An Overview of the DeeDS Real-Time Database Architecture[⋆]

S.F. Andler, J. Hansson, J. Mellin, J. Eriksson and B. Eftring

Department of Computer Science, University of Skövde
Box 408, S-541 28 Skövde, Sweden. Fax: +46 (0)500 464725
Email:{sten, jorgen, jonas, joakim, bengt}@ida.his.se

Abstract. The DeeDS architecture is a distributed real-time systems architecture developed for complex real-time systems by the DeeDS project. A major goal is to evaluate the effectiveness of certain approaches to ensuring predictability and efficiency in a dynamic and complex environment. DeeDS features time-cognizant reactive mechanisms complemented by multi-purpose predictable event monitoring and dynamic deadline scheduling with overload management, as well as support for delayed replication with eventual consistency. These functions are considered orthogonal and only a minimum of functionality included, to be able to control internal complexity. Predictability and efficiency are ensured by allowing all transactions to execute locally without delays due to disk accesses, network communication, or distributed commit processing. Further, a mix of criticality from hard to soft allows hard critical time constraints to be met at the expense of softer constraints, and contingency actions allow handling of transient overloads. To promote predictability, the critical system services, such as the event monitor and the scheduler, may execute on a service processor separate from the application-related functions.

1 Introduction

Many current and future complex real-time systems require distributed processing and sophisticated sharing of an extensive amount of data under critical timing constraints. Major concerns in such a system are dependability and timeliness, where we mainly address the latter for a distributed real-time database environment. Timeliness requires predictability and efficiency, which are both hampered by properties of traditional databases, such as unpredictable and inefficient disk accesses, network communication, transaction scheduling, and global concurrency control. In this paper we describe an approach to making critical real-time database transactions predictable and efficient by removing all dependencies on disk accesses, network communication, and global concurrency control. We also describe techniques required to ensure that proper deadline scheduling of critical transactions and other critical system activities is guaranteed.

[⋆] This work was supported by NUTEK (The Swedish National Board for Industrial and Technical Development) and the Ministry of Education and Science of Sweden.

Many distributed real-time applications have a need for a distributed real-time database system that supports replication of all or large portions of the data. Often, they place hard timing constraints only on access to local data, and less critical timing constraints on propagation of updates to other nodes, even allowing for temporary inconsistencies in the distributed database. For example, in integrated vehicle systems control there are often autonomous nodes controlling individual subsystems. These are handling local data under hard timing constraints (e.g., fuel injection, ignition control), while requiring parameters from other subsystems on a much less critical time scale (e.g., operating data from transmission, environment sensors, ABS brakes). We have identified similar behavior in automated manufacturing and distributed naming services.

The semantics of active (or reactive) database mechanisms closely match the typical processing patterns of real-time systems, which is the reason for including active rules in the definition of the DeeDS (Distributed Active Real-Time Database Systems) architecture. A DeeDS prototype, based on this architecture, is being completed for evaluation while further research is carried out in each of the areas being addressed.

For the purpose of improving system predictability and flexibility, we have configured different functions onto separate processors, depending on whether they are tightly coupled functions invoked by the application, or loosely coupled critical system services not directly invoked by the application. The former category of functions is executed synchronously with the application, while the latter is executed asynchronously. This reduces the difficulty of obtaining predictability by reducing the interdependence of the application and the critical system services, and offers flexibility in configuring the system.

1.1 Background

Real-time systems, monitoring and controlling the external environment, often require storage of extensive amounts of data, which, especially as the storage requirement grows, require predictable and efficient representation, access methods, and query processing. Few, if any, traditional databases can provide these services in a predictable way [20]. A great source of unpredictability, disk accesses, can be avoided by using a *main-memory DBMS* [12].

Active databases have been proposed as an approach to efficient situation monitoring in which the database works as an active component in the system. Most approaches adopt the notion of *event-condition-action* rules (ECA) [10], which is used to model reactive behavior. The modeling of reactive behavior is an important concept in *event-triggered* real-time systems and, hence, this and other current research investigates the combination of real-time and active databases.

Event monitoring consists of observation and analysis, preceded by *instrumentation* [13, 22], which means adding software and, possibly, hardware event generators (sensors) to the system. The *probe-effect* [11] is avoided by letting all event generators remain in the operational system [21], where they may be used for on-line event monitoring.

1.2 Goals

The overall goal of DeeDS is to develop and evaluate an architecture for distributed active real-time database systems. Our strategy to handle the inherent complexity is to treat each function as orthogonal and including only the minimum required functionality.

This overall goal is further subdivided into i) restricting the behavior of the reactive mechanisms, and using contingency plans, in order to guarantee predictability; ii) introducing a deadline and value-driven dynamic heuristic scheduling algorithm, dynamically guaranteeing hard deadlines in the presence of soft deadline transactions and during transient overloads; iii) designing a loosely coupled predictable general purpose composite event monitoring facility, which, in particular, can service the reactive components in DeeDS; and iv) designing a replication and concurrency control mechanism that supports predictable and efficient database access in the distributed environment.

As a means for evaluating this architecture we are currently building a basic prototype (research vehicle). This prototype is being transferred to an industrial partner, and used in an evaluation of the architecture by using a pilot application.

2 DeeDS Architecture

The functions of the DeeDS architecture [1, 17] are split into application-related modules and critical system services. The application-related modules are horizontally layered and tightly coupled via synchronous communication semantics, e.g., procedure call, while the critical system services are loosely coupled to the application-related modules via asynchronous communication semantics, e.g., message passing. All functions in DeeDS are shielded from the underlying operating system by the DeeDS operating system interface (DOI), which is a generic abstraction of a distributed real-time operating system interface (currently mapped to OSE Delta [19]).

The application-related modules are (from the top, see Fig 1) the rule manager, the object store (OBST) [7], and the storage manager (tdbm) [5]. The main rationale for replacing the original storage manager in OBST with tdbm [14, 2] is that we obtain nested transaction semantics [18], which is strongly desirable for active database functionality and exception handling. Another reason is the improved handling of in-memory data structures offered by tdbm, e.g., hashing. Main memory residency is currently provided by a main-memory-resident file system invoked by tdbm.

The critical system services consist of the scheduler and the event monitor. For predictability and flexibility reasons, these services are off-loaded from the processor executing the application-related modules (the application processor), to one or more dedicated processors (the service processors). If the services share a processor they execute in lock-steps by a fixed schedule [17, 16]. The reason for this application and service processor design is that predictability is simpler to obtain while maintaining the required flexibility of dynamic scheduling and

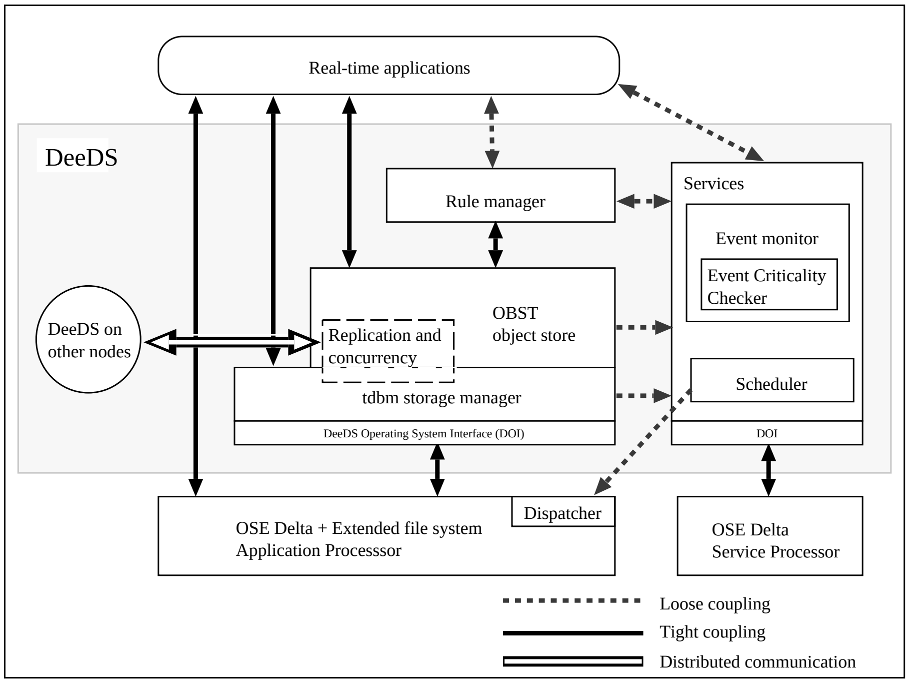
Real-time applications
DeeDS
Rule manager
Services
Event monitor
Event Criticality Checker
DeeDS on other nodes
OBST object store
Replication and concurrency
tdbm storage manager
Scheduler
DeeDS Operating System Interface (DOI)
DOI
Dispatcher
OSE Delta + Extended file system Application Processsor
OSE Delta Service Processor
Loose coupling
Tight coupling
Distributed communication

Fig. 1. The DeeDS architecture

event monitoring compared to a uniprocessor design. That is, the interdependence between the application-related modules, and the critical system services is reduced. In this design, different trade-offs between system requirements, e.g., earliest possible release time vs. maximum transaction load, are easier to evaluate early in a project compared to a uniprocessor solution, in which the execution of the critical system services is not strictly contained. Note, however, that the application-related modules, the application, and critical system services may share one processor, which emulates two or more processors by fixed allocation of processing time.

2.1 Fighting Complexity

As mentioned, the approach we use to limit the impact of internal complexity in DeeDS is to treat each function as orthogonal and only include minimum required functionality. The rule manager offers time cognizant rules and predictable rule execution, while limiting available coupling modes and condition expressions. Critical system services such as a predictable general purpose event monitor and a dynamic scheduler are running on separate processors to limit the impact on executing applications. For updates of distributed data, we support hard local time constraints at the cost of temporary inconsistencies by employing delayed replication, with conflict detection and resolution based on application semantics.

2.2 Rule Management

Since DeeDS is to operate in an environment with temporal constraints, the reactive mechanisms need to be aware of these constraints. Rules in DeeDS are specified using the ECA paradigm. These rules have been extended with temporal attributes, such as deadlines and execution time on actions. These attributes are mainly used by the scheduler, but also by the rule manager and event monitor in determining the criticality of rules and events.

The task of the rule manager is to store rules, retrieve rules on event occurrence, perform condition evaluation, and submit actions and necessary information to the scheduler.

Since DeeDS provides a real-time database service, the rule manager must, like the rest of the database system, be predictable and efficient. Therefore, rules must be retrieved and executed in a predictable and efficient manner, including event detection, condition evaluation, and action execution. Since event detection and action execution is performed by the event monitor and scheduler, the timeliness of services provided by the rule manager is dependent on the guarantees given by these modules, such as performing event monitoring in a known and bounded time and meeting deadlines, as well as the predictability and efficiency of the methods used by the rule manager itself to retrieve rules and evaluate conditions.

Moreover, triggered actions themselves must not bring unpredictability to the system. One method to achieve this is to only allow triggered actions to execute in separate top-level transactions, thereby not affecting the calculated worst-case execution time of the transaction in which the triggering event was raised [8].

2.3 Event Monitoring

The event monitor detects foreseeable events of specified event types (describing event patterns) in a predictable and efficient way. That is, when a critical type is detected, the event monitor disseminates the information about this event to the subscribers in a known and bounded time. The types can be categorized as primitive or composite, depending on whether they are basic events of interest, or composed of other primitive and composite events. All events, even composites, are considered to be atomic and instantaneous from the viewpoint of the subscribers. All events carry the following information: event type, time of occurrence, time of detection, and scope of occurrence. It is important to differentiate between detection and occurrence during event showers. The composite event types are specified using operators, which are conjunction, disjunction, sequence, and bounded closure operators. These types are translated into an event graph [9] that is used in performing the actual event composition during event monitoring. The event composition policies [9, 6] investigated are restricted to the recent, which is typical for oversampled sensors, and chronicle, which is useful in matching corresponding events.

The event criticality checker, a part of the event monitor, continuously checks and computes the queue of incoming primitive events. The task of this checker is to guarantee the response of critical events at the cost of non-critical events. Critical events are associated with transactions with critical (hard critical) timing constraints, while non-critical events are associated with transactions with non-critical (not hard critical) timing constraints (see Dynamic Scheduling). That is, the timing constraint of the associated transaction is used to compute the priority of the event. However, if event criticality is used without caution all event types may become critical [3].

Monitoring the environment is an issue that is investigated in the DeeDS architecture. The motivation is that the event monitor can serve as a filter, in which events can be handled systematically, and the load on the system can be reduced. However, to perform event composition on the environment implies that this is done outside the scope of transactions. The reason is that doing event composition inside the scope of transactions of events stemming from the environment, can either lead to long-running transactions, or to an unknown amount of transactions blocking each other in an unpredictable way. Thus, there is a need to investigate how to monitor composite events outside the scope of transactions.

2.4 Dynamic Scheduling

In DeeDS dynamic scheduling is used, i.e., scheduling decisions are made at run-time.

The transaction workload is complex due to the multiple importance levels of the transactions and their aperiodic nature. Transactions are characterized along the following dimensions: i) periodicity of appearance in the system, i.e., periodic, sporadic or aperiodic; and ii) criticality of timing constraints, i.e., hard critical, hard essential, firm or soft deadlines[1] where the timing constraints are expressed with value functions in parameterized linear form. Such parameterization reduces storage requirements compared to arbitrary value functions [15], and reduces computational cost associated with calculating the value at a point in time. Moreover, transactions are considered as autonomous executable preemptable entities without precedence relationships. A priori knowledge about the worst-case execution time of the transactions exist.

Informally, a transient overload occurs when the execution time requirements of a set of transactions exceed the available processing time, potentially causing deadlines to be missed. DeeDS uses two overload resolution strategies, namely, i) dropping transactions with firm and soft deadline transactions in a controlled manner without jeopardizing the timeliness of the critical transactions; and ii)

[1] In order to ensure predictability and schedulability, hard critical transactions must be periodic or sporadic.

replacing critical transactions and executing their contingency action. Contingency actions, which should be seen as autonomous transactions, have significantly less computational requirements than the original transactions. However, invoking a contingency plan will impose a penalty on the overall transaction utility given to the system. The underlying workload assumption is that there exists a feasible schedule for the set of transactions with critical deadlines based upon the known worst-case execution of the contingency plans. Hence, the set of contingency actions and their schedulability is analyzed and guaranteed off-line.

2.5 Replication and Concurrency Control

In order to enforce predictability in DeeDS, a number of design decisions have been made to eliminate unpredictable delays. The largest such delays are due to disk accesses to retrieve and update database values, network communication to access or modify distributed data, and distributed commit processing enforced by global concurrency control of distributed transactions. First, to eliminate delays due to disk accesses, the database is main memory-resident. Second, to eliminate delays due to network communication, the database is fully replicated, i.e., each node has a copy of the entire database. Third, to eliminate delays due to distributed commit processing, transactions are allowed to commit locally and updates of the database are replicated as-soon-as-possible (ASAP). This implies that temporary inconsistencies may arise and must be tolerated by applications [4], until the system eventually becomes consistent again (ASAP, or in bounded time if requirements so dictate).

Together, these three design decisions ensure that all accesses and modifications of the database can be made directly in local main memory. Changes are replicated to other nodes (ASAP or bounded replication) upon commit of a transaction by i) propagating the changes to all other nodes, ii) detecting any possible conflict between updates of concurrent transactions at different nodes, and iii) resolving any conflict as specified by the application programmer. It should be noted that each transaction runs locally in a single node, and that pessimistic concurrency control (supported by OBST/tdbm) is used to prevent conflicts within each node.

The assumption of main memory residency, full replication, and eventual consistency must apply to all data accessed by transactions with hard critical timing constraints, but can be relaxed in varying degrees for transactions of lesser criticality, such as hard essential, firm, and soft timing constraints. For example, main memory residency of hard critical data does not preclude other parts of the database from being supported by virtual memory. Also, actual full replication is not necessary for segments of the database never accessed or modified from certain nodes, as long as the semantics or underlying algorithms of the database do not change as a result of this so-called virtual full replication.

3 Summary

A distributed, active, real-time database systems architecture has been designed for use in research and for experimental evaluation.

Predictability is enforced by applying the following constraints to the system and to the applications: The database is fully replicated and main-memory resident, thus avoiding network and disk delays on all accesses; application processing is only dependent on operations at a single node, thereby avoiding problems of global commit and global deadlocks; reactive mechanisms are restricted in their behavior to allow predictable rule execution; and finally, scheduling and event monitoring (including event composition) are processed on a separate processor. Furthermore, a rule specification language has been developed which includes the ability to specify timing constraints.

Dynamic preemptive transaction scheduling is used, including mixtures of hard, firm and soft transactions. Overload is handled by discarding transactions with soft and firm deadlines and using contingency plans for hard deadline transactions. The event monitor detects event patterns and disseminates events to subscribers in a predictable and efficient way.

References

1. S. Andler, J. Hansson, J. Eriksson, J. Mellin, M. Berndtsson, and B. Eftring. DeeDS towards a distributed active and real-time database system. *Special Issue on Real Time Data Base Systems, SIGMOD Record*, 25(1), March 1996.
2. O. Bergström. Instrumentation of an extended tdbm in a preemptive environment. Third year final project report. Dept. Comp. Sci, University of Skövde, Sweden, March 1996.
3. M. Berndtsson and J. Hansson. Issues in active real-time databases. In M. Berndtsson and J. Hansson, editors, *Active and Real-Time Database System (ARTDB-95)*. Springer-Verlag, 1995.
4. A. D. Birrell, R. Levin, R. M. Needham, and M. D. Schroeder. Grapevine: An exercise in distributed computing. *Comm. of the ACM*, 25(4):260–274, 1982.
5. B. Brachman. TDBM: A dbm library with atomic transactions. In *Summer '92 USENIX*, pages 63–80, June 1992.
6. A. P. Buchmann. Active object systems. In A. Dogac, M. T. Ozsu, A. Biliris, and T. Sellis, editors, *Advances in Object-Oriented Database Systems*, pages 201–224. Springer-Verlag, 1994.
7. E. Casais, M. Ranft, B. Schiefer, D. Theobald, and W. Zimmer. OBST—an overview. Technical Report FZI.039.1, Forschungszentrum Informatik (FZI), Karlsruhe, Germany, 1992.
8. S. Chakravarthy, B. Blaustein, A. P. Buchmann, M. Carey, U. Dayal, D. Goldhirsch, M. Hsu, R. Jauhuri, R. Ladin, M. Livny, D. McCarthy, R. McKee, and A. Rosenthal. HiPAC: A research project in active time-constrained database management. Technical Report XAIT-89-02, Xerox Advanced Information Technology, July 1989.
9. S. Chakravarthy and D. Mishra. Snoop: An event specification language for active databases. *Knowledge and Data Engineering*, 13(3), October 1994.

10. U. Dayal, B. Blaustein, A. Buchmann, S. Chakravarthy, M. Hsu, R. Ladin, D. McCarty, A. Rosenthal, S. Sarin, M. J. Carey, M. Livny, and R. Jauharu. The HiPAC project: Combining active databases and timing constraints. *ACM Sigmod Record*, 17(1), March 1988.
11. J. Gait. A debugger for concurrent programs. *Software-Practice And Experience*, 15(6), June 1985.
12. H. Garcia-Molina and K. Salem. Main memory database systems: An overview. *IEEE Trans. on Knowledge and Data Engineering*, 4(6):509–516, December 1992.
13. D. Haban and D. Wybranietz. A hybrid monitor for behavior and performance analysis of distributed systems. *IEEE Trans. on Software Engineering*, 16(2):197–211, February 1990.
14. M. Johansson. A storage manager for an experimental real-time database platform. Master's thesis, University of Skövde, Dept of Computer Science, 1994.
15. C. Douglas Locke. *Best-Effort Decision Making for Real-Time Scheduling*. PhD thesis, Department of Computer Science, Carnegie-Mellon University, May 1986. Technical Report, CMU-CS-86-134.
16. J. Mellin, J. Hansson, and S. F. Andler. Refining design constraints using a system services model of a real-time DBMS. In *Proc. 1st Int'l Workshop on Real-Time Databases*, pages 84–91, Newport Beach, Carlifornia, March 1996.
17. J. Mellin, J. Hansson, and S. F. Andler. Refining timing constraints of application in deeds. In S. H. Son A. Bestavros, K-J Lin, editor, *Real-Time Database Systems: Issues and Applications*, pages 325–343. Kluwer, 1997.
18. J. Eliot B. Moss. *Nested Transactions: An Approach to Reliable Distributed Computing*. MIT Press, 1985.
19. Ose delta soft kernel R1.0 getting started & user's guide. Part of OSE Delta distribution, ENEA DATA AB, OSE Support Group, Box 232, S-183 23 Täby, Sweden, 1995.
20. K. Ramamritham. Real-time databases. *Distributed and Parallel Databases*, 1:199–226, 1993.
21. W. Schütz. Fundamental issues in testing distributed real-time systems. *Real-Time Systems*, 7(2):129–157, September 1994.
22. R. Snodgrass. A relational approach to monitoring complex systems. *ACM Transactions on Computer Systems*, 6(2), May 1988.