DESIGN ISSUES AND EXPERIMENTAL DATABASE ARCHITECTURE FOR TELECOMMUNICATIONS

Juha Taina and Kimmo Raatikainen
University of Helsinki, Department of Computer Science
P.O.Box 26 (Teollisuuskatu 23)
FIN-00014 University of Helsinki, FINLAND
e-mail: {juha.taina,kimmo.raatikainen}@cs.helsinki.fi
fax: + 358 0708 44441

Abstract

Databases in telecommunications have become an important research area in recent years. The persistent and temporal information needed in operations and management of the telecommunication networks and services will be in databases. The current IN Recommendations of ITU-T (Q.1200 Series) imply that real-time transaction processing capabilities should be provided. We examine the design issues of real-time transaction processing in database architectures that can be integrated into any teleoperator's service provision architecture. We also introduce an experimental database architecture which is a real-time distributed object-oriented database architecture. The architecture can be used to implement the Service Control Function in the Intelligent Network and other database services needed in telecommunications.

1. Introduction

Databases will, already in the near future, have an important role in telecommunication. The databases will contain the persistent and temporal information needed in operations and management of the telecommunication networks and services. Such databases exhibit stringent reliability, availability, and performance requirements. Thousands of retrievals must be delivered in a second. Allowed downtime is only a few seconds in a year. The globalization of services implies that databases managed by different operators must cooperate. The interoperability requires that the database systems must be open, must use standardized protocols, must be protected against misuse and intruders, and must have a common logical view of information.

In this paper we examine some fundamental design issues that affect the feasible architectures of database systems which can be integrated into teleoperator's service provision architecture. We also introduce a database architecture for telecommunication use and its prototype. Our premises are primarily derived from the *Intelligent Network Long-Term Architecture* (IN LTA) framework as described in Recommendation Q.1201 [ITU 1993b]. The key premises are compatibility with the OSI *Reference Model of Open Distributed Processing* (RM-ODP) [ITU 1995] and with the *Telecommunications Management Network* (TMN) [ITU 1993a]. Since both RM-ODP and TMN are based on object-oriented modeling, an object-oriented database is our first presumption. Our second presumption is scalability which implies that the database system may be distributed.

The rest of the paper is organized as follows. In Section 2 we briefly discuss our premises — the ways how the IN architecture, together with the compatibility with RM-ODP and TMN, affects the database architecture. In addition, we summarize our assumptions about the future database needs in telecommunication and summarize requirements for the database architecture. In Section

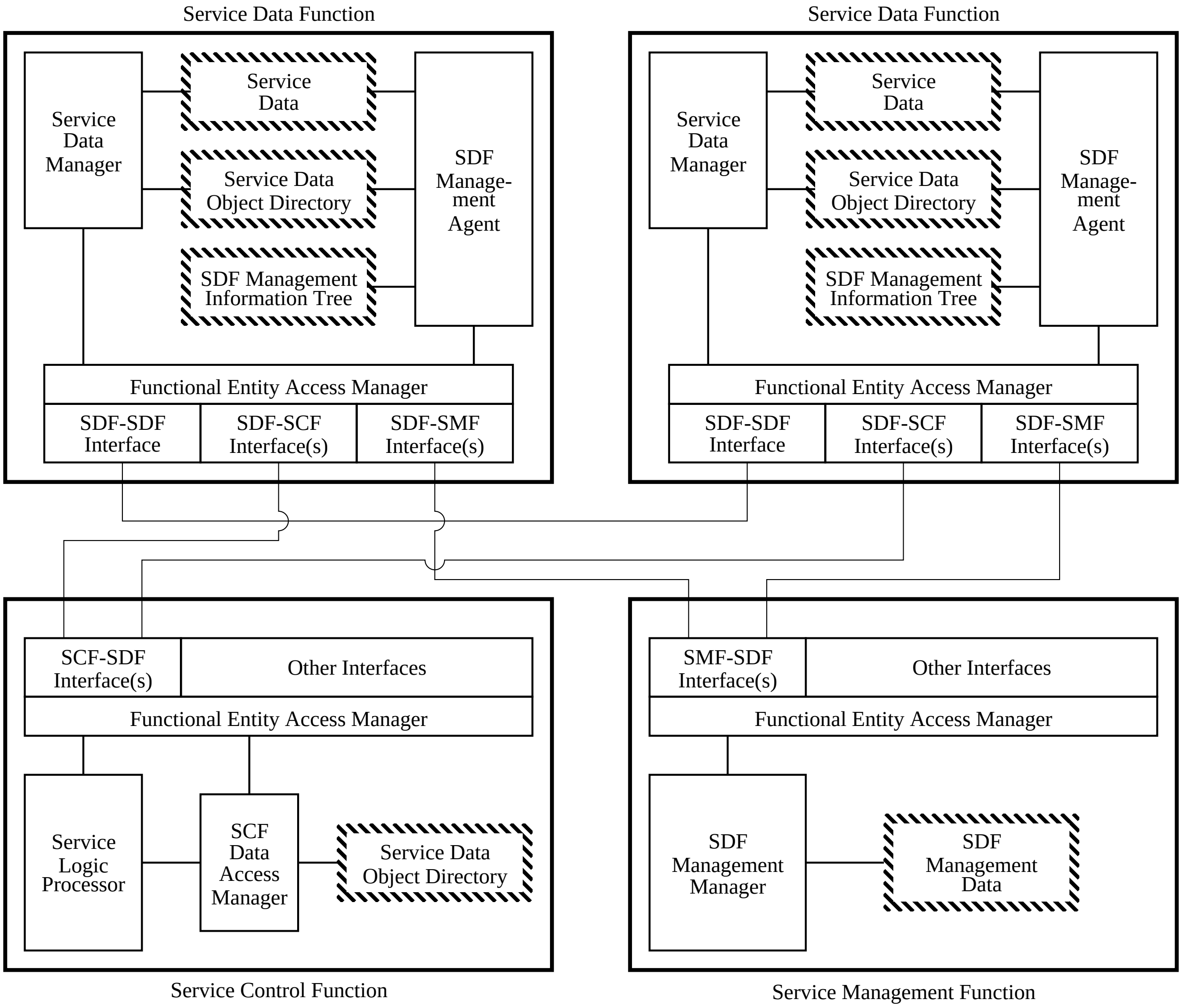

Figure 1: Elements of Database Servers and Clients in the IN Functional Architecture

3 we examine the design issues in real-time databases. In Section 4 we give an overview of the database architecture and of the prototype architecture developed in the Darfin[1]-project.

2. External Interfaces and Design Constraints

In addition to the current ITU-T recommendations in Q.1200 Series (*Intelligent Networks*), the Basic Reference Model of ODP and the concepts of TMN create the architectural framework into which the database system is to be embedded. The architectural framework sets the requirements for external interfaces that the database system may have to support.

2.1 IN Architectural Framework

The *distributed functional plane* in the IN architecture is described in terms of *functional entities*. In that model the database services are available in the *Service Data Function* (SDF). The

[1]Research project *Database ARchitecture For Intelligent Networks* funded by Telecom Finland

database users (clients of SDF) are *Service Control Functions* (SDFs), *Service Management Functions* (SMFs), and other SDFs. Figure 1 outlines the part of the IN functional architecture involving database services. The internal structures of SCF and SDF in the figure are based on the figures 4-19 and 4-20 in ITU-T Recommendation Q.1214 [ITU 1994b], respectively. The outlined internal structure of the SMF and its impact on the SDF are based on the principles of OSI Management [ITU 1991–4]. Detailed description of database access in the IN, especially in the Capability Set 1 (CS-1), can be found in Raatikainen [1995].

2.2 Basic Reference Model of Open Distributed Processing

Open Distributed Processing is a joint standardization initiative by ISO and ITU-T. The Basic Reference Model of ODP [ITU 1995] is a framework for standardization of ODP. The objective of RM-ODP is to create an architecture that supports distribution, interworking, interoperability, and portability. All these are important goals for future telecommunication.

The engineering viewpoint of RM-ODP defines a model for the infrastructure of a distributed system. The model, in turn, can be viewed as an abstract platform for ODP applications. The fundamental entities described in the engineering viewpoint are basic engineering objects, infrastructure objects, and channels. A channel provides the communication mechanism. In addition, the transparency functions are contained or controlled by a channel. The compatibility with RM-ODP requires that the database system must be accessible through an ODP-channel.

2.3 Telecommunication Management Network

Telecommunication Management Network is a generic architecture to be used for all kinds of management services. TMN is based on the principles of the OSI Management as specified in the Recommendations in the X.700 Series [ITU 1991–4].

The fundamental idea in the OSI Management is that the knowledge representing the information used for management is separated from the functional modules performing the management actions. OSI Management is based on interactions between management applications that can take the roles of *manager* and *agent*. The interactions that take place are abstracted in terms of *management operations* and *notifications*. Management activities are effected through the manipulation of *managed objects* (MOs).

An agent manages the MOs within its local system environment. It performs management operations on MOs as a consequence of management operations issued by a manager. An agent may also forward notifications emitted by MOs to a manager. The agent maintains a part of the *Management Information Tree* (MIT), which is a dynamic database. The MIT contains instances of MOs organized as a hierarchical database tree.

Agents and managers exchange information using the services of *Common Management Information Service Element* (CMISE). CMISE uses the *Common Management Information Protocol* (CMIP) and utilizes the services of *Association Control Service Element* (ACSE) and *Remote Operations Service Element* (ROSE). In brief, the principles of OSI Management (and TMN) require that the database system contains the functionality of an OSI Management Agent.

2.4 Anticipated Database Needs

In Raatikainen [1994] we have identified five basic types of database operations needed in services and service features of IN Capability Set 1:

1) Retrieval of structured objects from persistent subscriber data objects. Some of the retrievals trigger a later update. For example, if Freephone service provides call routing statistics to the subscriber, an update is necessary.

2) User management actions that modify persistent subscriber data objects. These actions must be protected against unauthorized use. Some updates may also have to be verified against other criteria. For example, the subscriber must have the possibility to cancel or prohibit call forwarding to his or her number.
3) Verification of Personal Identification Number. For security reasons the PIN verification must be done in the database system when services are globalized. The actual PIN cannot be delivered into a foreign network.
4) Writing sequential log records.
5) Mass calling and Televoting. They need high-volume small updates that must be serialized. A special aspect of these updates is that they can be done in large blocks.

In addition to the IN services, future databases must support mobility and management. From a database point of view, the most challenging aspect of mobility is the maintenance of location information. This contradicts the traditional claim that most database operations needed in telecommunication services are reads. The management implies a twofold functionality. Firstly, the database and its elements are managed objects. Secondly, a management agent needs an efficient object-oriented database to maintain the Management Information Base.

2.5 Requirements for Database Architecture

In Taina [1995] we have identified the key requirements for an IN database architecture. The requirements that we identified are *data distribution, data replication, object orientation, object directories, multiple application interfaces, fault tolerance,* and *real-time transactions.*

Data distribution: A set of SDFs may cooperate to accomplish database tasks. An SDF may request information from another SDF in case needed information is not locally available. This implies data distribution among several SDFs. It is possible to implement the underlying database architecture without data distribution. In such an implementation all applications use a single database server. This differentiates logical data distribution between SDFs from physical data distribution among database nodes. The former is necessary, the latter is a database architecture decision. We have chosen a distributed database architecture. Our assumptions are that few SDF requests must consult other SDFs and that the request distribution between different SDFs is relatively uniform. Thus using a distributed architecture allows several transactions to be run in parallel on several database nodes and give better throughput. In a centralized architecture the database transaction processing can become a bottleneck. However, if the principle assumptions of few distributed operations and uniform request distribution between different SDFs turn out to be wrong then the distribution control itself may become a severe bottleneck. In such a case using a single database node for or IN services is a reliable alternative.

Data replication: In ITU [1994a] it is stated that the SDF contains customer and network data for real time access. This implies that the underlying database architecture must be able to answer requests in real time. We believe that currently most distributed operations are reads. Therefore, replication is an effective way to speed up distributed operations because it gives better throughput by allowing read operations to run parallel on several nodes compared to the extra overhead of replicated writes. For instance, service information is a good candidate for replication because usually all SDFs share the same image of different service profiles.

Object orientation: In ITU [1993b] it is stated that the best long term architecture for IN is probably object oriented.

Object directories: In ITU [1994b] the SCF holds a Service Data Object Directory. A similar concept is defined in SDF function model. This implies that the SDF architecture must support object directories.

Multiple application interfaces: Currently the exact interfaces to SDF are still evolving. Currently the only defined interface is CS-1 IN Application Protocol (INAP) that is defined in ITU [1994c]. According to Chatras and Gallant [1994] the next refinement of the SCF-SDF interface will be based on a subset of the Directory Access Protocol (DAP) in CCITT X.500 directory standard. Other possible interfaces to the SDF are X.700 management interface for management functions, and Open Distributed Processing (ODP) interface for distributed operations.

Fault tolerance: Real time access implies that data must be continuously available. The result of the implications is that the database architecture must be fault tolerant to a certain extent. Currently we believe that the maximum down time of an SDF must be at most seconds in a year. Yet we are not certain how much tolerance a customer has when a system failure occurs. The actual allowed down time may be minutes instead of seconds. Nevertheless a fault tolerant database architecture is a necessity.

Real-time transactions: A real time transaction has an explicit deadline that the transaction must meet. Although a real time access to data in SDF, as stated in ITU [1994a], does not directly imply that the underlying database architecture must support real time transactions, we believe that the most convenient way to support real time access to data is to use a database architecture that is specialized for real time transactions. A convenient database architecture allows time limits to transactions and to define actions if the deadlines are not met.

The list above shows the general outline of our Darfin DBMS architecture. It is distributed, object oriented, and supports real time transactions. Thus Darfin DBMS can be called a *Real Time Distributed Object Oriented Database Management System.*

Currently we are implementing a prototype database architecture that can answer some of the requirements listed above. With the prototype we hope to gain insight of the data access nature in IN environment. We are especially interested in statistics of the database usage, combining real time transactions and object orientation, and learning implementation methods for SDFs and other network elements using the database architecture. The prototype architecture is called Darfin Proto.

3. Real-Time Databases

Recommendation Q.1204 (*Intelligent Network Distributed Functional Plane Architecture*) [ITU 1994a] specifies that the *Service Data Function* (SDF) provides real-time access to customer and network data. This statement raises the question whether a real-time database is needed. Recommendation Q.1211 (*Introduction to Intelligent Network Capability Set 1*) [ITU 1993c] augments the functional specification of SDF by requiring that SDF provides consistency checks on data. This raises the fundamental question whether it is possible to lower serialization requirements in order to gain better real-time transaction processing capabilities.

3.1 Principles of the Real-Time Database System

According to Haritsa *et al.* [1990] *Real-Time Database System* (RT-DBS) is a transaction processing system that tries to satisfy the explicit timing constraints associated with each incoming transaction. The timing constraint is usually given as a *deadline*. The ultimate goal of an RT-DBS is to maximize the fraction of transactions which meet their deadlines. This is in contrast to the goal of traditional database systems (DBMSs) which is to minimize the average response time.

At any time, all transactions in an RT-DBS can be divided into two classes: *feasible* transactions and *late* transactions. A feasible transaction still has a possibility to meet its deadline. A late transaction has either already missed its deadline or cannot any more meet its deadline. The RT-DBS executes feasible transactions until they complete in time or until the system detects that they can no more meet their deadlines. The policy of how late transactions are handled depends on the application needs. The two primary policies are: 1) to continue the execution with a low priority and 2) to abort the transaction without restart.

Real-time systems are usually classified into three categories according to the effects of missing a transaction deadline. *Hard* deadline transactions are those which may result in a catastrophe if the deadline is missed. A hard real-time system must guarantee that any transaction never misses its deadline. *Firm* deadline transactions are those the results of which are not any more of any value if the deadline is missed but have no serious consequences. *Soft* deadline transactions are those the results of which gradually decrease their value when the deadlines are missed.

The deadlines raise several difficult questions. How are deadlines assigned? Are deadlines firm or soft? It must be remembered that a transaction accessing the database is only a subtask in a telecommunication service. Unfortunately *subtask deadline assignment* is still almost an unexamined problem [see e.g. Kao and Garcia-Molina 1993]. The primary implication of transactions being subtasks is that the task can still meet (miss) its deadline even if the transaction misses (meets) its deadline.

Since the main goal of a firm or soft RT-DBS is to maximize the fraction of transactions that meet their deadlines, a key issue in transaction processing is *predictability* of transactions' execution and turnaround times. In a database system, data and hardware resource conflicts are an important source of unpredictability. Another significant source is the unpredictability of execution time caused by execution sequence depending on data values and by disk operations. Distributed databases have additional problems due to communication delays and site failures.

3.2 Designing an Real-Time Database System

One of the primary difficulties in designing an RT-DBS is the fact that the concurrency control of data access must be combined with timing constraints. The two primary approaches to arrange the concurrency control in database systems are based on *locking* and on the *optimistic approach*. Most commercial DBMSs are based on *two-phase locking* (2PL) [Eswaran *et al.* 1976]. This is due to the fact that under most operating circumstances locking algorithms give shorter response times than optimistic algorithms [see e.g. Agrawal *et al.* 1987].

Recent performance studies [Abbott and Garcia-Molina 1992, Haritsa *et al.* 1990, Huang *et al.* 1992, and Lin and Son 1990, among many others] indicate that the results obtained from traditional DBMSs do not hold in RT-DBSs. The relative performance of concurrency control algorithms in real-time database systems is heavily affected by several factors including policy with late transactions, a priori knowledge of transaction resource requirements, and the availability of resources. This leaves the floor open to several design issues that must be evaluated.

3.2.1 Design Issues in Concurrency Control

The major design issues in concurrency control are conflict detection and conflict resolution. In addition, the run policy is an important issue. Below we discuss these issues separately although they are not independent. We also name several mechanisms proposed in the literature.

Conflict Detection. Conflicts in data granule access that violates the correctness criteria of a transaction can be detected either before the granule access (*pessimistic approach*) or after the granule access (*optimistic approach*). In the optimistic approach the correctness is later verified at the certification time. The mechanisms proposed to be used in the detection include *locks*, *time stamps*, and *serialization graphs*.

When a lock-based scheme is used, a lock must be obtained before an access is allowed. In pessimistic approaches, such as various 2PL schemes, the lock is either shared (*read*) or exclusive (*write*). When an optimistic approach is based on locks, there are *weak* locks as well as normal (*strong*) locks. When a transaction has a weak lock, either read or write lock, on a granule, it only indicates that the granule is accessed. A weak lock conflicts with a strong write lock but is compatible with other weak locks (both read and write) and with strong read locks. Typically, the weak locks are converted to strong locks at the beginning of the certification time.

In some situations two different kinds of write locks may be profitable: an *update lock* (ordinary write lock) and a *replace lock* (blind write lock). A blind write lock is used when the locked item is updated only from an external source at predefined intervals. This kind of special situation occurs in mobile databases when subscriber location information is updated. Two *blind write* operations do not conflict with each other, since the old value is obsolete as soon as the new value arrives, and the write operations occur only on isolated objects. However, the operations do conflict with read operations.

Conflict Resolution. A conflict resolution mechanism is invoked once a conflict is detected. In the resolution at least one transaction is penalized through appropriate actions selected by the conflict resolution mechanism. The actions most commonly used are *blocking* (wait), *abort* (restart), *multiversioning*, and *dynamically readjusting the serialization order* (delayed commit or abort). It should be noted that the resolution mechanism is not independent from the detection mechanism. For example, if the conflict is detected after the granule access, the action must be abort.

Several conflict resolution mechanisms have been proposed in the literature. Variants of the 2PL include *no wait* [Tay *et al.* 1985], *wound-wait* [Rosenkrantz *et al.* 1978], *wait-die* [Rosenkrantz *et al.* 1978], *running priority* [Franaszek and Robinson 1985], *wait depth limited* [Franaszek *et al.* 1992], and *delayed abort* [Yu 1990]. In addition to the pure *optimistic concurrency control* (OCC) *broadcast OCC* [Kung and Robinson 1981] and a combination of pure and broadcast OCC [Yu *et al.* 1993] have been proposed. *Locking with deferred blocking* [Yu and Dias 1993] is a scheme that combines 2PL and OCC. Multiversioning opens a wide variety of conflict resolution mechanisms. A detailed comparison can be found in [Wu *et al.* 1993].

Run Policy. When abort is used as an action in the conflict resolution, a transaction may need to be restarted several times. The restarts create an effect known as *buffer retention* [Yu and Dias 1992]. In the first run it may be profitable to run a transaction that will be aborted to end so that all granules accessed will be in the buffer and the CC manager learns all the locks the transaction needs.

3.2.2 Integrating Concurrency Control and Real-Time Scheduling

There are four major problem areas when a real-time transaction scheduler is to be developed: *priority inversion*, *scheduling priority*, *overload management*, and *IO scheduling*.

Priority Inversion. A phenomenon called *priority inversion* arises when a low-priority transaction blocks a high-priority transaction due to the concurrency control. At least three different approaches have been proposed to solve the problem. In *priority inheritance* [Sha *et al.* 1990] the low-priority transaction inherits the priority of the high-priority transaction. In the *priority ceiling protocol* [Sha *et al.* 1991] blocking time of the high-priority transaction is bounded. The *dynamic priority ceiling protocol* [Chen and Lin 1990] is a modification of the previous based on earlist deadline first instead of fixed external priorities.

Scheduling Priority. *Earlist Deadline First* (EDF) is most widely used in the experimental real-time database systems as the scheduling priority. It can usually minimize the number of late transactions but not when the system is highly loaded [Liu and Layland 1973]. Another drawback in the EDF is that large transactions are discriminated. Alternatives to the EDF include *Least Slack First* (LSF), *fixed priorities*, *criticality of transaction*, *weighted priority scheduling*: see e.g. Abbott and Garcia-Molina [1992], Buchmann *et al.* [1989], Sha *et al.* [1991], Huang *et al.* [1989], Huang *et al.* [1991]. The fundamental design criteria are the heterogenity of transactions and transaction characteristics available a priori.

Overload Management. If the EDF scheduling is used, then occasional periods of high load can significantly decrease the fraction of transactions that complete in time. In order to cope with the overload situations the scheduling needs an overload management policy. In Haritsa *et al.* [1991] an *adaptive earlist deadline* (AED) scheduling scheme was proposed. In AED, arriving transactions are placed in a HIT group or a MISS group depending on the system condition. In the HIT group the transactions are treated in the normal way but in the MISS group the transactions get low priority. In Pang *et al.* [1992] an *adaptive earlist virtual deadline* (AEVD) scheduling scheme was proposed. AEVD is an extension of AED that tries to take into account the fairness issue in an overloaded system.

IO Scheduling. When the primary copy of the database is on the disk, IO operations are extremely important for the RT-DBS. Traditional disk scheduling algorithms should be replaced by novel ones as in Sprunt *et al.* [1988] and in Abbott and Garcia-Molina [1990]. When a large fraction of the database can be held in the main memory buffers, the situation is completely different. We believe that in such cases hybrid databases (see the next section) should be used.

4. Prototype Database Architecture

Both Darfin DBMS and Darfin Proto consist of a set of database nodes that interact with each other. Every database node is an autonomous database and can be used alone.

A database node communicates with a set of applications, a mirror node, and the other database nodes in the distributed system. Applications, database nodes, and mirror nodes may be geographically distributed (see Figure 2).

A single application may communicate with several database nodes or only with a single node. If the application uses services of a single database node, then that node is the entry point to the whole distributed database. Then the actual distribution is internal: From the application's point of view the database architecture is not distributed. If the application uses services of several database nodes, then the application is responsible for taking care of distributed operations. For instance, an application may send a request to a single database node to add a new service to every customer, or it may send a request to every database node to update local information.

Every database node has a mirror node. All updates that are sent to the database node are also sent to the mirror node. In case of a system failure in the database node the mirror node changes its status into the database node. The architecture decision supports well fault tolerance but it may waste resources. All data is replicated at least to the mirror node. Usually this is not necessary. For instance, it is not necessary to store all IN database usage statistics into the mirror node.

4.1 Database Node Architecture

The architecture of a single database node consists of an Object-Oriented Database Management System (OODBMS), User Request Interpreter Subsystem (URIS), Distributed Database Subsystem

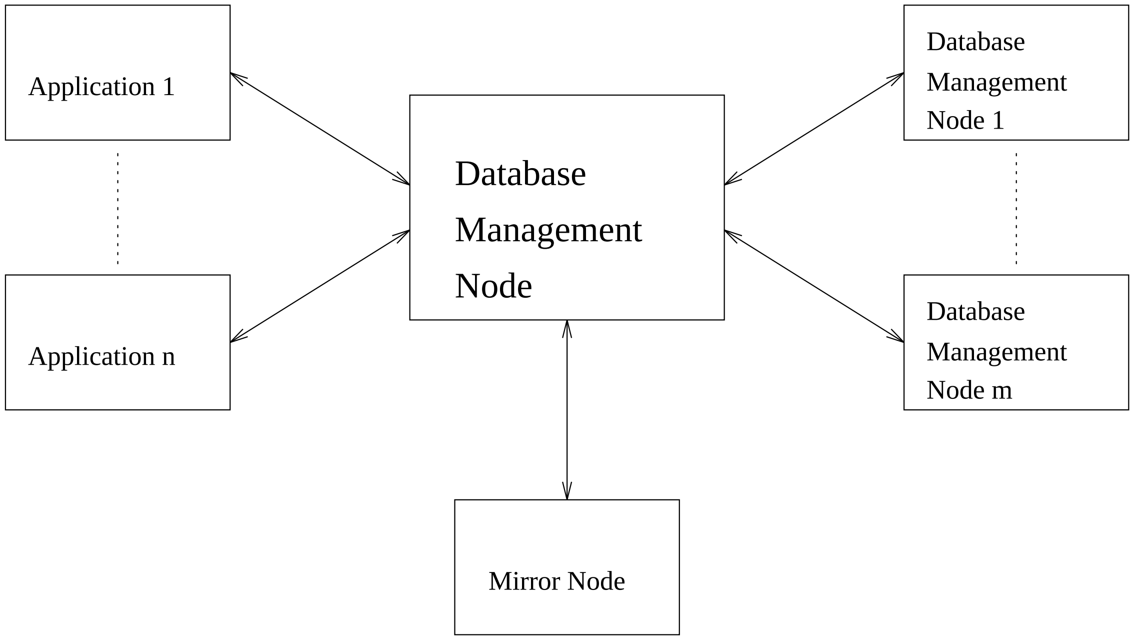

Figure 2: Overview of Darfin Distributed Database Architecture

(DDS), and Fault-Tolerance and Recovery Subsystem (FTRS) (see Figure 3). The URIS communicates with applications, the DDS with other database nodes, and the FTRS with the mirror node. The subsystems free the core database management system to be invisible to the network. It is possible to use alternate network communication methods by changing the subsystems, or it is possible to use a different core database architecture by changing the OODBMS.

A database node system or subsystem may have its own processes. Thus the database node process architecture has two levels: at the operating system level the operating system scheduler schedules system and subsystem processes. At the database system level a system or subsystem schedules its private processes. Currently only the OODBMS has its private processes. If we used only the operating system level scheduler, we could not use different real time scheduling algorithms in the OODBMS unless we first implemented a suitable operating system.

Object-Oriented Database Management System (OODBMS). The OODBMS is the core element of a database node. It implements the actual database architecture and a logical object model.

From the application designer's point of view, the logical object model is the most important feature of the database architecture. It defines the entry points that the application can use to access data in the database and the appearance of objects in the database. The OODBMS implements the object model services and maintains persistent objects and object information.

We have chosen the ODMG-93 object database standard (Cattell [1994]) as the logical object model for our database architecture. The ODMG-93 standard consists of the object model, object definition language, object query language, C++ binding, and Smalltalk binding. Together they form a complete definition for object oriented database design, implementation, and use. We hope to benefit from future commercial database management systems that follow the same ODMG-93 standard.

The OODBMS architecture consists of a set of database processes that communicate with each other. The actual OODBMS implementation is a service for the database processes. The processes offer services to the subsystems of the database node, and the subsystems themselves offer services to

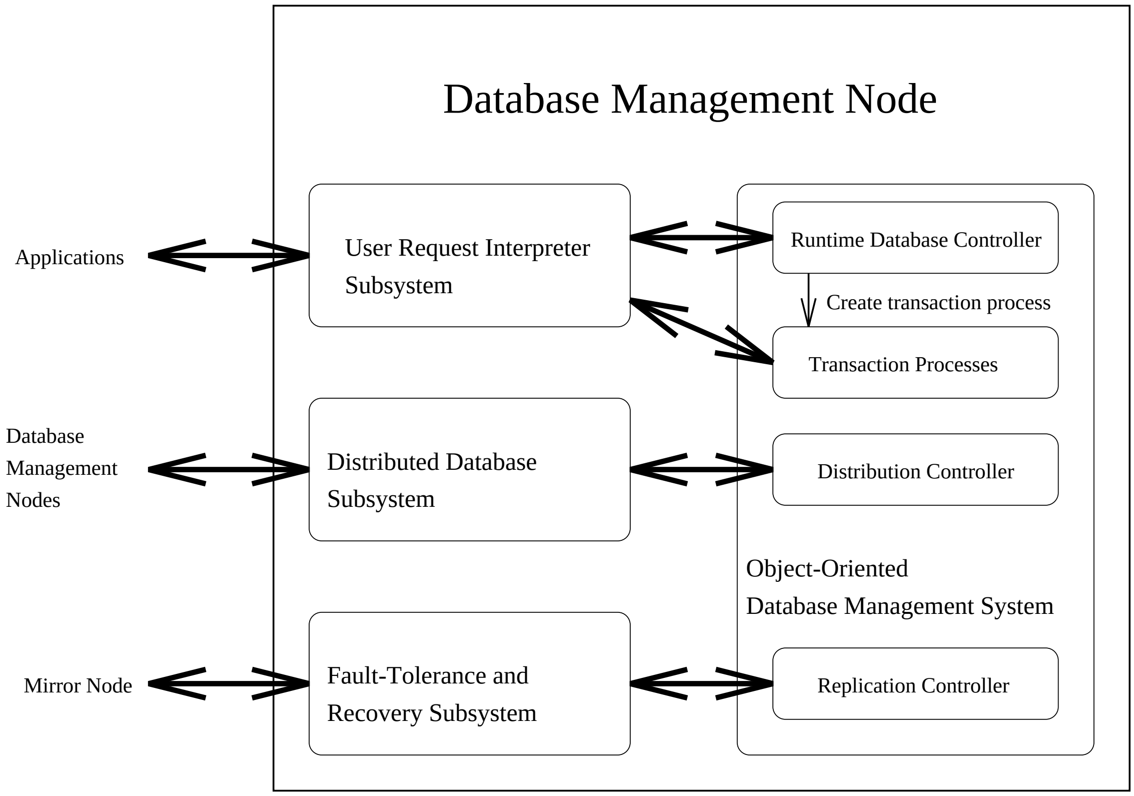

Figure 3: Database Node Architecture in Darfin

applications, other database nodes (where again a subsystem takes care of processing the services), and the mirror node. The database processes are Runtime Database Controller (RDC), Distribution Controller (DC), Replication Controller (RC), and Transaction processes.

The RDC creates a new transaction process when the URIS informs it that a new transaction request has arrived. It is the main entry point for the URIS. After creation the new transaction process communicates with the URIS to send and receive data. The RDC continues to listen to the URIS for new transaction requests.

The DC is a transaction process that sends and receives distributed operations. It is responsible for hiding data distribution in the OODBMS. When a distributed operation arrives from an application, the DC forwards the operation to the DDS that is responsible for the actual network communication to other database nodes. The DC also serves DDS requests that are sent from other database nodes.

The RC is a transaction process that maintains data replication. It hides the replication method from the OODBMS. Every update operation to the database triggers the RC. It forwards the update to the FTRS that is responsible for mirror node maintenance.

A Transaction process receives an array of commands from the URIS and interprets the command line accordingly. It processes the commands and then uses the database services to gain results. The results are then forwarded to the URIS that forwards them to the requesting application.

User Request Interpreter Subsystem (URIS). All applications that use the services of the database node communicate with the URIS. It is responsible for maintaining network connections with the applications, translating their requests to the OODBMS, and forwarding the results from the OODBMS back to the applications. The URIS implements different language bindings to the

OODBMS. When a new language binding is added, only the URIS need to be updated.

Distributed Database Subsystem (DDS). The DDS is responsible for processing and forwarding distributed operations between other DDS subsystems in other database nodes. The use of the DDS frees the OODBMS from low level distribution. In the OODBMS the DC is responsible for the distributed database operations, either operations from other nodes, or operations to the other nodes. The DDS is only a low level communication process.

Fault-Tolerance and Recovery Subsystem (FTRS). The FTRS is responsible for fault tolerance and recovery control. It communicates with the mirror node and with the RC in the OODBMS. When the OODBMS fails, the FTRS informs the mirror node to take control of the database node responsibilities. Then it informs the URIS to forward requests to the new node, and begins the recovery operation for the failed OODBMS. When the OODBMS is up and running again, the FTRS informs the mirror node that the actual database node is available. When the mirror node and database node are in the same state, that is when the mirror node has forwarded all updates that arrived while the database node OODBMS was down, the mirror node informs the FTRS that the system is stable again, and puts itself back to read-only state. The FTRS then informs the URIS that the requests should no longer be forwarded to the mirror node.

4.2 OODBMS Process Services

The most important element of the Darfin DBMS and Darfin Proto is the OODBMS and the services that it offers to the local database processes. The services that the processes may access are called the ODMG Interface Layer. The other deeper layers that implement the interface layer services are Physical Object Layer, Global Entity Layer, and Real-Time Core Layer (see Figure 4). A layer consists of a set of Manager Services that implement layer functionality.

ODMG Interface Layer. The ODMG Interface Layer is the service interface layer to the OODBMS services. It offers all services that the processes need. Currently the ODMG Interface Layer consists of three manager services: Schema Manager Service, Object Manager Service, and Directory Manager Service. Together the manager services offer a full ODMG compliant interface to the database.

The Schema Manager Service manages metainformation of objects and object classes. Every object belongs to a specific object class that defines the object structure.

The Object Manager Service manages objects. It offers the services that are defined to objects and types, such as accessing attributes or using operations.

The Directory Manager Service manages object directories. The principal object directory is based on object identity. There may be other directories based on names, attribute values, or conditions.

Physical Object Layer. The Physical Object Layer implements the services of the ODMG Interface Layer. On the Physical Object Layer level all objects are alike. The ODMG object model is no longer visible. In principle it is possible to use the layer and the lower layers in an object oriented database architecture that is not based on ODMG.

The Physical Object Layer has two manager services: Physical Object Manager Service and Dynamic Linking Manager Service.

The Physical Object Manager Service manages the physical objects. It offers services for object creation, deletion, access, object attribute access, and operation access. In the future the ODMG Interface Layer may need additional services. The Physical Object Manager Service is the core

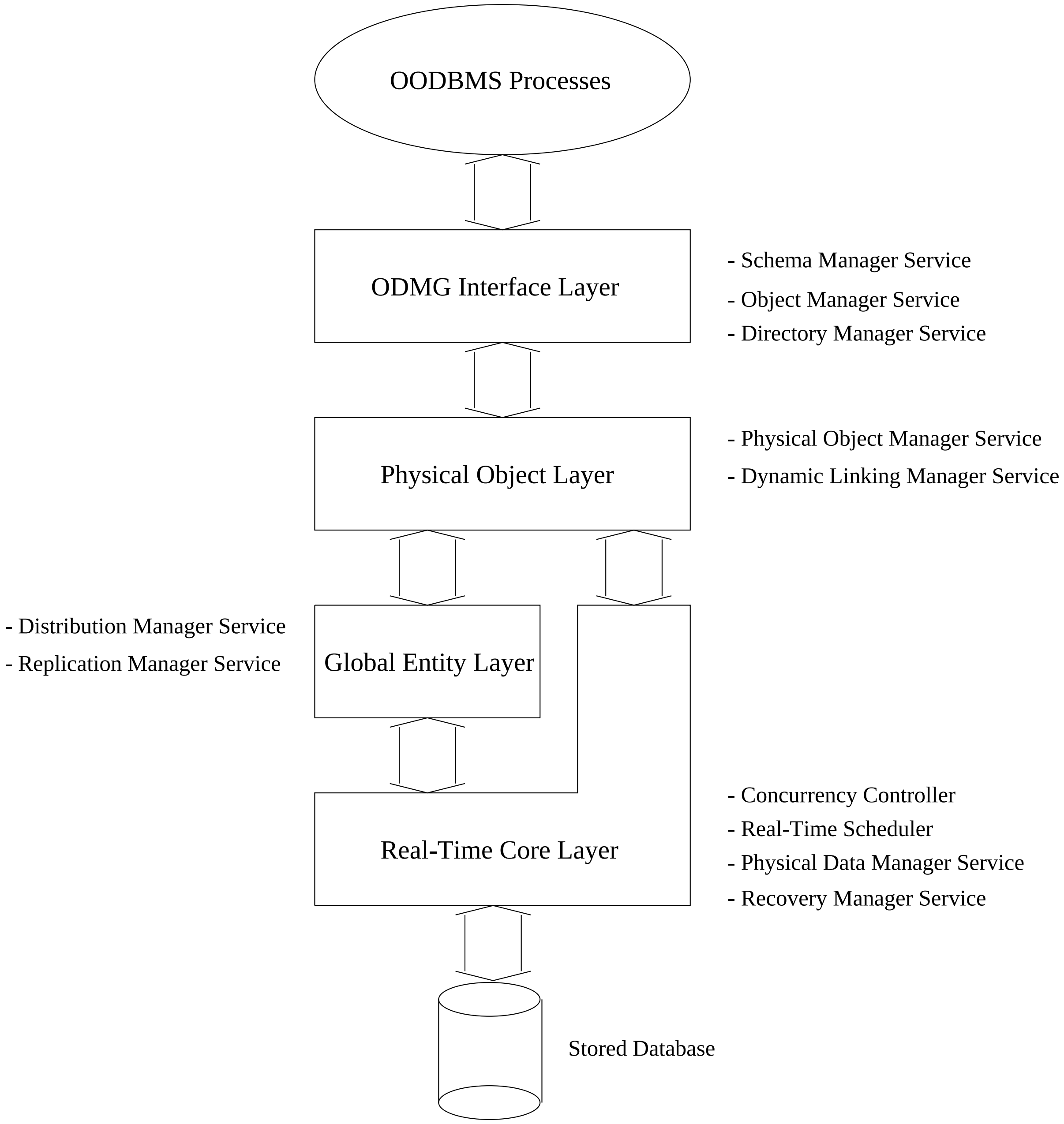

Figure 4: Layers of OODBMS Process Service in Darfin

manager service for a logical object model. It uses services of the lower layers to implement the physical object managing.

The Dynamic Linking Manager Service is responsible for linking new executable code to the database when a new object operation is created.

Global Entity Layer. The Global Entity Layer offers services for distributed and replication operations. It is mostly used by the Distribution Controller and the Replication Controller. The layer has two manager services: Distribution Manager Service and Replication Manager Service.

The Distribution Manager Service manages distributed operations on objects. It is mostly used by the Distribution Controller. In addition, the Physical Object Manager Service uses the Distribution

Manager Service when an access to an object in a different database node is needed.

The Replication Manager Service manages replication. It is mostly used by the Replication Controller but also all operations that change object's status are forwarded to the Replication Manager Service.

In the Darfin Proto both the Distribution Manager Service and Replication Manager Service only gather statistics of the service usages. The full Distribution and Replication Manager Services will be implemented in the Darfin TDMS.

Real-Time Core Layer. The Real-Time Core Layer implements the core operations of a real-time database management system. In the Real-Time Core Layer the processing model no longer deals with object orientation. This is dealt with in the Physical Object Layer. Instead, the Real-Time Core Layer maintains concurrency control, scheduling, physical data store maintenance, and recovery processing. The corresponding manager services are Concurrency Controller, Real Time Scheduler, Physical Data Manager Service, and Recovery Manager Service, respectively.

The Concurrency Controller is responsible for letting transactions to execute without breaking the isolation levels of transactions and objects. When the isolation levels are absolute, the transactions are serializable. The other extreme is to let transactions interfere with each other. Darfin DBMS supports various isolation levels depending on running transactions and referenced objects. We will use locking and the 2PL-HP algorithm, in which a higher priority process that claims a lock kills a lower priority process that holds the lock. The lower priority process is restarted if the deadline is not yet expired.

The Real-Time Scheduler schedules OODBMS processes. We will use a two level prioritized scheduling method. In the higher level, Runtime Database Controller, Distribution Controller, and Replication Controller have a higher priority than any of the transaction processes. In the lower level, the transaction processes have priorities according to their deadlines. We use the *Earliest Deadline First* (EDF) scheduling.

The Physical Data Manager Service manages the physical data storages: object data storage and metadata store. It is the only Manager Service that has access to the data stores. In Darfin Proto we use a pure main memory database architecture. All data resides in main memory, disks are used only for backup purposes. However, the physical data store architecture is not visible outside the Physical Data Manager Service. It is possible to use other data store architectures by changing the Physical Data Manager Service. The rest of the OODBMS architecture need not to be altered.

The Recovery Manager Service is needed when the OODBMS is recovering from a failure. It reloads the database to main memory and starts the necessary processes.

5. Summary

When a database systems for telecommunication services is to be designed the most important issues include *real-time*. We believe that a Real-Time Database System will in the future have an important role in telecommunication. We anticipate that the both *firm* and *soft* deadlines will be needed.

Since the needs for database services have quite different characteristics, the database system should be quite flexible. We believe that the required functionality can most easily and economically be achieved by an *object-oriented* approach. The functional needs are not easy to fulfil but we believe that a distributed real-time object-oriented database architecture is able to do that.

The Darfin DBMS architecture consists of a set of database nodes that communicate with each other. Together the nodes implement a distributed database architecture. Each database node is a full real-time object-oriented database that follows the ODMG-93 standard. We believe that the architecture is flexible enough to answer the current and future IN requirements, as well as a lot of other telecommunications network requirements.

Acknowledgements

The authors are thankful to the rest of the Darfin-team, particularly to Mika Rautila, Tapani Karttunen, and Harri Töhönen from Telecom Finland.

References

ABBOTT, R. AND GARCIA-MOLINA, H. 1990. Scheduling I/O Requests with Deadlines: A Performance Evaluation. In *Proceedings of the 11th Real-Time Systems Symposium*. IEEE, Los Alamitos, Calif., 113–124.

ABBOTT, R. AND GARCIA-MOLINA, H. 1992. Scheduling Real-Time Transactions: A Performance Evaluation. *ACM Transaction on Database Systems 17*, 3 (Sept.), 513–560.

AGRAWAL, R., CAREY, M. J., AND LIVNY, M. 1987. Concurrency Control Performance Modeling: Alternatives and Implications. *ACM Transaction on Database Systems 12*, 4 (Dec.), 609–654.

BUCHMANN, A. P., MCCARTHY, D. R., HSU, M., AND DAYAL, U. 1989. Time-Critical Database Scheduling: A Framework For Integrating Real-Time Scheduling and Concurrency Control. In *Proceedings of the 5th International Conference on Data Engineering*. IEEE, Los Alamitos, Calif., 470–480.

CATTELL, R. G. G., EDITOR. 1994. *The Object Database Standard: ODMG-93*, Release 1.1. Morgan Kaufmann, San Francisco, Calif.

CHATRAS, B. AND GALLANT, F. Protocols for Remote Data Management in Intelligent Networks CS1. In *Workshop Record of IEEE Intelligent Network '94 Workshop*, Volume 1. IEEE Communication Society, May 1994.

CHEN, M.-I. AND LIN, K.-J. 1990. Dynamic Priority Ceilings: A Concurrency Control Protocol for Real-Time Systems. *The Journal of Real-Time Systems 2*, 4 (Dec.), 325–346.

ESWARAN, K. P., GRAY, J. N., LORIE, R. A., AND TRAIGER, I. L. 1976. The Notions of Consistency and Predicate Locks in a Database System. *Commun. ACM 19*, 11 (Nov.), 624–633.

FRANASZEK, P. A. AND ROBINSON, J. T. 1985. Limitations of Concurrency in Transaction Processing. *ACM Transactions on Database Systems 10*, 1 (Mar.), 1–28.

FRANASZEK, P. A., ROBINSON, J. T., AND THOMASIAN, A. 1992. Concurrency Control for High Contention Environments. *ACM Transactions on Database Systems 17*, 2 (June), 304–345.

GARCIA-MOLINA, H. AND SALEM, K. 1992. Main Memory Database Systems: An Overview. *IEEE Transactions on Knowledge and Data Engineering 4*, 6 (Dec.), 509–516.

HARITSA, J. R., CAREY, M. J., AND LIVNY, M. 1990. On Being Optimistic about Real-Time Constraints. In *Proceedings of the 9th ACM Symposium on Principles of Database Systems*. ACM, 331–343.

HARITSA, J. R., LIVNY, M., AND CAREY, M. J. 1991. Earliest Deadline Scheduling for Real-Time Database Systems. In *Proceedings of the 12th Real-Time Symposium*. IEEE, Los Alamitos, Calif., 232–242.

HUANG, J., STANKOVIC, J. A., TOWSLEY, D., AND RAMAMRITHAM, K. 1989. Experimental Evaluation of Real-Time Transaction Processing. In *Proceedings of the 10th Real-Time Systems Symposium*. IEEE, Los Alamitos, Calif., 144–153.

HUANG, J., STANKOVIC, J. A., RAMAMRITHAM, K., AND TOWSLEY, D. 1991. Experimental Evaluation of Real-Time Optimistic Concurrency Control Schemes. In *Proceedings of the 17th VLDB Conference*. Morgan Kaufmann, San Mateo, Calif., 35–46.

HUANG, J., STANKOVIC, J. A., RAMAMRITHAM, K., TOWSLEY, D., AND PURIMETLA, B. 1992. Priority Inheritance in Soft Real-Time Databases. *The Journal of Real-Time Systems 4*, 2 (June), 243–268.

ITU. 1991–4. *Recommendations in X.700-Series: OSI Management*. International Telecommunications Union, Geneva, Switzerland.

ITU. 1993a. *Recommendations in M.3000-Series: Telecommunication Network Management*. International Telecommunications Union, Geneva, Switzerland.

ITU. 1993b. *Principles of Intelligent Network. Recommendation Q.1201*. International Telecommunications Union, Geneva, Switzerland.

ITU. 1993c. *Introduction to Intelligent Network Capability Set 1. Recommendation Q.1211*. International Telecommunications Union, Geneva, Switzerland.

ITU. 1994a. *Intelligent Network Distributed Functional Plan Architecture. Recommendation Q.1204*. International Telecommunications Union, Geneva, Switzerland.

ITU. 1994b. *Distributed Functional Plane for Intelligent Network CS-1. Recommendation Q.1214*. International Telecommunications Union, Geneva, Switzerland.

ITU. 1994c. *Interface Recommendation for Intelligent Network CS-1. Recommendation Q.1218*. International Telecommunications Union, Geneva, Switzerland.

ITU. 1995. *Basic Reference Model of Open Distributed Processing — Parts 1–3: Introduction, Foundations, and Architecture. Draft Recommendations X.901, X.902, and X.903*. International Telecommunications Union, Geneva, Switzerland.

Kao, B. and Garcia-Molina, H. 1993. Deadline Assignment in a Distributed Soft Real-Time System. In *Proceedings of the 13th Int. Conf. on Distributed Computing Systems*. IEEE Computer Society Technical Committee on Distributed Processing, Los Alamitos, Calif., 428–437.

Kung, H. T. and Robinson, J. T. 1981. On Optimistic Methods for Concurrency Control. *ACM Transaction on Database Systems 6*, 2 (June), 213–226.

Lin, K.-J. and Son, S. H. 1990. Concurrency Control in Real-Time Databases by Dynamic Adjustment of Serialization Order. In *Proceedings of the 11th Real-Time Systems Symposium*. IEEE, Los Alamitos, Calif., 104–112.

Liu, C. L. and Layland, J. W. 1973. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the ACM 20*, 1 (Jan.), 46–61.

Pang, H., Livny, M., and Carey, M. J. 1992. Transaction Scheduling in Multiclass Real-Time Database Systems. In *Proceedings of the 13th Real-Time Systems Symposium*. IEEE, Los Alamitos, Calif., 23–34.

Raatikainen, K. E. E. 1994. Information Aspects of Services and Service Features in Intelligent Network Capability Set 1. Report C-1994-45, University of Helsinki, Dept. of Computer Science, Helsinki, Finland.

Raatikainen, K. E. E. 1995. Database Access in Intelligent Networks. In *Intelligent Networks*, J. Harju, T. Karttunen, and O. Martikainen (Eds). Chapman & Hall, London, UK, 173–193.

Rosenkrantz, D. J., Stearns, R. E., and Lewis II, P. M. 1978. System Level Concurrency Control for Distributed Database Systems. *ACM Transactions on Database Systems 3*, 2 (June), 178–198.

Sha, L., Rajkumar, R., and Lehoczky, J. P. 1990. Priority Inheritance Protocols: An Approach to Real-Time Synchronization. *IEEE Transactions on Computers C-39*, 9 (Sept.), 1175–1185.

Sha, L., Rajkumar, R., Son, S. H., and Chang, C.-H. 1991. A Real-Time Locking Protocol. *IEEE Transactions on Computers 40*, 7 (Jan.), 793–800.

Sprunt, B., Kirj, D., and Sha, L. 1988. Priority-Driven, Preemptive I/O Controllers for Real-Time Systems. In *Proceedings of Int. Symp. on Computer Architecture*. ACM, 152–159.

Taina, J. 1995. Requirements Analysis for Database Services in Telecommunications. Report C-1995-17, University of Helsinki, Dept. of Computer Science, Helsinki, Finland.

Tay, Y. C., Suri, R., and Goodman, N. 1985. A Mean Value Performance Model for Locking in Databases: The No-Waiting Case. *Journal of the ACM 32*, 3 (July), 618–651.

Wu, K.-L., Yu, P. S., and Chen, M.-S. 1993. Dynamic Finite Versioning: An Effective Versioning Approach to Concurrent Transaction and Query Processing. In *Proceedings of the nth Int. Conf. on Data Engineering*. IEEE, Los Alamitos, Calif., 577–586.

Yu, P. S. 1990. Effective Concurrency Control With No False Killing. *IBM Tech. Disclosure Bull. 33*, 2 (July), 193–194.

Yu, P. S. and Dias, D. M. 1992. Analysis of Hybrid Concurrency Control for a High Data Contention Environment. *IEEE Transactions on Software Engineering SE-18*, 2 (Feb.), 118–129.

Yu, P. S. and Dias, D. M. 1993. Performance Analysis of Concurrency Control Using Locking With Deferred Blocking. *IEEE Transactions on Software Engineering SE-19*, 10 (Oct.), 982–996.

Yu, P. S., Dias, D. M., and Lavenberg, S. S. 1993. On the Analytical Modeling of Database Concurrency Control. *Journal of the ACM 40*, 4 (Sept.), 831–872.