-time aspects, which are aspects of the database towards the run-time system, and
- composition aspect, which are aspects that influence composition of the system.

Application aspects can change the internal behavior of components as they cross-cut them in the database system (as shown in figure 2). The application in this context refers

[2]The internal behavior and attributes of the black box component are strongly encapsulated and cannot be changed or modified.

to the embedded and real-time application towards which the database should be tailored. Application aspects can be divided into: memory management, real-time, security and transaction aspects. We view memory management as an application aspect of a database system, since size and allocation of memory influences the system's structure. Additionally, real-time properties are viewed as an application aspect as they influence the overall structure of the database system. Real-time properties could be further divided into categories, e.g., absolute and relative validity. Security is another application aspect that influences the system behavior and structure, e.g., user interface must be able to distinguish users with different security clearance. Depending on the application requirements, ACID (atomicity, consistency, isolation, and durability) properties of a transaction, for some applications, need to be relaxed. Thus, the transaction properties could be viewed as a transaction application aspect.

Run-time aspects are the most critical as they refer to aspects of the monolithic database system that need to be considered when integrating the database system into the run-time environment of a real-time system. Run-time aspects give information which is needed by the run-time system to ensure that integrating a database would not compromise timeliness, or memory consumption of the overall system. Therefore, each component could have declared resource demands in its resource demand aspect, and could have information of its temporal behavior, contained in the temporal constraints aspect, e.g., worst-case execution time (WCET), deadline and period. Additionally, each component must contain information of the platform with which it is compatible, e.g., real-time operating system supported, and other hardware related information. This information is contained as a portability aspect. It is imperative that this information is provided, to ensure predictability of the composed database system, ease the integration of the database into a real-time system and the underlying run-time system, and ensure portability to different hardware and/or software platforms.

Composition aspects do not cross-cut components, rather they are descriptive. Composition aspects describe with which components a component can be combined (compatibility aspect), version of the component (version aspect), and possibilities of extending the component with additional aspects (flexibility aspect).

While conventional AOP has focused predominantly on application aspects, having separation of aspects in different categories eases reasoning about different embedded and real-time related requirements, as well as the composition of the database and its integration into a real-time system. For example, a run-time system can define which (run-time) aspect the database must fulfill, so that proper database components can be chosen from the library, and composed into a monolithic system. This approach offers a significant flexibility, since additional aspect types can be added to components, and therefore, to the monolithic database system, further improving the integration of the database into a run-time system.

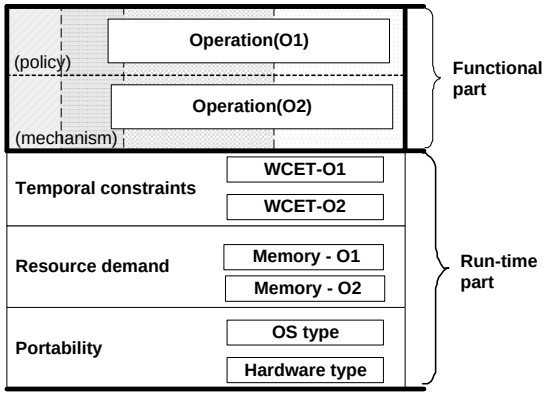

Figure 4. COMET unified component model

3.6. COMET Unified Component Model

Having described components, and categorized aspects that cross-cut or describe a component, we present a more uniform view of a component using a unified component model that unifies functional and aspectual decomposition. A unified component can be viewed as a component colored with aspects, both inside (application aspects), and outside (run-time and composition aspects). Due to space limitations of this paper, in the unified component model we do not consider composition aspects.

A unified component consists of the run-time system dependent part, and the functional part (see figure 4). This simplified view of a uniform component presents application aspects as vertical layers on the functionality, whereas run-time aspects are horizontal parallel layers to the functionality. Every component provides a set of database operations to the real-time application. These operations are contained in the functional layer of the unified component model. If needed, the functional part could be separated further on the policy layer, in the higher level, the mechanism layer, in the lower level. This is useful, for example, in cases when different scheduling (or CC) policies must be exchanged that work on the same mechanisms, e.g., `SetPriority()`.

When a transaction enters the database system, it will execute a number of operations provided by different components. To ensure predictability of transaction execution, WCET (run-time temporal aspect) of each operation needed by the transaction must be known. The total WCET of the

transaction could then be determined by aggregating the WCETs of different operations. WCET information for a specific operation within the component is contained in the run-time layer, as it depends on the operating system used, and the hardware platform on which the operation is to be executed, i.e., it depends on the platform aspects of the component. Of course, the scenario explained is a simplified one. In a more realistic situation we must consider composition aspects, and the complexity of all categories of aspects, i.e., the WCET is only one of the run-time temporal aspects.

4. Related Work

In this section we recognize the research in the area of component-based embedded and real-time systems, and the database and real-time research projects that are using aspects to separate concerns.

The focus in existing component-based real-time systems is enforcement of real-time behavior. In these systems a component is usually mapped to a task, e.g., passive component [13], binary component [5], and port-based object component [15]. Therefore, analysis of real-time components in these solutions addresses the problem of temporal scopes at a component level as task attributes [5, 13, 15]: worst case execution time, release time, deadline.

While there are some projects in the area of real-time systems and database systems that use the aspect-oriented programming paradigm, to the best of our knowledge, there is no project that looks on both these issues. Normally, these projects either focus on providing component-based platforms for development of real-time systems but without database functionality [13], or focus on providing a non-real-time database with limited tailorability using only aspects (i.e., no components) [11].

5. Summary

Designing an embedded database that can be tailored for different real-time applications implies careful balance between application requirement and run-time system requirements. The COMET database aims to balance these requirements, by having exchangeable components encapsulating different functionalities, and use three distinct types of aspects. The design of COMET is flexible, since initial classification of aspects could be, if needed, extended to include other (types of) aspects.

Our future work will focus on validation of the outlined design, and implementation of COMET. This includes: (i) developing a set of components and aspects, (ii) defining rules for composing these components into a real-time database system, and (iii) developing a set of tools to support the designer when composing and analyzing the database system.

References

[1] S. Andler, J. Hansson, J. Eriksson, J.Mellin, M. Berndtsson, and N.Elftring. DeeDS towards a distributed and active real-time database system. *ACM Sigmon Record*, 25, 1996.

[2] H. Bobzin. *Component Database Systems*, chapter The Architecture of a Database System for Mobile and Embedded Devices. Morgan Kaufmann Publishers, 2000.

[3] M. J. Carey, L. M. Haas, P. M. Schwarz, M. Arya, W. F. Cody, R. Fagin, M. Flickner, A. Luniewski, W. Niblack, D. Petkovic, J. T. II, J. H. Williams, and E. L. Wimmers. Towards heterogeneous multimedia information systems: The Garlic approach. In *Proceedings of the RIDE-DOM*, pages 124–131, Taipei, Taiwan, March 1995. IEEE Computer Society.

[4] 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.

[5] D. Isovic, M. Lindgren, and I. Crnkovic. System development with real-time components. In *Proceedings of ECOOP Workshop - Pervasive Component-Based Systems*, France, June 2000.

[6] 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.

[7] Y. Kim, M. Lehr, D. George, and S. Son. A database server for distributed real-time systems: Issues and experiences. In *Proceedings of the 2nd Workshop on Parallel and Distributed Real-Time Systems*, Cancun, Mexico, April 1994.

[8] J. Lindström, T. Niklander, P. Porkka, and K. Raatikainen. A Distributed Real-Time Main-Memory Database for Telecommunication. In *Lecture Notes in Computer Science*, volume 1819.

[9] M. A. Olson. Selecting and implementing an embedded database system. *IEEE Computers*, 33(9):27–34, Sept. 2000.

[10] K. Ramamritham. Real-time databases. *Distributed and Parallel Databases*, 1(2):199–226, 1993.

[11] A. Rashid and E. Pulvermueller. From object-oriented to aspect-oriented databases. In *Proceedings of the DEXA 2000*, volume 1873 of *Lecture Notes in Computer Science*, pages 125–134. Springer-Verlag, 2000.

[12] J. Stankovic. Misconceptions about real-time computing: a serious problem for next-generation systems. *Computer*, 21(10):10–19, October 1988.

[13] J. Stankovic. VEST: A toolset for constructing and analyzing component based operating systems for embedded and real-time systems. Technical Report CS-2000-19, Department of Computer Science, University of Virginia, May 2000.

[14] J. Stankovic, S. Son, and J. Liebeherr. BeeHive: Global multimedia database support for dependable, real-time applications. In *2nd Workshop on Active Real-Time Databases*, 1997.

[15] D. S. Stewart. Designing software components for real-time applications. In *Proceedings of Embedded System Conference*, San Jose, CA, September 2000. Class 408, 428.