University of Helsinki
Department of Computer Science
Series of Publications C, No. C-1995-30

An Experimental Database Architecture for Intelligent Networks*

Juha Taina (University of Helsinki), Mika Rautila (Telecom Finland), and Kimmo Raatikainen (Univcersity of Helsinki)

Helsinki, June 1995

Report C-1995-30

University of Helsinki
Department of Computer Science
P. O. Box 26 (Teollisuuskatu 23)
FIN-00014 University of Helsinki, FINLAND

* This report is based on work done in Darfin-project (Project No 30018) funded by Telecom Finland.
The papers in the series are intended for internal use and are distributed by the author. Copies may be ordered from the library of Department of Computer Science.

An Experimental Database Architecture for Intelligent Networks[†]

Juha Taina (University of Helsinki), Mika Rautila (Telecom Finland), and Kimmo Raatikainen (Univcersity of Helsinki)

Report C-1995-30
14 pages
Helsinki, June 1995

Abstract. Databases in telecommunications have become an important research area in recent years. In this paper we will introduce an experimental database architecture for Intelligent Networks use. The architecture is a real time distributed object oriented database architecture that may be used to implement Service Control Functions and other telecommunications network elements that need database services. We will also introduce a prototype database architecture that is based on the final product. With the prototype we hope to gain insight of the actual database service usage in Intelligent Networks, how well real time databases and object orientation suit each other, and how a Service Data Function may be implemented with our architecture.

Key Words: Real-Time Databases, Concurrency Control, Relaxed Serializability, Main Memory Databases, Telecommunication Software

CR Classification: C.2.1, H.2.4

[†]This report is based on work done for Darfin-project (Project No 30018) funded by Telecom Finland.

Contents

1. Introduction

In the recent years the Intelligent Network (IN) architecture has reached a relatively stable state. With IN platform it is now possible to create new services fast and reliably. Also the recent standardization, first in CCITT and currently in ITU-T, has stabilized the field. The IN Capability Set 1 (CS-1) has been available for quite a while and now CS-2 and CS-3 are under construction. The different revisions to the standards and new capability sets have an effective foundation in the IN research that has been active for over a decade.

Recently IN database research has become one of the most important research areas in IN field. This is due to the fact that IN standards leave the database implementation aspects relatively undefined.

In project Darfin (Database ARchitecture For Intelligent Networks) we are working on a full scale database architecture for Intelligent Networks. In this paper we call the final database architecture Darfin DBMS and the prototype database architecture Darfin Proto, respectively.

In [Raat94a] we have quoted what the ITU-T IN standards state of database access in IN. In the IN Conceptual Model (INCM) database access is defined in the Distributed Functional Plane and in the Physical plane. The highest planes, Service Plane and Global Functional Plane, do not define database access.

In the Distributed Functional Plane the Functional Entity *Service Data Function (SDF)* contains customer and network data for real time access by the Service Control Function (SCF) in the execution of an IN provided service. It interfaces and interacts with SCF, interfaces and interacts with other SDFs, and it is managed, updated and/or otherwise administered by a Service Management Function (SMF). The SDF must contain at least the following Service-Independent Building blocks (SIB): Log Call Information, Screen, Service Data Management, Status Notification, and Translate.

In the Physical Plane the Physical Entity *Service Data Point (SDP)* contains the customer and network data that must be stored for IN services. Functionally the SDP contains an SDF. The SDP may be accessed from a Service Control Point (SCP), from a Service Management Point (SMP), or through the signaling network. It can access other SDPs in its own or other networks. A SDP consists of at least a SDF that implements the SDP functionality.

The total IN network in the Physical Plane may consist of several SDPs all of which have different SDF functional entities. This implies that the whole IN data access is distributed into several SDFs. A SDF is the central point for database implementation. It gives an IN compatible image of the underlying database architecture that is used to implement it. In our terminology, we call a SDF an application. The application uses services that the actual database architecture offers to fulfil the service requests that are sent to it.

A SDF, like any other IN distributed function plane entity, may be present in several different physical entities. Usually a SDP consists of a single SDF. Yet a SDF may be present in a SCP with a SCF, or even in a Service Switching Point (SSP) with a Service Switching Function (SSF). If we designed a pure SDP architecture it would not be sufficient for a stand-alone SDF function. If we implemented a pure SDF we could not use the database architecture for other telecommunications use, such as TMN. We chose to design and implement a general database architecture for telecommunications use. The database serves several database applications. A SDF implementation is an example of a serviced application.

The rest of this paper is organized as follows. In Section 2 we summarize the requirements that we consider important for our IN database and prototype database architectures. In Section 3 we introduce an overview of our distributed database architecture. In Section 4 we introduce the general architecture of a single node in our distributed database, and in Section 5 we introduce the services and structure of the database node architecture. Section 6 is the summary.

2. Requirements

2.1 Final database architecture requirements

Our interests are in an architecture that can be used to implement an efficient SDF. Such an architecture must answer the key requirements for data access that are defined in the IN standardization.

In [Tain95] we have identified the key requirements for 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. A 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 among 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.

Data replication. In [CCIT92b] 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. In such an environment data replication is an effective way to reduce slow distributed operations. 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 [CCIT92a] it is stated that the best long term architecture for IN is probably object oriented. In [Raat94a] we propose a stronger claim. We are convinced that today object modeling is the easiest way to satisfy the IN modeling needs.

Object directories. In [CCIT92c] 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) [CCIT92d]. 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 [ChGa94]. Other possible interfaces to the SDF are X.700 management interface for management functions, and Open Distributed Processing (ODP) interface for distributed operations. This implies that the set of request the SDF and the underlying database architecture must be able to answer is not fixed.

Fault tolerance. Like in data replication, fault tolerance is implied from the fact that SDFs support real time access to data. 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 a SDF must be at most seconds in a year [Tain94b]. 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 [CCIT92b], 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 states the requirements that may be derived from the ITU-T and CCITT recommendations. It also 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.*

2.2 Prototype database requirements

The requirements listed in the previous section are for the final database architecture for Intelligent Networks. 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 the following: *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.*

Statistics of the database usage. Although quite a few papers have been written about IN databases and database architectures, few papers have dealt with the actual implementation and request distribution to SDFs. We are interested to gather data of the actual database usage. This includes information of the request distribution, transaction lengths, transaction sizes, and database element references. We hope to gain insight of the actual database architecture design from the gathered data. For instance, the best way to gain performance is to use a pure main memory database. However, the needed database sizes in IN may not fit into any main memory databases without high costs. A hybrid solution where some data resides permanently in main memory and some in a disk is a better alternative. We hope to find out from the statistical information how data should be clustered between main memory and disks.

We already know a lot of database usage in CS-1 services. In [Raat94b] we have identified the principal database access scenarios in IN CS-1 services and service features.

1) *Retrieval of structured objects from persistent subscriber data objects.* Some of the retrievals trigger a later update.
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.
3) *Verification of Personal Identification Number.* For security reasons the PIN verification must be done in the database system when services are globalized.
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.

The list above is based on the CS-1 requirements and definitions. The CS-2 will at least define management functions which implies that the database architectures must also support management. The CS-3 and CS-2 together define mobility aspects, especially in Universal Mobile Telecommunication System (UMTS) [Mitt94]. Both the management operations and the mobility aspects must be considered already in the current stage of IN database architecture design.

Combining real time transactions and object orientation. Like the final Darfin DBMS, Darfin Proto is a real time distributed object oriented database architecture. The main aspects of the database architecture hold in the prototype. We have lowered the requirements of fault tolerance, data distribution, and database interfaces. Also the chosen object model for the prototype is a subset of the final object model.

A lot of research is going on in object oriented databases. The general structures are well known and accepted but the details are not yet clear. Also usually object orientation and real time requirements are two distinct research areas. We are interested to know how well we can combine the two areas. With the prototype database we are able to test different real time transaction scheduling methods and priorities. We are also able to combine areas of data storage, concurrency control, and fault tolerance to gather data for the possible bottlenecks in the architecture.

Learning implementation methods for SDFs and other network elements using the database architecture. From the application designer's point of view, we are interested to know how well the database architecture may be used to implement SDFs and other applications that use database services. All database requirements raise from the application designer's needs. A good architecture is able to support most or all of the requirements efficiently. We hope to gain information of what the final architecture must support in order to accomplish the needs.

3. Global distributed 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 (figure 1).

A single application may communicate with several database nodes or only with a single node. If the application uses single database node services then the database node is an entry point to the whole distributed database. All distribution occurs internally. 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 of 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.

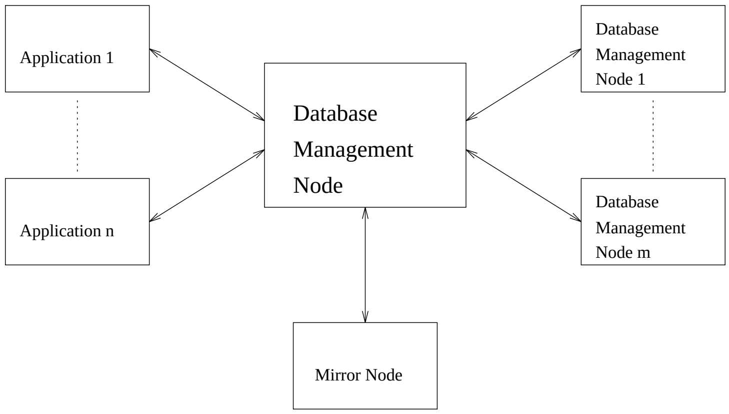

Figure 1: Overview to Darfin distributed database architecture

Usually this is not necessary. For instance, it is not necessary to store all IN database usage statistics into the mirror node.

Data distribution and replication to several database nodes and the nature of each node in the distributed database is an interesting research area. There are several questions in data distribution in IN environment.

1) *How much distributed operations are needed?* We believe that most operations are local to a single node. Our database operation list in CS-1 imply the same conclusion. Yet not all operations are local, for instance when user profiles are updated on several nodes.
2) *How complex are the distributed operations?* We believe that the distributed operations are not very complex. Usually they consist of operations that are directed only to a single object in another node, such as simple updates or retrieves. The regular distributed databases may have very complex join operations that need a lot of information from several nodes.
3) *How real time requirements should be met in distributed operations?* We believe that distributed requests are a minority in IN database usage. Yet also the distributed operations should be served in real time. We do not know yet how well distributed operations behave in IN environment.
4) *How much data replication is needed?* We believe that it is probably better to replicate some data to several nodes to reduce distributed reads.

Our conclusion is that data distribution and replication play a vital role but still most operations are local on a single node. In Darfin Proto we concentrate on requests that only concern a single database node. However, we keep track of distributed operations for statistical analysis, and perhaps add support for them in the future releases of Darfin Proto. Full Darfin DBMS architecture will support distributed operations.

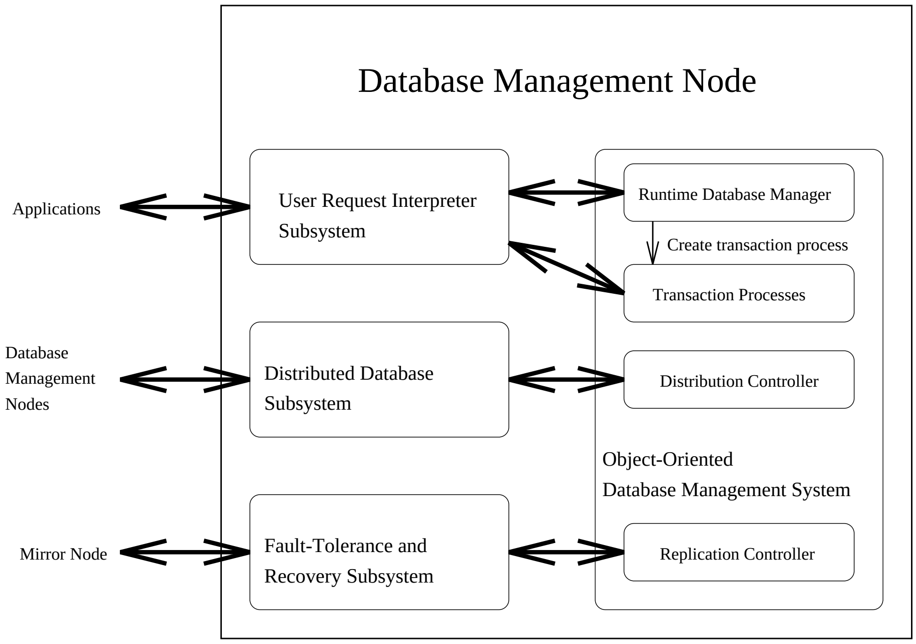

Figure 2: A database node architecture in Darfin

4. Node database 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 (DDS), and Fault-Tolerance and Recovery Subsystem (FTRS) (figure 2). 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: in the operating system level the operating system scheduler schedules system and subsystem processes. In 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 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 ODMG-93 object database standard [Catt94] for 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 future commercial database management systems that follow the same ODMG-93 standard.

In [Tain94a] we compared OMG, ODMG, X.500, and X.700 models for suitability for IN use. We concluded that the ODMG object model is flexible enough for the logical object model in Darfin DBMS. It supports objects, object types, object identities, inheritance, operation overloading, and late binding among other things. It is a superset of the OMG object model [OMG92] which makes the model flexible to use in OMG object request brokers and ODP communication channels, both of which are important in distributed telecommunications use.

The OODBMS architecture consists of a set of database processes that communicate with each other: Runtime Database Manager (RDM), Distribution Controller (DC), Replication Manager (RM), and Transaction processes. 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 applications, other database nodes (where again a subsystem takes care of processing the services), and the mirror node. For instance, the RDM is responsible of creating new transactions upon request. The RDM uses database services of objects and object locations to resolve an estimation of the transaction length. The actual database implementation offers the service of the objects and object locations.

Runtime Database Manager (RTDM). The RTDM 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 RTDM continues to listen the URIS for new transaction requests.

Distribution Controller (DC). The DC is a transaction process that sends and receives distributed operations. It is responsible of 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 of the actual network communication to other database nodes. The DC also serves DDS requests that are sent from other database nodes.

Replication Manager (RM). The RM is a transaction process that maintains data replication. It hides the replication method from the OODBMS. Every update operation to the database triggers the RM. It forwards the update to the FTRS that is responsible of mirror node maintenance.

Transaction Processes. 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 of 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 of 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 DM is responsible of 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 of fault tolerance and recovery control. It communicates with the mirror node and with the RM 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 on the same state (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.

5. OODBMS process services

The most important element of the Darfin DBMS and Darfin Proto is the OODBMS and the services that it offers to its processes: the transaction processes, runtime database manager, distribution manager, and replication manager. The services that the processes may access are called the ODMG Interface Layer. The other deeper layers that implement the interface

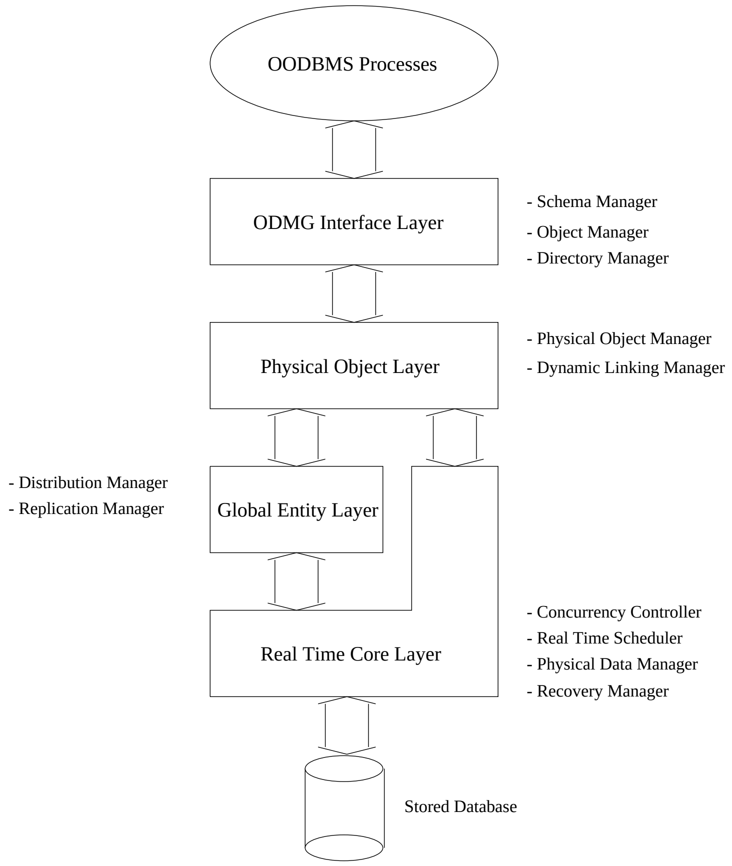

Figure 3: Darfin OODBMS process service layers

layer services are Physical Object Layer, Global Entity Layer, and Real Time Core Layer (figure 3). Every layer consists of a set of managers that implement layer functionality.

5.1 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 managers: Schema Manager, Object Manager, and Directory Manager. Together the manages offer a full ODMG compliant interface to the database.

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

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

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

5.2 Physical Object Layer

The Physical Object Layer implements the ODMG interface layer services. 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 managers: Physical object manager and dynamic linking manager.

Physical Object Manager The Physical Object Manager manages the physical objects. It offers services for object creation, deletion, access, object attribute access, and operation access. There may be other services that the ODMG layer needs. This manager is the core manager for a logical object model. It uses services of the lower layers to implement the physical object managing.

Dynamic Linking Manager Objects may have executable operations. The Dynamic Linking Manager is responsible of linking new executable code to the database when a new object operation is created.

5.3 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 managers: Distribution manager and Replication Manager.

Distribution Manager The Distribution Manager manages distributed operations on objects. It is mostly used by the Distribution Controller, but also the Physical Object Manager uses the Distribution Manager when it needs to access an object in a different database node.

Replication Manager The Replication Manager 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.

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

5.4 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. Respectively, the managers are Concurrency Controller, Real Time Scheduler, Physical Data Manager, and Recovery Manager.

Concurrency Controller The Concurrency controller is responsible of 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 2PL-HP algorithm. In 2PL-HP, a higher priority process that claims a lock kills a lower priority process that holds it. the lower priority process is restarted if it has not exceeded its deadline.

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

Physical Data Manager The Physical data manager manages the physical data storages: object data storage and metadata store. It is the only manager 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. It is possible to use other data store architectures by changing the Physical data manager. The rest of the OODBMS architecture need not to be altered.

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

6. Summary

We have introduced Darfin DBMS and Darfin Proto architecture for Intelligent Networks Database use. The architecture may be used in implementing a SDF, or any other telecommunications element that needs database services.

We have identified the needs that the ITU-T recommendations list for IN database services and designed the architecture accordingly. The most important requirements are object orientation, real time access to data, and fault tolerance. The needs are difficult to fulfil but we believe that a distributed real time object oriented database architecture is able to do that.

There is not enough information available of database access in IN environment. Due to that we are implementing a prototype database architecture that can be used to gather database usage data. The prototype architecture is a reduced version of the final Darfin DBMS architecture.

Our 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 architecture 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. Yet a lot of work and research is ahead before we have a full working database architecture for Intelligent Networks.

References

[Catt94] Cattel R. G. G. (Ed.), Object Database Standard: ODMG–93 Release 1.1 Morgan Kaufman Publishers, 1994

[CCIT92a] Principles of Intelligent Network Architecture. Recommendation I.312 / Q.1201, CCITT, 1992

[CCIT92b] Intelligent Network - Distributed Functional Plane Architecture. Recommendation Q.1204, CCITT, 1992

[CCIT92c] Distributed Functional Plane for Intelligent Network CS1. Recommendation Q.1214, CCITT, 1992

[CCIT92d] Interface Recommendation for Intelligent Network CS1. Recommendation Q.1218, CCITT, 1992

[ChGa94] Chatras B. and Gallant F., Protocols for Remote Data Management in Intelligent Networks CS1. Intelligent Network'94 Workshop Record – Volume 1 (1 of 2). IEEE, 1994

[Mitt94] Mitts H., Use of Intelligent Networks in the Universal Mobile Telecommunication System (UMTS). pp. 125 – 134 in Proceedings of the Third Summer School on Telecommunications – Volume I: Workshop on Intelligent Networks, August 8–9, 1994. Lappeenranta University of Technology, 1994

[OMG92] OMG, Object Management Architecture Guide – Revision 2.0 - Second Edition. Object Management Group, 1992

[Raat94a] Raatikainen K. E. E., Database Access in Intelligent Networks. pp. 163 – 184 in Proceedings of the Third Summer School on Telecommunications – Volume I: Workshop on Intelligent Networks, August 8–9, 1994. Lappeenranta University of Technology, 1994

[Raat94b] Raatikainen K. E. E., Information Aspects of Services and Service Features in Intelligent Network Capability Set 1. Technical report, University of Helsinki, Department of Computer Science, 1994

[Tain94a] Taina J., Evaluation of OMG, ODMG, X.500, and X.700 Data Models. Technical report, University of Helsinki, Department of Computer Science, 1994

[Tain94b] Taina J., Problem Classes in Intelligent Network Database Design. pp. 185 – 200 in Proceedings of the Third Summer School on Telecommunications – Volume I: Workshop on Intelligent Networks, August 8–9, 1994. Lappeenranta University of Technology, 1994

[Tain95] Taina J., Requirements Analysis for Database Services in Telecommunications. Technical report, University of Helsinki, Department of Computer Science, 1995