

# A Distributed Real-Time Main-Memory Database for Telecommunication

Jan Lindström, Tiina Niklander, Pasi Porkka, Kimmo Raatikainen  
University of Helsinki, Department of Computer Science

P.O. Box 26 (Teollisuuskatu 23), FIN-00014 University of Helsinki, Finland

Email: {jan.lindstrom,tiina.niklander,pasi.porkka,kimmo.raatikainen}@cs.Helsinki.FI

## 1 Introduction

Recent developments in network and switching technologies have increased data intensity of telecommunications systems and services far more data intensive. This is clearly seen in many areas of telecommunications including network management, service management, and service provisioning. For example, in the area of network management the complexity of modern networks leads to a large amount of data on network topology, configuration, equipment settings, and so on. In the area of service management there are customer subscriptions, the registration of customers, and service usage (e.g. call detail records) that lead to large databases.

The integration of network control, management, and administration also leads to a situation where database technology becomes an integral part of the core network; For example in architectures like TMN [16], IN [9, 17], and TINA [7]. The combination of vast amounts of data, real-time constraints, and necessity of high availability creates challenges for many aspects of database technology including distributed databases, database transaction processing, storage and query optimization.

Until now, the database research community has only paid a little attention to data management in telecommunications and has contributed little beyond core database technology. The challenges facing telecom data management are now at a point that the database research community can and should become deeply involved.

The performance, reliability, and availability requirements of data access operations are demand-

ing. Thousands of retrievals must be executed in a second and the allowed down time is only a few seconds in a year.

Telecommunication requirements and real-time database concepts are somewhat studied in the literature [4, 2, 24, 3].

Kim and Son [27, 23] have presented a *StarBase* real-time database architecture. This architecture has been developed over a real-time micro kernel operating system and it is based on relational model. Wolfe & al. [46] have implemented a prototype of object-oriented real-time database architecture *RTSORAC*. Their architecture is based on open OODB architecture with real-time extensions. Database is implemented over a thread-based POSIX-compliant operating system. Another object-oriented architecture is presented by Cha & al. [6]. Their *M<sup>2</sup>RTSS*-architecture is a main-memory database system. It provides classes, that implement the core functionality of storage manager, real-time transaction scheduling, and recovery.

*Dali* [22] is a main memory storage manager, that can be used as a memory manager in any single main memory database system. *BeeHive* [41] is a real-time database supporting transaction and data deadlines, parallel and real-time recovery, fault tolerance, real-time performance and security. *Clus-tRa* [14, 45] is a database engine developed for telephony applications like mobility management and intelligent networks.

The RODAIN<sup>1</sup> database architecture is *real-time*, *object-oriented*, *fault-tolerant*, and *distributed* data-

---

<sup>1</sup>RODAIN is the acronym of the project called *Real-Time*

base management system. The RODAIN architecture is designed to fulfill the requirements of a modern database system for telecommunications. The requirements are derived from the most important standards in telecommunications including *Intelligent Network* (IN), *Telecommunications Management Network* (TMN), and *Telecommunication Information Networking Architecture* (TINA). The requirements originate in the following areas: real-time access to data, fault tolerance, distribution, object orientation, efficiency, flexibility, multiple interfaces, and compatibility [35, 43].

In RODAIN we have developed a real-time main-memory database architecture that is an object-oriented database that uses object model of ODMG [5] with real-time extensions of our own [25]. The additional characteristics are designed so that the original ODMG model is a true subset of our extended model. The essence of the extensions is to guarantee that the real-time scheduler and the concurrency controller have enough knowledge to overcome the problems due to heterogeneous lengths of transactions. The prototype implementation of RODAIN database architecture supports real-time scheduling of real-time and non-real-time transactions.

In telecommunication applications, database transactions have several different characteristics. In RODAIN we have studied two basic types of transactions: service provision transactions (IN) and service management transactions (TMN). The IN transactions are expressed as firm real-time transactions<sup>2</sup> whereas the TMN transactions are expressed as non-real-time transactions. Traditional concurrency control methods use serializability as the correctness criterion when transactions are concurrently executed. However, strict serializability as the correctness criterion is not always suitable in real-time databases, in which correctness requirements may vary from one type of transactions to another and from one type of data to

---

*Object-Oriented Database Architecture for Intelligent Networks* funded by Nokia Telecommunications, Solid Information Technology Ltd., and the Finnish Technology Development Center (TEKES).

<sup>2</sup>Firm real-time transactions have a deadline that should be met. If the deadline expires, then the transaction is aborted since its results do not have any value after the deadline.

another. Some data may have temporal behavior, some data can be read although it is already updated but not yet committed, and some data must be guarded by strict serializability. These conflicting requirements must be solved through using a special purpose concurrency control scheme. Therefore, the RODAIN concurrency control is based on an optimistic method which is extended with relaxed serializability and semantic conflict resolution.

Any database used in many telecommunication services must also be continuously available. The “official” ITU requirements allow down time to be only few seconds per failure. The high availability of the RODAIN database system is achieved by using two separate nodes with their own copies of the full database. One node, called the primary node, executes the database updates. The other node, called the mirror node, monitors the changes in the database content and is ready to take the update responsibilities if the primary node fails.

The rest of the paper is organized as follows. In Section 2 we briefly summarize the key requirements for a database that is suitable for telecommunication applications. The architecture of the RODAIN Database Management System is presented in Section 3. In Section 4 we introduce the application interfaces and in Section 5 we discuss transaction processing. A detailed description of the concurrency control is given in Section 6.

## 2 Database Requirements in Telecommunications

Below we summarize the key requirements including *data distribution*, *data replication*, *object orientation*, *object directories*, *multiple application interfaces*, *fault tolerance*, and *real-time transactions*. A more detailed description can be found in [44].

**Data distribution.** A set of database nodes may cooperate to accomplish database tasks. A node may request information from other nodes in case information needed is not locally available. It is possible to implement the underlying database system without data distribution. In such an implementation all applications use a single database

server. This differentiates logical data distribution from physical data distribution among database nodes. The former is necessary but the latter is a decision internal to the design of the database architecture. Our current belief is that only a few requests need to access more than one database node.

**Data replication.** It is stated in ITU-T Recommendation Q.1204 [20] that a database node contains customer and network data for real-time access. We believe that currently most distributed operations are reads. Therefore, replication is an effective way to speed up distributed operations.

**Object orientation.** It is a common belief that the best long term telecommunications architectures are object oriented. The implication is that a database system must support object access. In other words, the database system must either be object oriented or have an object interface.

**Object directories.** The conceptual model of IN Service Control Function (ITU-T Recommendation Q.1214 [18] ) defines a Service Data Object Directory that is used to access data stored in the Service Data Functions. This implies that object directories must be supported. The invocation mechanism of CORBA is a good alternative to implement the functionality.

**Multiple application interfaces.** In telecommunications different architectures define different access interfaces. The IN Capability Set 1 defines IN Application Protocol (INAP). TMN has its own access methods based on the OSI (X.700) management protocols. The current definition in TINA is based on TINA ODL which is quite similar to the OMG IDL interface. Therefore, OMG IDL interface is also needed. In addition, OQL interface—probably without response-time guarantees—would be convenient for ad hoc queries and database maintenance.

**Fault tolerance.** Real-time access implies that data must be continuously available. The result of the implications is that the database system must be fault tolerant. In the current definitions the maximum allowed down time is a few seconds a year.

**Real-time transactions.** Although the real-time data access as stated in ITU-T Recommendation Q.1204 does not directly imply that the underlying database system must support real-time transactions, we believe that the most convenient way to support real-time access to data is to use a real-time database system. In telecommunications we will need both soft and firm transactions. We do not believe that hard transactions will be used in near future because systems supporting hard transactions are too expensive for open telecommunication markets.

### 3 Overview of the RODAIN Database Architecture

The RODAIN Database Architecture is a hierarchical distributed database architecture. The RODAIN Database Nodes are linked together to form database clusters. Furthermore, the database clusters are then connected together to allow access between different, possibly heterogeneous, distributed databases (Figure 1).

RODAIN Database Nodes within one database cluster share common metadata. They also have access to a global dictionary in order to locate any data item. Different telecommunication applications as clients of RODAIN Database Cluster can access any one of the Database Nodes. Each node serving a request can fully hide data distribution by providing access to all data items within the cluster. For the most time-critical transactions the RODAIN Database offers the possibility to learn the fastest access point of each time-critical data item.

RODAIN Database Clusters do not necessarily share common metadata. Instead, schema translations may be needed when data items from remote cluster are accessed. It should also be noted that no assumption of real-time behavior of the remote cluster can be made since the remote cluster usually belongs to a different administration domain. Therefore, we assume that the communication between clusters will be based on standard communication protocols and object access models like the ones used in CORBA [32]. In this way

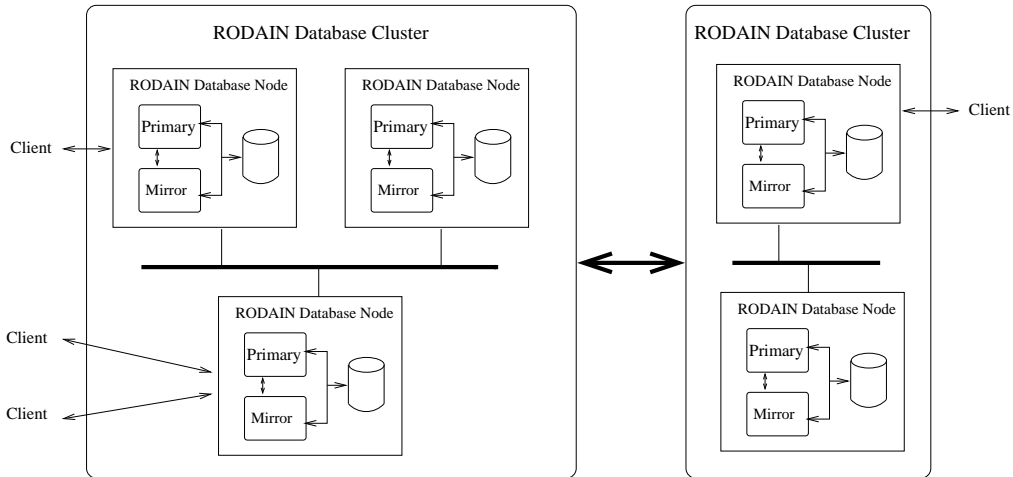


Figure 1: RODAIN Database Clusters.

a remote cluster does not have to be a RODAIN Database Cluster. In fact a remote cluster can be any object or relational database.

The data in each Rodain Database Node is divided into two parts; each data item belongs to hot data or to cold data [37]. The data items are stored in different databases in the Rodain Database Node: hot data is stored in a main-memory database and cold data is stored in a disk-based database. All updates in the hot data are done in main memory. A transaction log of hot data is maintained to keep the database in a consistent state. A secondary copy of hot data is located in a mirror node and only a backup copy is maintained on the disk. Since cold data is stored in a disk-based database we use a hybrid data management method that combines a main-memory database and a disk-based database. RODAIN Database Nodes that form a RODAIN Database Cluster are real-time, highly-available, main-memory database servers. They support concurrently running real-time transactions using optimistic concurrency control protocol with deferred write policy. They can also execute non-real-time transactions at the same time on the database. Real-time transactions are scheduled based on their type, priority, mission criticality, or time criticality. In order to increase the availability of the database each Rodain Database Node consists of two identical co-operative nodes. One of the nodes acts as the Database Primary Node and the other one, Database Mirror Node, is mirroring the Primary

Node. When ever necessary, that is when a failure occurs, the Primary and the Mirror Node can switch their roles. When there is only one node in function we call it as a Transient Node. A Transient Node behaves like Primary Node, but it is not accompanied by a Mirror Node and, therefore, it behaves like a stand-alone database system. The role of Transient Node is designed to be temporary and used only during the failure period of the other node.

Database Primary Node and Mirror Node use a reliable shared Secondary Storage Subsystem (SSS) for permanently storing the cold data database, copies of the hot data database, and log information. The nodes themselves are further divided into set of subsystems (Figure 2) that perform the needed functionality on both nodes. Below we will shortly summarize the functionality of each subsystem.

**User Request Interpreter Subsystem.** The Rodain Database Node can have multiple application interfaces. Each interface is handled by one specific User Request Interpreter Subsystem. It translates its own interface language into a common connection language that the database management subsystem understands. The URISs on the Primary Node are active and communicate with the clients. On the Mirror Node the URISs are not needed.

**Distributed Database Subsystem.** A Rodain Database Node may either be used as a stand-alone system or in co-operation with the other autonomous Rodain Database Nodes within one RODAIN

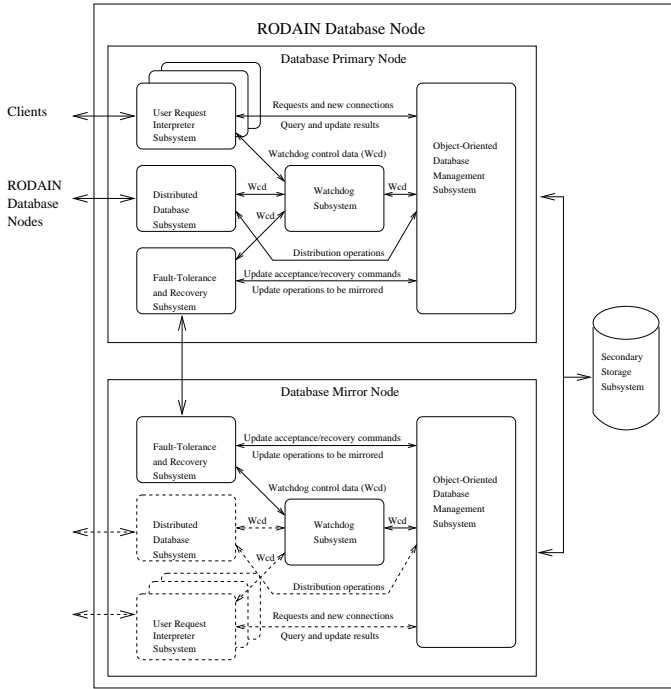


Figure 2: Rodain Database Node.

Database Cluster. The database co-operation management in the Database Primary Node is left to the Distributed Database Subsystem (DDS). The Distributed Database Subsystem on the Mirror Node is passive or non-existent. It is activated when the Mirror Node becomes a new Primary or Transient Node.

**Fault-Tolerance and Recovery Subsystem.** The FTRS on both nodes controls communication between the Database Primary Node and the Database Mirror Node. It also co-operates with the local Watchdog Subsystem to support fault tolerance.

The FTRS on the Primary Node handles transaction logs and failure information. It sends transaction logs to the Mirror Node. It also monitors the Mirror Node. When it notices a failure it reports that to the Watchdog Subsystem for changing the role of the node to the Transient Node. On the Transient Node FTRS stores the logs directly to the disk on the SSS.

The FTRS on the Mirror Node receives the logs sent by its counterpart on the Primary Node. It then saves the logs to disk on Secondary Storage Subsystem and gives needed update instructions to the local Database Management Subsystem. When

it notices that the Primary Node has failed, it informs the local Watchdog Subsystem to start role change.

**Watchdog Subsystem.** The Watchdog subsystem watches over the other local running subsystems both on the Primary and on the Mirror Node. It implements a subset of watchd [13] service. Upon a failure it recovers the node or the failed subsystem. Most subsystem failures need to be handled like the failure of the whole node. The failure of the whole node requires compensating operations on the other node. On Primary Node when the failure of the Mirror Node is noticed, the WS controls the node change to the Transient Node. This change affects mostly the FTRS, that must start storing the logs to the disk. On Mirror Node the failure of the Primary Node generates more work. In order to change the Mirror Node to the Transient Node the WS must activate passive subsystems such as URIS and DDS. The FTRS must change its functionality from receiving logs to saving them to the disk on SSS.

**Object-Oriented Database Management Subsystem.** The OO-DBMS is the main subsystem on both Primary Node and Mirror Node. It maintains both hot and cold databases. It maintains real-time constraints of transactions, database integrity, and concurrency control. It consists of a set of database processes, that use database services to resolve requests from other subsystems, and a set of manager services that implement database functionality. The Object-Oriented Database Management Subsystem needs the Distributed Database Subsystem, when it can not solve an object request on the local database.

## 4 Usage Interfaces

As described in Section 3 there are multiple URISs in the RODAIN architecture. For each protocol allowed to access RODAIN database system, there must exist an URIS. Each URIS receives queries and service requests presented in one specific protocol and transforms these queries and requests into a language that the DBMS understands.

An open database management system must support several interfaces to the database. We have studied requirements set by two Intelligent Network's protocols. *Intelligent Network Application Protocol (INAP)* is described in the capability set 1 (CS-1) [19]. It introduces a *Service Data Function (SDF)*, which provides the functionality needed for storing, managing and accessing information in the network. CS-2 introduces *Data Access Protocol (DAP)* [21]. IN CS-2 defines the rest of the IN services. The enhancements of the DAP include authentication, security, assignment of the access rights, control user's data access and block data access.

An interface for OSI's *Common Management Information Protocol (CMIP)* [15] is also studied. With CMI protocol user can insert an information tree into RODAIN and manage it. With CMIP the user can successfully make connections to the database and manage the information tree and manipulate the information in it.

CORBA [30, 32] offers means for distribution and built in connections to several different platforms. It has gained growing interest also in the telecommunication. Therefore, we have evaluated the integration of CORBA and DBMS. As a result, we have specified the requirements for an *Object Database Adapter (ODA)* [33]. The direct invocation of database objects from a CORBA application is an important performance issue. It would be impossible for the CORBA to work properly if all database objects were to be registered as CORBA objects. It would cause severe overhead for the CORBA system. An ODA should provide both static and dynamic invocation (DII) of database objects. This is achieved by introducing the interfaces of database objects to CORBA. Plenty of the functionality provided by CORBA is defined in different service specifications [32]. Some of the services include functionality that ODA should also provide. In Figure 3 the basic architecture of RODAIN Object-Oriented Database Adapter (ROODA) is shown. The names Persistent Object Service (*POS*), Object Transaction Service (*OTS*) and Object Query Service (*OQS*) refer to the corresponding services. *DII* and *DSI* refer to dynamic invocation and skeleton interfaces. It should be

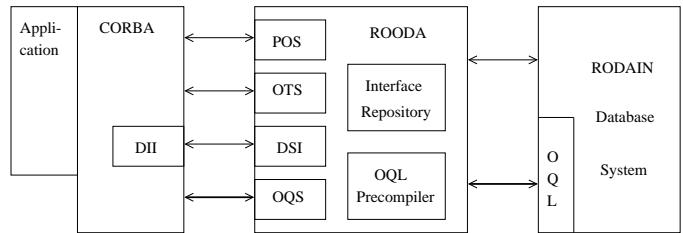


Figure 3: CORBA, OODA and RODAIN interaction.

noted that, in the context of ROODA only certain parts of these services are implemented.

**The Persistent Object Service (POS)** provides a common interface to the mechanism used for retaining and managing the persistent state of objects. The client, that has requested or created an object, may make it persistent by invoking objects introduced in the POS interface. For this service, ROODA acts as a *persistent data service (PDS)* and the RODAIN DBMS as a *datastore*. However, in the version 2.2 of CORBA, the POS is abandoned and a *Persistent Object Adapter (POA)* is taking care of persistence [31].

**The Object Transaction Service (OTS)** provides transaction synchronization across the elements of a distributed client/server application. A transaction can involve multiple objects performing multiple requests. OTS places no constraints on the number of objects involved, the topology of the application or the way in which the application is distributed across the network. However, there is one restriction for the participating objects. The objects called within OTS must be either transactional or recoverable for that operation to display transaction semantics [40].

**Dynamic Invocation and Skeleton Interfaces (DII & DSI)** provide means for dynamic object invocation. DII allows clients to access objects, which were not available in the system when the client was compiled. DSI is the server's part of the dynamic invocation procedure. They are required in the ROODA, since the database evolves constantly.

**Object Query Service (OQS)** [32] provides predicate-based queries of selection, insertion, updating, and deletion on collections of objects. Operations are executed to *source collections* and they

may return *result collections* of objects. The result collections may be either selected from the source collections or produced by query evaluators. Queries may be further set also on the result collections.

The OQS consists of two different interfaces; the query interface and the interface for iterable collections. CORBA objects may participate in the OQS in two different ways. All CORBA *objects as themselves* are queryable. A query may be set on object's attributes, relationships, or methods. CORBA objects can also be *members of collections*. With the iterable collection interface queries may be set efficiently on these collections.

OQS may set queries also to objects that are not managed by CORBA. Usually these external objects reside in a database. The CORBA can access database objects basically only one at a time, since CORBA handles single objects. If the database management provides class collection, objects of that class may be requested by CORBA. Efficient querying of objects from an external database requires the use of database's native query language. Hence, we must provide an efficient OQL interface in the RODAIN DBMS.

## 5 Transaction Processing

There are three problems that any real-time database system must deal with: 1) resolving resource contention, 2) resolving data contention, and 3) enforcing timing constraints. In the RODAIN database transactions to be executed are classified according to their *deadline* and to their *importance*. Real-time transactions differ from traditional transactions in many ways. In traditional database systems the main goal of transaction scheduling is to maximize the throughput of transactions. In real-time database systems the goal is to maximize the number of transactions that complete before their deadlines. Predictability of transaction lengths is one of the most important prerequisites for meeting the deadlines in real-time systems.

In real-time database systems transactions must be controlled by a real-time scheduler. A scheduling priority for each transaction is automatically evaluated and modified during the run time. Evaluation

process is based on characteristics of the transaction, validity of the accessed data object, and the dynamic behavior of the system.

The goal of real-time scheduling is to maximize the number of transactions that complete before their deadlines. However, this is not always possible. In these situations transactions having low values of the `transaction_importance` property are sacrificed in favor for those having high values. In RODAIN database transactions arrive asynchronously. Due to the nature of telecommunication systems the arriving rate of transactions may vary within a very large bounds. Furthermore, a length of each transaction and its data accessing patterns may vary in unpredictable way. Therefore, there are sometimes situations when DBMS load temporarily exceeds its processing capacity. In literature this is called an *overload* situation.

In RODAIN database we use an *adaptive overload prevention policy*. A principle of this policy is to limit the number of active Transaction Processes. The present limit is based on system load, which is observed in periodic basis. The metric of system load is how many transaction misses its deadline within an observation period. Another factor for limit calculation is how many aborts are made due to concurrency control.

Transaction processing in the RODAIN system is a task which requires co-operation of numerous processes. These processes are located 1) in the Object-Oriented Database Management Subsystem (OO-DBMS), 2) outside the OO-DBMS but inside the RODAIN node, and 3) outside the RODAIN node. Transactions can directly modify local and remote data, call (invoke) local and remote methods, and commit or abort modifications done in local node or remote nodes.

During the last 10–15 years several algorithms have been developed to schedule real-time transactions (e.g.,[47]). The main goal of the real-time scheduling algorithms is to guarantee that as many transactions as possible will meet their deadline. The scheduling algorithms have different behavior depending on both data and processing resource contention [1]. For the RODAIN database system the challenge in real-time scheduling is the fact that existing real-time scheduling algorithms do not guar-

antee that non-real-time transactions will receive enough resources to complete.

In telecommunication applications database transactions have several different characteristics [36]. In RODAIN database system two transaction types have been studied: service provision (IN) and service management (TMN) transactions. IN transactions are used to access data of one customer. TMN transactions are used to update database contents widely, for example, when adding new users or configuring user profiles.

IN type transactions are short reads or updates, which usually affect a few database objects. Typically the object is fetched based on the value of its key attribute value. Transaction atomicity and consistency requirements can sometimes be relaxed [43]. However, IN transactions have quite strict deadlines and their arrival rate can be high, but most IN transactions have read-only semantics. In existing IN applications ratio of write transactions is approximately 1 to 10 percent [43]. In RODAIN transaction scheduling IN transactions are expressed as firm real-time transactions.

TMN type transactions have opposite characteristics than IN transactions. They are long updates which write many objects. Strict serializability, consistency and atomicity are required for TMN transactions. However, they do not have explicit deadline requirements. Thus, TMN transactions are expressed as non-real-time transactions.

Most database systems offer either real-time or fair scheduling for concurrent transactions. These aspects often conflict with each other. To fulfill telecommunication database requirements conflicting transaction types must be scheduled simultaneously in the same database. Short IN transactions must be completed due to their deadlines as well as long TMN transactions must get enough resources to complete.

The RODAIN scheduling algorithm, called *FN-EDF*, is designed to support simultaneous execution of both firm real-time and non-real-time transactions. In RODAIN firm deadline transactions are scheduled according to the EDF scheduling policy [29, 12]. The FN-EDF algorithm guarantees that non-real-time transactions receive a respecified amount of execution time.

The FN-EDF algorithm periodically samples the execution times of all transactions. The operating system scheduling priority of a non-real-time transaction is adjusted if its fraction of execution time is either above or below the respecified target value. For each class of non-real-time transactions ( $c = 1, \dots, C$ ) the target value is given as a system parameter  $\gamma_c$ , which can be changed while the system is running.

The scheduling of transaction processes is based on the FIFO scheduling policy provided by the Chorus operating system [34]. A continuous range of priorities is reserved for transactions. For firm deadline transactions the priority is assigned once and subsequent transactions receive priorities based on previous assignments. When a non-real-time transaction is started, it receives the lowest priority of priority range. During adjustment phases the priority is raised until the transaction receives the deferred fraction of execution time. Transaction processing of the RODAIN DBMS is discussed more detailed in [25].

## 6 Concurrency Control

Traditional concurrency control methods use serializability [8] as the correctness criterion when transactions are executed concurrently. However, strict serializability as the correctness criterion is not always the most suitable one in real-time databases, in which correctness requirements may vary from one type of transactions to another. Some data may have temporal behavior, some data can be read although it is already written but not yet committed, and some data must be guarded by strict serializability. These conflicting requirements may be solved through using a special purpose concurrency control scheme.

A heterogeneous transaction behavior introduces problems also to concurrency control. In traditional databases, database correctness is well defined and homogeneous between transactions. However, a demand for strict database correctness is not applicable for real-time databases, where correctness requirements can be heterogeneous. Real-time data is often temporal and neither serializing concurrency

control nor full support for failure recovery is not required because of the overhead [42]. This has led to ideas such as *atomic set-wise serializability* [39], *external versus internal consistency* [28], *epsilon serializability* [38], and *similarity* [26]. Graham [10, 11] has argued that none are as obviously correct, nor as obviously implementable, as serializability.

Due to the semantic properties of telecommunications applications, the correctness criterion can be relaxed to so called *semantic based serializability*. We decompose the semantic based serializability into two parts. Firstly, we define a temporal serializability criterion called  $\tau$ -serializability [36], which allows old data to be read unless the data is too old. Secondly, we use a semantic conflict resolution model that introduces explicit rules, which are then used to relax serializability of transactions. The first method reduces the number of read-write conflicts whereas the second one reduces the number of write-write conflicts.

We use the  $\tau$ -serializability as a correctness criterion to reduce read-write conflicts. Suppose that transaction  $T_A$  updates the data item  $x$  and gets a write lock on  $x$  at time  $t_a$ . Later transaction  $T_B$  wants to read the data item  $x$ . Let  $t_b$  be the time when  $T_B$  requests the read lock on  $x$ . In  $\tau$ -serializability the two locks do not conflict if  $t_a + \min(\tau_b, \tau_x) > t_b$ . The tolerance  $\min(\tau_b, \tau_x)$  specifies how long the old value is useful, which may depend both on data semantics ( $\tau_x$ ) and on application semantics ( $\tau_b$ ).

These novel ideas are utilized with optimistic concurrency control protocol designed for RODAIN. Optimistic concurrency control protocols have the nice properties of being non-blocking and deadlock-free. These properties make them especially attractive for RTDBS. As conflict resolution between the transactions is delayed until a transaction is near completion, there will be more information available for making the choice in resolving the conflict. We have presented a method to reduce the number of transaction restarts and a new optimistic concurrency control protocol, called OCC-DATI.

## 7 Conclusion

We have described the RODAIN database architecture and main parts of its prototype implementation. The RODAIN database architecture is designed to meet the challenge of future telecommunication systems including Intelligent Networks, Telecommunication Management Network, and Telecommunications Information Networking Architecture.

In order to fulfill the requirements of the next generation telecommunications systems, the database architecture must be fault tolerant and support real-time transactions with explicit deadlines. The internals of the RODAIN DBMS are designed to meet the requirements of telecommunications applications including a real-time access to data, fault tolerance, distribution, object orientation, efficiency, flexibility, multiple interfaces, and compatibility with telecommunications practices. The requirements are, in some extent, conflicting. Therefore, the RODAIN database system is based on trade-offs; novel and innovative solutions are used only when old and exercised methods are found to be insufficient.

## References

- [1] R. Abbott and H. Garcia-Molina. Scheduling real-time transactions: A performance evaluation. *ACM Tr on Database Systems*, 17(3):513–560, Sep 1992.
- [2] I. Ahn. Database issues in telecommunications network management. *ACM SIGMOD Record*, 23(2):37–43, 1994.
- [3] R. F. M. Aranha, V. Ganti, S. Narayanam, C. R. Muthukrishnan, S. T. S. Prasad, and K. Ramamritham. Implementation of a real-time database system. *Information Systems*, 21(1):55–74, 1996.
- [4] T. F. Bowen, G. Gopal, G. Herman, and W. Mansfield Jr. A scale database architecture for network services. *IEEE Communications Magazine*, 29(1):52–59, Jan 1991.
- [5] R. G. G. Cattell, ed.. *The Object Database Standard: ODMG 2.0*. Morgan Kaufmann, San Francisco, Calif., 1997.

- [6] S. Cha, B. Park, S. Lee, S. Song, J. Park, J. Lee, S. Park, D. Hur, and G. Kim. Object-oriented design of main-memory dbms for real-time applications. In *2nd Int. Workshop on Real-Time Computing Systems and Applications*, pp 109–115, Tokyo, Japan, Oct 1995. IEEE Communication Society.
- [7] M. Chapman and S. Montesi. Overall concepts and principles of TINA. TINA-C Deliverable TB\_MDC.018\_1.0\_94, TINA Consortium, Feb 1995.
- [8] K. P. Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger. The notions of consistency and predicate locks in a database system. *Communications of the ACM*, 19(11):624–633, Nov 1976.
- [9] J. J. Garrahan, P. A. Russo, K. Kitami, and R. Kung. Intelligent network overview. *IEEE Communications Magazine*, 31(3):30–36, Mar 1993.
- [10] M. H. Graham. Issues in real-time data management. *The Journal of Real-Time Systems*, 4:185–202, 1992.
- [11] M. H. Graham. How to get serializability for real-time transactions without having to pay for it. In *Real-time System Symp.*, pp 56–65, 1993.
- [12] J. R. Haritsa, M. Livny, and M. J. Carey. Earliest deadline scheduling for real-time database systems. In *Proc. of the 12th Real-Time Symp.*, pp 232–242, Los Alamitos, Calif., 1991. IEEE, IEEE Computer Society Press.
- [13] Y. Huang and C. Kintala. Software implemented fault tolerance: Technologies and experience. In *The 23rd Int. Symp. on Fault-Tolerant Computing*, pp 2–9. IEEE, 1993.
- [14] S.-O. Hvasshovd, Ø. Torbjørnsen, S. E. Bratsberg, and P. Holager. The clustra telecom database: High availability, high throughput, and real-time response. In *Proc. of the 21th VLDB Conf.*, pp 469–477, San Mateo, Calif., 1995. Morgan Kaufmann.
- [15] ITU. *Common Management Information Protocol Specification for CCITT applications. Recommendation X.711*. ITU, Geneva, Switzerland, 1991–2.
- [16] ITU. *Principles for a Telecommunications Management Network. Recommendation M.3010*. ITU, Geneva, Switzerland, 1992.
- [17] ITU. *Principles of Intelligent Network. Recommendation Q.1201*. ITU, Geneva, Switzerland, 1993.
- [18] ITU. *Distributed Functional Plane for Intelligent Network CS-1. Recommendation Q.1214*. ITU, Geneva, Switzerland, 1994.
- [19] ITU. *Global Functional Plane for Intelligent Network CS-1. Recommendation Q.1213*. ITU, Geneva, Switzerland, 1994.
- [20] ITU. *Intelligent Network Distributed Functional Plan Architecture. Recommendation Q.1204*. ITU, Geneva, Switzerland, 1994.
- [21] ITU. *Draft Q.1224 Recommendation IN CS-2 DFP Architecture*. ITU, Geneva, Switzerland, 1996.
- [22] H. V. Jagadish, D. Lieuwen, R. Rastogi, Avi Silberschatz, and S. Sudarshan. Dalí: A high performance main memory storage manager. In *Proc. of the 20th VLDB Conf.*, pp 48–59, 1994.
- [23] Y.-K. Kim and S. H. Son. Developing a real-time database: The StarBase experience. In A. Bestavros, K. Lin, and S. Son, eds, *Real-Time Database Systems: Issues and Applications*, pp 305–324, Boston, Mass., 1997. Kluwer.
- [24] Y. Kiriha. Real-time database experiences in network management application. Tech. Report CS-TR-95-1555, Stanford University, USA, 1995.
- [25] J. Kiviniemi, T. Niklander, P. Porkka, and K. Raatikainen. Transaction processing in the RODAIN real-time database system. In A. Bestavros and V. Fay-Wolfe, eds, *Real-Time Database and Information Systems*, pp 355–375, London, 1997. Kluwer Academic Publishers.
- [26] T. Kuo and A. K. Mok. Application semantics and concurrency control of real-time data-insensitive applications. In *Proc. of Real-Time System Symp.*, pp 76–86, 1993.
- [27] M. Lehr, Y.-K. Kim, and S. Son. Managing contention and timing constraints in real-time database system. In *Proc. of 16th IEEE Real-Time Systems Symp.*, Pisa, Italy, Dec 1995.

- [28] K.-J. Lin. Consistency issues in real-time database systems. In *Proc. of the 22nd Hawaii Int. Conf. on System Sciences*, pp 654–661, 1989.
- [29] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, Jan 1973.
- [30] OMG. *CORBA: Common Object Request Broker Architecture and Specification*. Number Number 91.12.1. Revision 2.0 in OMG Document. John Wiley & Sons, New York, N.Y., 1996.
- [31] OMG. *The Common Object Request Broker: Architecture and Specification*. Number Revision 2.2 in OMG Document. John Wiley & Sons, New York, N.Y., 1998.
- [32] OMG. *CORBA services: Common Object Service Specification*. Number Dec 1998 in OMG Document. John Wiley & Sons, New York, N.Y., 1998.
- [33] P. Porkka and K. Raatikainen. CORBA access to telecommunications databases. In D. Gaïti, ed., *Intelligent Networks and Intelligence in Networks*, pp 281–300, Paris, France, 1997. Chapman&Hall.
- [34] D. Pountain. The Chorus microkernel. *Byte*, pp 131–138, Jan 1994.
- [35] K. Raatikainen. Real-time databases in telecommunications. In A. Bestavros, K.-J. Lin, and S. H. Son, eds, *Real-Time Database Systems: Issues and Applications*, pp 93–98. Kluwer, 1997.
- [36] K. Raatikainen, T. Karttunen, O. Martikainen, and J. Taina. Evaluation of database architectures for intelligent networks. In *Proc. of the 7th World Telecommunication Forum (Telecom 95), Technology Summit, Volume 2*, pp 549–553, Geneva, Switzerland, Sep 1995. ITU.
- [37] K. Raatikainen and J. Taina. Design issues in database systems for telecommunication services. Report C-1995-16, University of Helsinki, Dept. of Computer Science, Helsinki, Finland, Sep 1995.
- [38] K. Ramamritham and C. Pu. A formal characterization of epsilon serializability. *IEEE Tr on Knowledge and Data Engineering*, 7(6), Dec 1996.
- [39] L. Sha, J. P. Lehoczky, and E. D. Jensen. Modular concurrency control and failure recovery. *IEEE Tr on Computers*, 37(2):146–159, Feb 1988.
- [40] J. Siegel, ed.. *CORBA Fundamentals and Programming*. John Wiley & Sons, New York, N.Y., 1996.
- [41] J. A. Stankovic and S. H. Son. Architecture and object model for distributed object-oriented real-time databases. In *IEEE Symp. on Object-Oriented Real-Time Distributed Computing (ISORC'98)*, pp 414–424, Kyoto, Japan, Apr 1998.
- [42] J. A. Stankovic and W. Zhao. On real-time transactions. *ACM SIGMOD Record*, 17(1):4–18, Mar 1988.
- [43] J. Taina and K. Raatikainen. Experimental real-time object-oriented database architecture for intelligent networks. *Engineering Intelligent Systems*, 4(3):57–63, Sep 1996.
- [44] J. Taina and K. Raatikainen. Database usage and requirements in intelligent networks. In D. Gaïti, ed., *Intelligent Networks and Intelligence in Networks*, pp 261–280, Paris, France, 1997. Chapman&Hall.
- [45] Ø. Torbjørnsen, S.-O. Hvasshovd, and Y.-K. Kim. Towards real-time performance in a scalable, continuously available telecom DBMS. In *Proc. of the First Int. Workshop on Real-Time Databases*, pp 22–29. Morgan Kaufmann, 1996. <http://www.eng.uci.edu/ece/rtdb/rtdb96.html>.
- [46] V. Wolfe, L. DiPippo, J. Prichard, J. Peckham, and P. Fortier. The design of real-time extensions to the open object-oriented database system. Technical report TR-94-236, University of Rhode Island, Department of Computer Science and Statistics, Feb 1994.
- [47] P. S. Yu, K.-L. Wu, K.-J. Lin, and S. H. Son. On real-time databases: Concurrency control and scheduling. *Proc. of the IEEE*, 82(1):140–157, Jan 1994.