University of Helsinki
Department of Computer Science
Series of Publications C, No. C-1995-29

Evaluation of Database Architectures for Intelligent Networks*

Juha Taina (University of Helsinki), Tapani Karttunen (Telecom Finland), Olli Martkainen (Telecom Finland), and Kimmo Raatikainen (University of Helsinki)

Helsinki, May 1995

Report C-1995-29

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.

Evaluation of Database Architectures for Intelligent Networks[†]

Juha Taina (University of Helsinki), Tapani Karttunen (Telecom Finland), Olli Martkainen (Telecom Finland), and Kimmo Raatikainen (University of Helsinki)

Report C-1995-29
12 pages
Helsinki, May 1995

Abstract. Databases will have an important role in telecommunication services based on the Intelligent Network architecture. The databases hold the persistent information needed in operations and management of telecommunication networks and services. We examine database architectures that can be integrated into teleoperators' service provision architectures. Our premises are derived from the IN Long-Term Architecture framework. The database architectures examined are based on object-oriented information models. We evaluate the feasibility of approaches based on the ITU-T Recommendations in the X.500 Series and on the industrial standards called the ODMG-93 Standard and the Object Model of OMG.

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

Databases will have an important role in telecommunication services based on the Intelligent Network architecture. The databases hold the persistent information needed in operations and management of telecommunication networks and services. The performance, reliability, and availability requirements of the data access operations are stringent. Thousands of retrievals must be executed in a second. Allowed down-time is only a few seconds in a year. The globalization of services implies that databases managed by different operators must interoperate. This requires that the systems are open, use standardized protocols, are protected against misuse and intruders, and have a common logical view of information.

In this paper we examine database architectures that can be integrated into teleoperators' service provision architectures. Our premises are derived from the IN Long-Term Architecture (IN LTA) framework. The key requirements include 1) an OSI-aligned protocol architecture for all interfaces, 2) compatibility with the OSI Reference Model of Open Distributed Processing (RM-ODP), and 3) adapting ITU-T Recommendations for Telecommunications Management Network (TMN). In particular, the last two requirements call for an object-oriented approach.

The database architectures examined are based on object-oriented information models. Object-oriented modelling is currently claimed to be the easiest way to satisfy the modelling needs of IN LTA through the use of abstract object modelling concept. However, the field of object-oriented database management systems (ODBMSs) is not yet mature. There are several standardization activities going on. We evaluate the feasibility of approaches based on the ITU-T Recommendations in the X.500 Series and on the industrial standards called the ODMG-93 Standard [Cattell 1994] and the OMG [OMG 1992a] Object Model.

The rest of the paper is organized as follows. In Section 2 we present our premises. In Section 3 we describe the essential features in the three object models. Finally, in Section 4 we give our comparison and draw the conclusions.

2. Premises

In addition to the current ITU-T Recommendations in Q.1200 Series, the Basic Reference Model of ODP and the concepts of TMN create the architectural framework with which the database architecture must be compatible. The architectural framework sets the requirements for external interfaces that the database architecture may have to support. In this section we first give a high-level information model for telecommunication services and discuss the database services needed in telecommunication. Later we briefly summarize the IN functional architecture, RM-ODP, and TMN.

2.1 High-Level Information Model

On an abstract level we can interpret that a telephone call is controlled by call processing logic and data. The data can be termed as *Call Processing Data Object*. It is a dynamic object that is created during the call establishment. During the call the object may change as a result of the so-called mid-call service control actions. Logically the data object can be divided into two parts:

Calling Party Object and *Called Party Object*. Moreover, the data object may have attributes that trigger actions to modify and/or to create persistent data objects when the call is terminated.

Fig. 1 summarizes our abstract information model of a telephone call. It should be noted that the call is not necessarily between two telephones. The called party may also be an information processing system. User management of services is a typical case in which the called party is an information processing system.

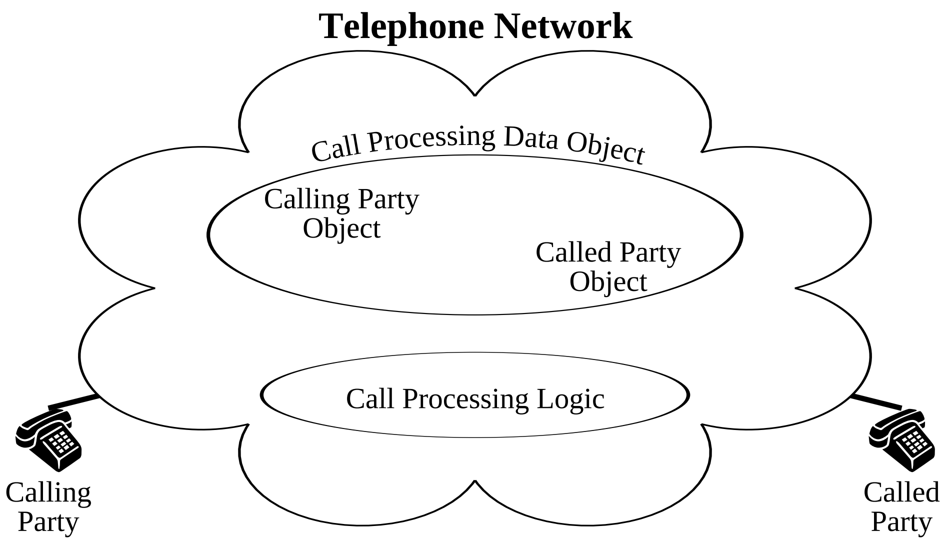

Figure 1: A high-level information model for telephone call

The Call Processing Data Object is typically created in two phases. The first phase consists of creating the Calling Party Object. When the switch detects that the hook is lifted up, a query is triggered. The query builds up the object by retrieving the service profile for the originating line from the persistent subscriber data object.

The Called Party Object is created in the second phase. This object depends on the access number dialled. Typically, the dialled digits are given as the key in the query that creates the Called Party Object. The query builds up the object by retrieving the service profile for the destination line from the persistent subscriber data object.

2.2 Anticipated Database Needs

In [Raatikainen 1994b] 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 owner of a telephone number 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 the databases' 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 its part of the Management Information Base.

2.3 IN Functional Architecture

On the *distributed functional plane* the IN architecture is described in terms of *functional entities*. In that model the database services are available in SDF. Recommendatation Q.1204 specifies that the SDF contains customer and network data for real time access. Recommendation Q.1211 augments the functional specification of the SDF by requiring that the SDF provides consistency checks on data. These two requirements imply that the SDF should provide *real-time transaction processing capabilities*.

The database users, clients of SDF, are SCFs, SMFs, and other SDFs. Fig. 2 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 Recommendation Q.1214, respectively. The outlined internal structure of the SMF and its impact on the SDF are based on the principles of OSI Management. Our earlier work [Raatikainen 1994a] provides a detailed description of database access in the IN, especially in the CS-1.

2.4 Basic Reference Model of ODP

Open Distributed Processing is a joint standardization initiative by ISO and ITU-T. The Basic Reference Model of ODP 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 distributed systems. 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.

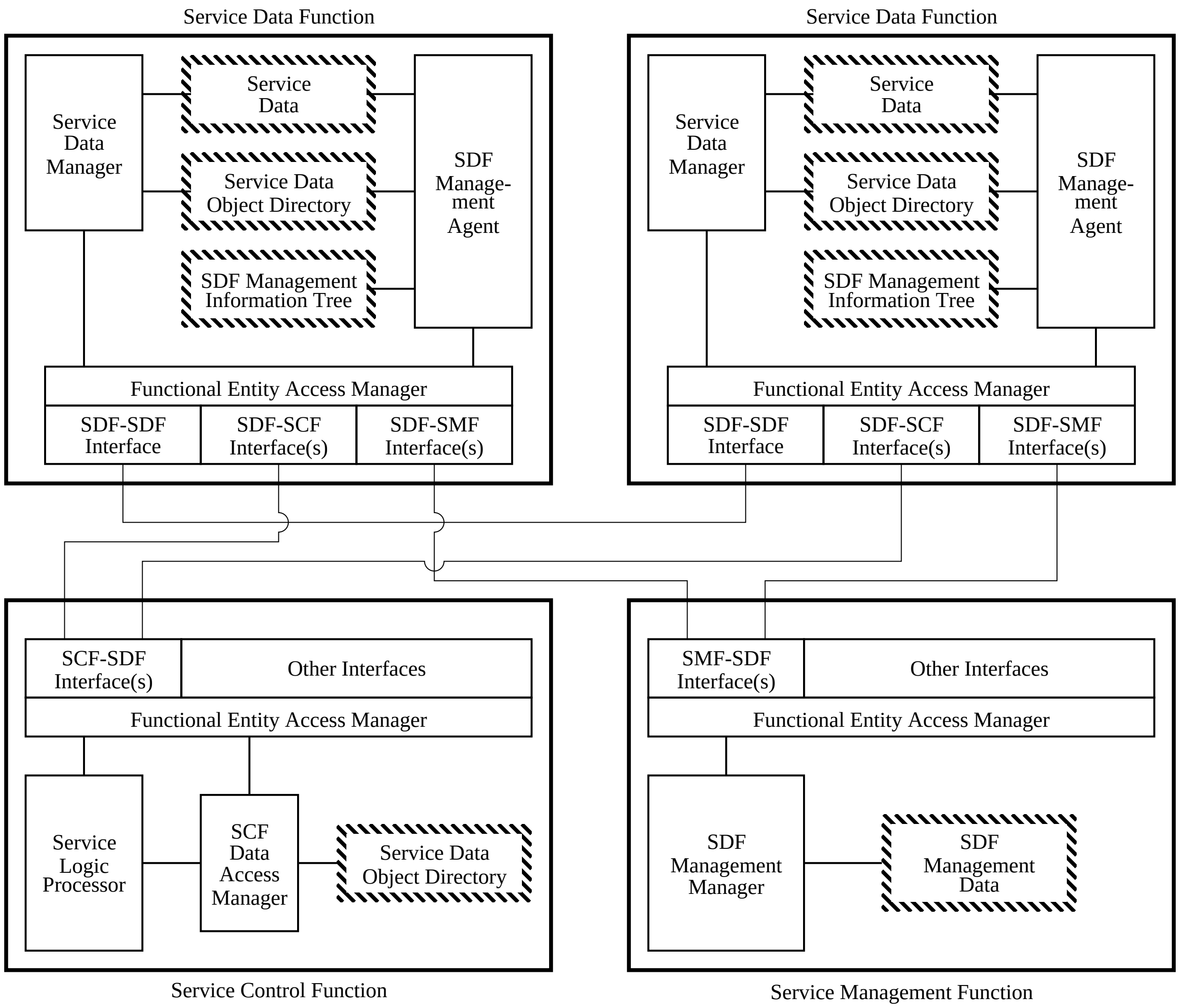

Figure 2: Elements of database servers and clients in the IN functional architecture

2.5 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 (X.700). The fundamental idea in X.700 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.

In brief, the principles of OSI Management (and TMN) require that the database system contains the functionality of an OSI Management Agent.

3. Object Models

Object-oriented modelling is pointed out in Recommendation Q.1201: *"The use of object modelling could satisfy the modelling needs of IN long-term architecture by the use of abstract object modelling concept."* Our claim is even stronger. We believe that today the object technology is the easiest way to satisfy those modelling needs. In this Section we present the essential features of three existing objects models: OMG, ODMG, and X.500.

3.1 The OMG Object Model

In the *Object Management Group, Inc.* (OMG) the ultimate goal is to develop and use integrated software systems. The methodology to be used in software development should 1) support modular production of software, 2) encourage reuse of code, 3) allow useful integration of different technologies, and 4) enchance long-range software maintenance. It is believed in the OMG that the object-oriented approach best supports the goals above.

In the OMG *Object Management Architecture*, outlined in Fig. 3, the *Object Request Broker* (ORB) is the communications heart of the standard. Commercially the ORB is referred to as *Common Object Request Broker Architecture* [OMG 1992b] (CORBA). CORBA provides an infrastructure that allows objects to communicate, independently of the platforms and implementation techniques. The most important feature of the CORBA specification is its Interface Definition Language (IDL). The applications use IDL to specify the various interfaces they intend to offer to other applications via the ORB layer.

The overall technical goal in the OMG is to adopt interface and protocol specifications that define an object management architecture which supports interoperable applications based on distributed interoperating objects. The list of detailed technical objectives include:

- distribution transparency,
- scalability,
- multiple naming contexts,
- queries based on object names, attributes, and relationships,
- discretionary access control,
- concurrency control,
- atomic transactions,
- a high level of availability,
- versioning of objects,
- notification of events to interested objects.

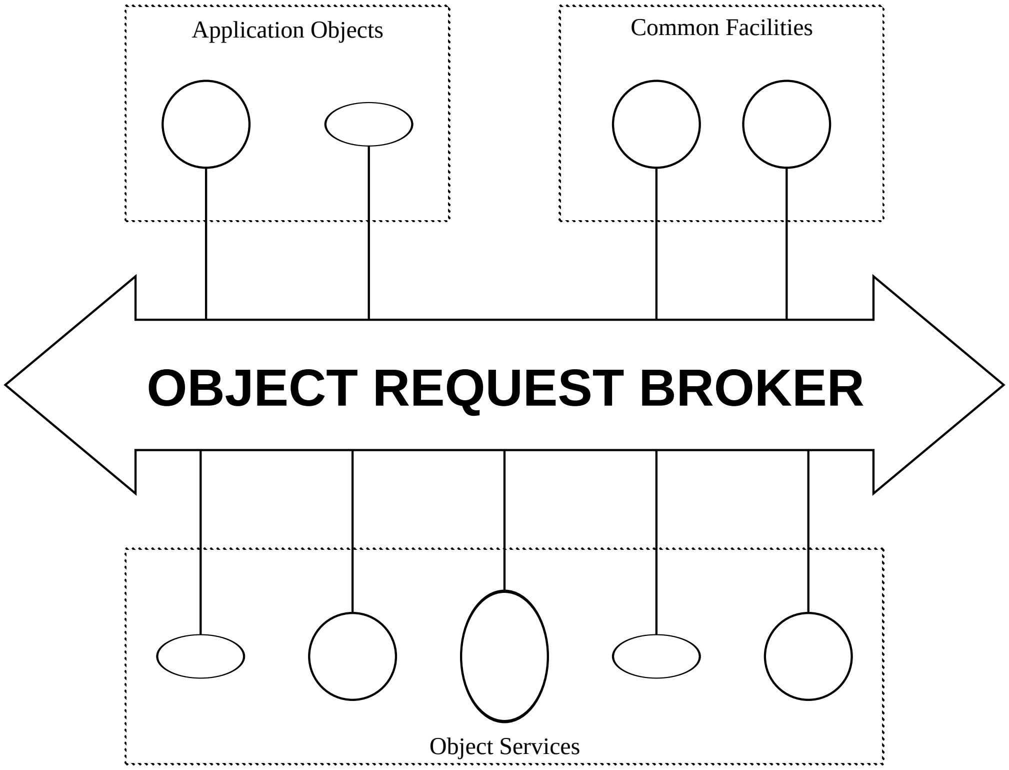

Figure 3: Overview of the OMG Object Management Architecture

The OMG Object Model defines a common semantics that specifies the externally visible characteristics of objects. These characteristics are described by an interface consisting of operation signatures. The operation signatures are used to model both the visible object behaviour and the visible object state.

In OMG the broad statement of the goals is to define an object model that facilitates portability (particularly design portability) of applications and type libraries, and interoperability. The Object Model currently focuses on a *common object semantics*. It defines a core set of requirements that must be supported. This set is called the *Core Object Model* and is the basis for portability and interoperability.

In addition to the Core Object Model, the OMG Object Model allows extensions, called *components*, which are compatible extension to the Core Object Model. The Object Model also defines a mechanism, called *profiles*, to group pertinent components. The model is based on a small number of basic concepts: *objects*, *operations*, *types*, and *subtyping*:

Object An object can model any kind of entity. A basic characteristic of an object is its distinct identity, which is independent of the properties or behaviour of the object.

Operation The set of operations associated with an object characterizes the behaviour of the object. The consequences of an operation invocation (*request*) can include:

1) an immediate set of results, and
2) side effects, manifested in changes of state.

The current Core Object Model does not include exceptions.

Type Objects are created as instances of types. Type characterizes the behaviour of its instances through describing the operations which can be applied to those objects. Types are related to one another by supertype/subtype relationships.

Subtyping Subtyping is a relationship between types based on their interfaces. It defines the rules by which objects of one type are determined to be acceptable in contexts expecting another type.

State State captures the information needed to affect operations. In the Core Object Model, operations are used to model the external interface to the state while *attributes* and *relationships* are used to model the externally visible declarations of the state in a more concise manner.

3.2 The ODMG Object Model

In the *Object Database Standard ODMG-93* (Release 1.1) the primary goal is to provide a standard Object Database Management System (ODBMS) so that applications using an ODBMS are portable. Therefore, data schema, programming language binding, data manipulation language, and data query language must be portable. In addition, ODMG-93 standard aims in interoperability between heterogeneous distributed databases communicating through the OMG Object Request Broker.

An ODBMS is a database management system that integrates database capabilities — persistent data, concurrency control, data recovery, associative quires, etc. — with the capabilties of object-oriented programming languages. An ODBMS makes the database objects to appear as objects in a programming language. The major components of an ODBMS include *object model*, *object definition language*, *object query language*, and *programming language bindings*.

The fundamental concepts in the ODMG Object Model are *object*, *type*, *operation*, *property*, *relationship*, and *attribute*. Objects, which are the basic modeling primitives, have a type. All objects of the same type have a common behaviour and a common range of states. The behaviour of an object is defined through a set of operations that can be executed on an object of that type. The state of an object is identified by the values it carries in the set of properties. The properties are either attributes of the object itself or relationships between the object and one or other other objects.

The major components of type definition are *type properties*, *instance properties*, and *instance operations*. The type properties include *supertypes*, *extents*, *keys*, and *persistency*. Instance properties are the properties in which an object carries values. Instance operations are the operations which can be executed on the object. Properties and operations are specified by *signatures*.

The fundamental concepts are summarized below.

Supertypes In the ODMG Object Model types are related in a subtype-supertype hierarchy. The subtype inherits all attributes, relationships, and operations that are defined on the supertype. However, the definition of subtype may also refine the properties (i.e. attributes and relationships) and operations which it inherits. In addition, the subtype may add additional properties and operations.

Extents The set of all instances of a given type is termed its extent. The primary usage of extent is that the programmer asks the system to maintain an index of instances of the given type.

Keys A key is a property or a set of properties that uniquely identify the individual instances. A type may have several keys.

Persistency Objects are either *persistent* or *transient*.

Attribute signatures Each signature defines the name of the attribute and the type of its legal values.

Relationship signatures Each signature defines the type of the other object or set of objects involved in the relationship and the name of a *traversal function* that is used to refer to the related object or set of objects. The cardinality of the relationship can be 1) one-to-one, 2) one-to-many, or 3) many-to-many.

Operation signatures Each signature defines the name of the operation, the names and types of the arguments, the names and types of the returned values, and the names of the *exceptions* (error conditions) that the operation can raise.

In ODMG all access, creation, modification, and deletion of persistent objects must be done with a transaction. In ODMG transaction are units of atomicity, consistency, and integrity. The object model supports a nested transaction model. For example, each remote operation is implicitly executed within a nested transaction. As default the ODMG model supports the pessimistic (lock-based) concurrency control scheme. In the current standard the transaction semantics only apply on persistent objects. Therefore, aborting a transaction does not restore the state of modified transient objects.

Persistent objects of a given set of types can be stored in a database. The *schema* includes the definitions of a given set of types. The names of the types in the schema and their associated extents are global to the database. The names of the so-called *"root objects"* are also global. These global names provide entry points into the database.

The *Object Query Language* (OQL) provides declarative access to objects. OQL is not computationaly complete. However, queries can invoke methods, that is operations defined on the objects. Moreover, queries can be used from any supported programming language. OQL provides high-level primitives that deal with sets of objects. OQL also provides primitives which deal with structures and lists. The OQL in ODMG does not provide an explicit update operators. Instead, it relies on operations defined on objects for that purpose. An OQL query is a function. When it is applied to its input, the query delivers an object the type of which may be inferred from the operator contributing to the query expression. Typically, the result is an object or a collection of objects. According to recent rumors, the OQL of the ODBMS called O_2 [Bancilhon *et al.* 1992] will be the OQL of ODMG Release 2.

3.3 X.500 Directory Object Model

The X.500 Directory can be viewed as a world-wide logical entity that contains all pieces of relevant information. In order to keep the Directory manageable and fault-tolerant, the Directory is distributed and replicated.

The information contained in the Directory is stored in *entries* and is organized in a *Directory Information Tree* (DIT). Collectively, the entries in the Directory are referred to as the *Directory Information Base* (DIB). Each entry in the DIB holds information about a single object. These pieces of information are organized as a set of *attributes*. Each attribute holds a certain piece of information about a single facet of the object. An attribute has the *attribute type* that indicates how the associated *attribute value* is to be interpreted. An attribute may be multi-valued. Each attribute type has an associated *attribute syntax* and *matching rule*. The syntax describes how the information of this type is represented in requests and replies. The matching rule determines how values of this type are to be compared. Typically, an attribute type is specified as a *subtype* (refinement) of some more general type. A subset of each entry's attribute values are distinguished to form the *relative distinguished name* (RDN) which is the unique (with the respect to all siblings of a common parent) name of this entry.

The *class* of a Directory entry is determined through the type of object that it represents. Entries of a particular class must have certain *mandatory attributes*. They may also have other *optional attributes*. Object classes are organized into a hierarchy according to "is-a" relationship. The class hierarchy makes it easy to create new classes that are extensions of existing classes. A special entry class Alias is defined to refer to an object by more than one name. The aliases imply the actual structure of DIT is directed graph.

To summarize, the Directory provides an object-oriented information model. X.500 information model is quite flexible. It permits generic operations which are parametrized by the data carried in the operations. The model is a formal description of the data that is needed to support the services. In addition to the formal description, the information model of X.500 provides a formal organization of the data.

When a *Directory System Agent* (DSA) receives a request from a *Directory User Agent* (DUA) that it cannot complete, the DSA has two options. Firstly, it can communicate with other DSAs on the behalf of the DUA to complete the request. Secondly, it can simply *refer* the DUA to a DSA that it believes will be better able to complete the request. The refer option implies that each DSA must maintain a set of *knowledge references*. Since each DSA contains a part of the DIT, the DSA must know the presentation addresses of those DSAs which contain the immediate parents and children of its entries. The X.500 Recommendations specify the protocols that are used to create, maintain, and extend the knowledge held by each DSA.

In the Directory exactly one DSA is responsible for the creation and maintenance of an entry. This DSA is called the *master DSA* for this piece of information. The Directory allows that other DSAs may have copies of some pieces of information. To simplify the implementation of the DSAs, the updates to the master information need not be synchronized with updates to its copies. As a result, the DIB may, at times, be in an inconsistent state. This is not a serious deficiency, since the users have always the option of requesting access to the master information.

The 1992 Amendments to X.500 present two different mechanisms for replication information: *caching* and *shadowing*. Caching occurs when a DSA stores locally a copy of an entry used to fulfill a request. This may result in a substantial performance improvement if the same entry is rerequested. Shadowing occurs when one DSA mirrors some information held by another DSA. The shadowed information is brought into agreement with the master information on some predetermined schedule.

In addition to attributes that hold information about single facets of an object, there are attributes that hold administrative information, e.g., knowledge references and access control items. Such attributes are known as *operational attributes* that are normally invisible to ordinary users. Operational attributes that apply to more than one entry are stored in a special kind of entry known as a *subentry*. Besides storing administrative information, subentries can also be used to store attributes shared by more one entry. Each subentry has a `subtreespecification` attribute that specifies the set a entries to which the remaining attributes apply.

4. Comparison and Conclusions

In the previous section we summarized three different object models. Of these models the X.500 model is the most mature. However, there are some major shortcomings in X.500. The two most serious ones are the formal organization of data into a directed graph and the lack of atomic transactions. Minor shortcomings include the lack of multiple inheritance, operations, relationships, extents, keys, and transient objects.

The formal organization as a directed graph makes it complicated to use X.500 as the object model of the database system for telecommunication services. As pointed out in the usage scenario in Section 2.1, the queries have different criteria in select operation: the line number and the dialed digits. Moreover, the management needs at least two different views into the data: one in which all services of a single subscriber can be found and another in which all subscribers of a single service can be found.

The ultimate goal in OMG is portability and interoperability. These goals are also very important in telecommunications. However, they are so general that the OMG Core Object Model does not take into account the specific needs of databases. Therefore, our choice is the ODMG Object Model. However, the OMG — particularly the OMG Object Request Broker — will obviously have an important role in future telecommunication systems.

The ODMG-93 is designed to be a general-purpose object database system. Therefore, all features presented in the ODMG-93 Release 1.1 are neither appropriate nor needed in a real-time database system [Raatikainen and Taina 1995] (RT-DBS). An RT-DBS is a transaction processing system that tries to satisfy the explicit timing constraints (deadlines) associated with each incoming transaction. The ultimate goal of an RT-DBS is to maximize the fraction of transactions which meet their deadlines. When the deadline is *firm* (outcome of a late transaction is useless or harmful), then the late transaction (or transaction that can no more meet its deadline) can be aborted without restart. When the deadline is *soft* (outcome of a late transaction gradually loses its value), then the late transaction is continued with a low priority.

One of the primary difficulties in designing an RT-DBS is the fact that the concurrency control of data access must be combined with real-time scheduling. The major design issuesYu *et al.* [1994] in concurrency control are conflict detection and conflict resolution. *Earlist Deadline First* is most widely used in the experimental real-time database systems as the scheduling policy. Alternatives include *Least Slack First*, *fixed priorities*, *criticality of transaction*, and *weighted priority scheduling*. In order to cope with the overload situations, which can significantly decrease the fraction of transactions that complete in time, the scheduling policy needs an overload management policy.

In the traditional DBMSs concurrency control restricts concurrent execution of transactions so that the transactions *serialize*. In real-time applications concurrent transactions do not necessarily need serializability to produce a correct result. In telecommunication there are several database operations that do not need to serialize. However, some transactions need serializability as identified

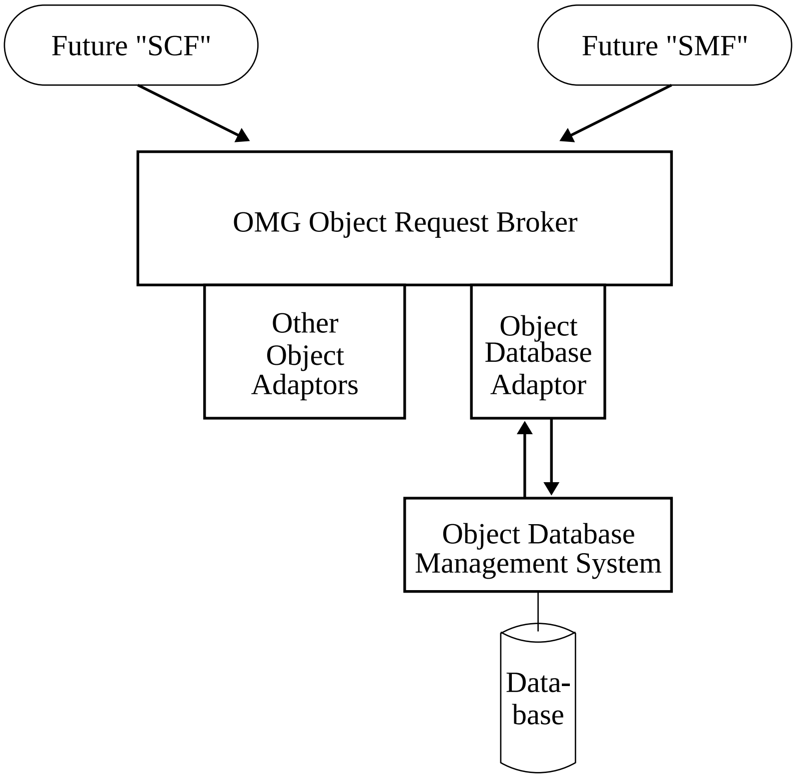

Figure 4: Database service architecture for future IN

in Section 2.2. If the situation is such that some data items need strict serializability while other items do not, the situation can be solved by the explicit use of weak and strong locks. However, if some transactions require strict serializability while some other transactions are satisfied by weaker constraints and these transactions access the same data items, the situation is quite complex.

An interesting correctness criteria, which we call τ-*serializability*, can be used to reduce the number of 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 (but before T_A either commits or aborts) 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 τ-serializability, the two locks do not conflict if $t_a + \tau > t_b$. The tolerance τ (how long the old value is useful) may depend on x (on data semantics) as well as on T_A and T_B (on application semantics).

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 primary advantage in the object orientation is that we obtain an elegant way to incorporate the additional information needed in real-time concurrency control and scheduling into the database. However, we must remember that *the real-time* is an almostly unexplored area in object-oriented database research.

An *"RT-ODMG"* will be a subset of ODMG-93 with predefined types of `RealTimeObject` and `RealTimeTransaction`. Type `RealTimeObject` will include attributes that specify the correctness criteria and resource demands of operations. Type `RealTimeTransaction` will include attributes that may overwrite the correctness criteria of operations and that specify the deadline and criticality for the instances of this type.

Since the ODMG Object Model is a superset of the OMG Object Model, the OMG ORB (CORBA) is one possible way to invoke database services. Fig. 4 outlines how future SCFs and SMFs may obtain database services through the OMG ORB. When we correlate the service provision architecture in Fig. 4 with the IN functional architecture in Fig. 2, we see that the functionality of communication and the Functional Entity Access Managers are provided by the OMG ORB and the Object Adaptors. We believe that Service Data Object Directory is not needed in the SCF. Instead, *Service Data Object Cache* would significantly improve the performance.

Acknowledgements

The authors are thankful to the rest of the Darfin team (Research project *Database ARchitecture For Intelligent Networks* funded by Telecom Finland), particularly to Pekka Olli, Mika Rautila, and Harri Töhönen.

References

BANCILHON, F., DELOBEL, C., AND KANELLAKIS, P., EDITORS. 1992. *Building an Object-Oriented Database System: The Story of O_2*. Morgan Kaufmann, San Mateo, Calif.

CATTELL, R. G. G., EDITOR. 1994. *The Object Databse Standard: ODMG-93*, revision 1.1 edition. Morgan Kauffmann, San Francisco, Calif.

OMG. 1992a. *Object Management Architecture Guide*. Number Number 91.11.1. Revision 2.0 in OMG TC Document. John Wiley & Sons, New York, N.Y.

OMG. 1992b. *The Common Object Request Broker: Architecture and Specification*. Number Number 91.12.1. Revision 1.1 in OMG Document. John Wiley & Sons, New York, N.Y.

RAATIKAINEN, K. E. E. 1994a. Database Access in Intelligent Networks. In *Proceedings of IFIP TC6 Workshop on Intelligent Networks*. Lappeenranta University of Technology, Lappeenranta, Finland, 163–183.

RAATIKAINEN, K. E. E. 1994b. 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. AND TAINA, J. 1995. Design Issues in Database Systems for Telecommunication Services. Report C-1995-16, University of Helsinki, Dept. of Computer Science, Helsinki, Finland.

YU, P. S., WU, K.-L., LIN, K.-J., AND SON, S. H. 1994. On Real-Time Databases: Concurrency Control and Scheduling. *Proceedings of the IEEE 82*, 1 (Jan.), 140–157.