1994

An Evaluation of the Open Object-Oriented Database

Frank Alsop
University of Rhode Island

AN EVALUATION OF THE OPEN OBJECT-ORIENTED DATABASE

BY

FRANK ALSOP

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

COMPUTER SCIENCE

UNIVERSITY OF RHODE ISLAND

1994

MASTER OF SCIENCE THESIS

OF

FRANK ALSOP

APPROVED:

Thesis Committee

Major Professor Victor Fay-Wolfe

Dilip K. Datta

Joan Peckham

Kent Mann

DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND

1994

ABSTRACT

The diversity of database designs has motivated standardization efforts. The Advanced Research Projects Agency (ARPA) sponsored Open Object-Oriented Database (Open OODB) project is the leading effort to develop an extensible, modular architecture for next-generation, object-oriented databases (OODBs). The Next-Generation Computer Resources (NGCR) Database Management Systems Interface Standards Working Group (DISWG) has published requirements that will help dictate how the Navy meets its future database needs. This thesis will evaluate Open OODB with respect to the DISWG requirements. Each DISWG requirement is evaluated from three distinct perspectives. First, we see if Open OODB has a matching requirement(s). Second, we evaluate if Open OODB's proposed architecture would meet the requirement, and third, we evaluate if Open OODB's implementation meets the requirement. If a particular DISWG requirement is found to be unmet, we investigate the feasibility of extending Open OODB to meet the requirement.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Victor Faye-Wolfe, for inspiring this thesis and for his help in seeing it through. I would also like to thank Dr. Joan Peckham for her answers to my many questions. Thanks, as well, to my committee chairperson, Dr. Sodhi, and my other committee member, Dr. Datta.

I would like to acknowledge all the members of the RTSORAC research group, including Paul Fourtier, Janet Prichard, Lisa Cingiser-DiPippo, Gary Hyslop and John Black for their support and input.

Finally, I thank my wife, Melissa, for all her support and patience.

PREFACE

This thesis provides an evaluation of the Open Object-Oriented Database (Open OODB) project with respect to the requirements of the Next-Generation Computer Resources (NGCR) Database Management Systems Interface Standards Working Group (DISWG). DISWG lists one hundred and nine requirements. Open OODB is evaluated with respect to each requirement on three levels: Open OODB's requirements, its proposed architecture and its implementation. The thesis is organized as follows.

Section one provides an introduction. We first discuss the shortcomings of current databases. We then motivate the importance of standardization to overcome those shortcomings. We then introduce Open OODB and DISWG and highlight their importance.

Section two provides a summary of Open OODB. Since the rest of this thesis is predicated upon an understanding of Open OODB, we go into some detail. We discuss Open OODB's requirements, its computational model and its architecture. We also briefly look at the CORBA model for a next-generation OODB because Open OODB and CORBA are closely related.

Section three provides a look at relevant research at the University of Rhode Island (URI). This includes a look at the RTSORAC (Real-Time Semantic Objects Relationships and Constraints) project which is a leading effort to develop a next-generation, real-time database. We then provide a brief look at the important question of where the operating system ends and where the DBMS begins. We conclude with a look at some of the author's individual research.

Section four contains the evaluation of Open OODB with respect to DISWG's requirements. DISWG partitions its requirements into nine requirements classes. Thus, this section is partitioned into nine subsections, one for each DISWG requirements class.

The first subsection contains an evaluation of Open OODB with respect to the DISWG General requirements class. The General requirements pertain to almost any computer system, not just databases. These General requirements include topics such as portability, modularity and scalability, etc.

The second subsection contains an evaluation of Open OODB with respect to DISWG's Basic DBMS requirements class. These Basic requirements pertain to DBMSs in general. Included are topics such as support for queries, transactions and persistent data, etc.

The third subsection evaluates Open OODB with respect to DISWG's Distribution requirements class. In this subsection, it is observed that Open OODB does not qualify as a distributed database system as per DISWG's definition. Open OODB is, however, a distributed database. Therefore, Open OODB is evaluated as a distributed database and we point out how Open OODB could be extended to be a distributed database system.

The fourth subsection evaluates Open OODB with respect to DISWG's Heterogeneity requirements class. These requirements address the interoperability of heterogeneous databases and DBMSs. Thus, this requirement does not really apply to Open OODB since Open OODB is a DBMS in and of itself. Therefore, we discuss work underway to allow Open OODB to be incorporated into a heterogeneous system.

The fifth subsection evaluates Open OODB with respect to DISWG's Real-Time requirements class. Open OODB is not a real-time database so these requirements are not met. Many of these requirements depend upon a real-time operating system. In this evaluation, we reference work being carried out at URI to extend Open OODB with the functionalities of URI's RTSORAC model and to port Open OODB to a real-time operating system.

The sixth subsection evaluates Open OODB with respect to DISWG's Fault Tolerance requirements class. Many of these requirements are reliant upon operating

system support. Currently, Open OODB defers recovery to the underlying Exodus storage manager. Therefore, most of the discussions in this class focus on how to extend Open OODB to meet the requirements.

The seventh subsection evaluates Open OODB with respect to DISWG's Integrity requirements class. Most of these requirements are not met by Open OODB. We once again discuss extending Open OODB with RTSORAC features in order to meet these requirements.

The eighth subsection evaluates Open OODB with respect to DISWG's Security requirements class. Security as a class has been de-emphasized by DISWG and security is outside the current scope of Open OODB. Therefore, this evaluation is different than the evaluations of DISWG's other eight classes. We discuss security as it relates to DBMSs and OODBs in general and do not perform a separate evaluation of the twenty-four requirements of this class.

The ninth subsection evaluates Open OODB with respect to DISWG's Advanced DBMS requirements class. This class is where DISWG incorporates requirements on OODBs specifically. Therefore, many of these requirements are met and we discuss how Open OODB meets them.

In the last section, the conclusion, we first summarize the main findings in the evaluations of each DISWG requirements class. Finally, we present some general observations on the strengths and weaknesses of Open OODB.

Contents

List of Figures

1 Introduction

Most large, important computer applications rely on databases to efficiently store and retrieve information. Scientific, medical and business applications all need vast stores of data in order to perform essential computations. For example, NASA estimates that just a few years of satellite image data will require storage for around 10^{16} bytes. This large figure does not even account for all the important, pre-existing data that must be maintained [84]. However, as the twenty-first century quickly approaches, the shortcomings of current database technology are well recognized [82]. Thus, new database technologies known as *next-generation* database systems are being proposed.

The new technology receiving the most attention is the object-oriented database (OODB). Many think that OODBs represent the future of database technology [39, 71, 93]. The object-oriented (OO) approach represents a new way of modeling the world in a computer environment. Rather than looking at data and the functions that manipulate data as separate entities, which is the traditional approach, the OO paradigm looks at data and its associated functions as a single entity, or object. For example, since it is natural to think of a sphere together with its volume, radius, and other attributes, every sphere object comes with functions that can compute its volume and radius, etc. Thus, a major benefit of using the OO approach is that it more closely models real world situations than do the previous approaches [19]. Since databases typically involve modeling some real word entity (a customer, a patient, a weather pattern, etc.), applying the OO paradigm to database technology seems natural.

As when almost any new technology surfaces, there are many, disjointed efforts underway to develop OODBs. This leads to the undesirable consequence of each endeavor having to "rediscover the wheel". A coordinated, unified attack on the problem of designing an OODB would have many benefits. One is that work which has already been done can be reused and improved upon without having to start

"from scratch". Also, different groups of researchers could work towards solutions of the *same* problems and share their experiences.

Central in developing a coordinated, unified attack is the establishment of standards. In order to develop standards, there must be in place an agreed upon set of requirements. A *requirement* is a desired quality that, in this case, every database should have. An example is extensibility which means that new functionalities can easily be added. A *standard* is much more implementation specific. For example, a standard might define exactly what values must be passed into a specific function. To develop standards, preliminary work must be done and proposed designs, or architectures, tested. In the field of databases, there are several such unifying efforts underway.

The leading effort to develop design proposals for future OODBs is the Open Object-Oriented Database (Open OODB) project ongoing at Texas Instruments (TI). This effort is sponsored by the Advanced Research Projects Agency (ARPA). Open OODB's overall goal is "to build a high-performance, multi-user object-oriented database management system (OODBMS) in which database functionality can be tailored by application developers for the diverse needs of demanding applications"[58]. Open OODB has an overall set of requirements describing the needs of next-generation OODBs [69] and has proposed a database design, or framework, that other developers can use to build customized OODBs. Additionally, Open OODB has implemented some of its proposed design. This allows Open OODB to be evaluated from three distinct perspectives: its overall requirements, its proposed design and its current implementation.

The United States Navy has formed the Next-Generation Computer Resources (NGCR) task force to standardize its computer systems. The Navy has a multitude of heterogeneous computer systems at sea, on land, in air and in outer space. It would obviously be helpful if these systems adhered to a common set of standards that would

allow them to communicate and share data efficiently. To quote the Chief of Naval Operations, "the force that wins the information battle will gain such ascendancy that it may not have to fight the real battle"[32].

One subgroup of the NGCR has the specific task of developing requirements that the Navy feels next-generation databases must meet. This group is the Database Management System Interface Standards Working Group (DISWG). Notice that the acronym does not refer directly to databases, but to database *interfaces*. An interface is a channel of communication, somewhat like a door into a room. Thus, DISWG is interested in establishing requirements on the way different parts of a database management system talk to each other and to the rest of the system.

The University of Rhode Island (URI) is an alpha site responsible for testing and issuing reports on Open OODB.[1] Researchers at URI are interested in extending the functionalities of Open OODB to meet more of the DISWG requirements. To guide this research, an evaluation of Open OODB must be performed. This evaluation must include the following.

- Which DISWG requirements are currently met by Open OODB. This part of the evaluation is performed from three distinct perspectives: Open OODB's requirements, its implementation and its proposed architecture.
- An examination of unmet DISWG requirements and a report on the viability of extending Open OODB to meet those requirements. Suggestions must also be made on how to implement those extensions.

Thus, this thesis performs the essential task of evaluating Open OODB with respect to DISWG's requirements. It also provides guidelines to help extend Open OODB to meet more of DISWG's requirements. The rest of this thesis is organized

[1]The alpha version of Open OODB is release 0.2. When we refer to Open OODB or the current Open OODB, we mean this release.

as follows. The next section reviews Open OODB's requirements and proposed system architecture. The following section contains the evaluations in nine subsections, one for each DISWG requirements class. Each DISWG requirements class is made up of a set of individual requirements. Open OODB is evaluated with respect to each, individual DISWG requirement from the aforementioned three perspectives: its requirements, its proposed architecture and its implementation. Any discussion of extending Open OODB to meet a particular requirement is included in the implementation evaluation. The thesis ends with some concluding remarks.

2 Open OODB

The diversity of OODB designs has slowed acceptance by potential users [94]. Each design must start from scratch and build from the bottom up. Development of reusable, standardized components would facilitate matters. Open OODB represents the leading effort to design "an architectural framework that allows flexible configuration of independently developed modules ..."[58]. Open OODB also wishes to verify its proposed architecture through implementation and testing.

Figure 1 pictorially represents how Open OODB's requirements and architecture relate to each other.[2] In Figure 1, we see that Open OODB has two sets of requirements: meta and functional requirements. The meta requirements are requirements on the meta architecture and the functional requirements are requirements on the extenders. The meta and functional requirements, as well as the extenders, are all described in detail in upcoming sections. Together, the meta architecture and the extenders make up the system architecture.

We begin this section with a motivation for the development of OODBs in general. Then, we provide an overview of the Open OODB project highlighting its

[2]This diagram also appears in a slightly different form in Open OODB's literature. The diagram included here is from the Technical Overview document included with release 0.2. Elsewhere, the system architecture and extenders are merged together with an arrow going from the meta architecture to the system architecture. That arrow is labeled *used to implement*.

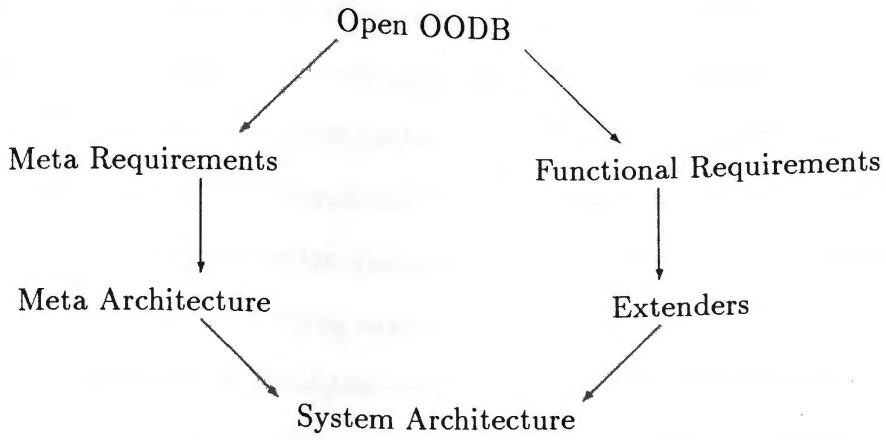

Figure 1: The Open OODB Requirements and Architecture

requirements, computational model and proposed architecture. Throughout these discussions, implementation details are brought up where relevant. Many more implementation details are left for the sections evaluating Open OODB with respect to DISWG's requirements. To conclude the overview of Open OODB, we take a brief look at an influential group working towards OODB standards to whom Open OODB is closely tied: the Object Management Group (OMG).

2.1 Motivation for OODBs

Research in OODBs has been motivated by the inability of traditional databases to deal with next-generation database needs. Indeed, some would say that traditional, existing databases do not adequately fulfill the needs (i.e., business applications) for which they were specifically designed [82]. Next-generation needs include computer aided design, office information systems and a host of multimedia applications. The three most widely accepted database models so far have been the relational, hierarchical and network models. However, all these models have limitations, which we next look at briefly.

Traditional relational systems are limited by a small set of data types and the functionality of those types [49]. Thus, relational models have trouble representing the

complex data types needed for multimedia applications, etc. Also, in the relational model, complex relationships between objects are hard to express [23]. Relational systems are inhibited by decreasing performance as their size grows [26].[3] Thus, in [23], it is concluded that the "simplicity of the relational model is both its strength and its weakness." Next-generation databases will have to be strong performers with a rich data set to enhance modeling capabilities.

Network and hierarchical databases both provide so-called *record-at-a-time* access. Performing queries in such systems is not natural for the unskilled. Complex programs must be written in order to access data, which limits short-notice data availability [84]. If the structure of the database were to change, chances are the application programs that access data would also have to change. These factors tend to place a great burden on the programmer [26, 84]. Also, performance is a concern in the network and hierarchical models because of the expense of crossing the application/database boundary many times [42].[4]

The OO model has been proposed as a viable alternative to the traditional data models. OODBs have the potential to more realistically model real world situations. There are many efforts underway aimed at developing both OODBs and relational databases extended to include OO features. These efforts include[5]: Orion [43], O_2 [22], Iris [95], GemStone [13], ObjectStore [48], POSTGRES [83] and Starburst [49]. However, because there is no coordination among these efforts and no accepted definition of an OODB, each project must start from scratch [5, 59]. Open OODB is the ARPA sponsored project at TI to develop an open, extensible architecture for the use of future OODB developers.

[3]This is not to imply that performance is not a concern in OODBs. In fact, it is a concern [15]. However, the performance problems of relational systems are much more understood.

[4]Many would say that the network/hierarchical models outperform the relational model. However, there are concerns about how performance in all those models will scale to next-generation needs.

[5]This is only a partial list; there are certainly other worthwhile efforts. Also, this list includes both OODBs and extended relational databases because there is disagreement on where the dividing line between the two falls.

2.2 Open OODB's Requirements

Requirements are needs that must be met to satisfy some customer(s). Requirements need to be precise without including any implementation details [69]. The Open OODB project defines three types of requirements: meta requirements, functional requirements and next-generation OODB requirements. The meta requirements describe "the organizational and operational characteristics that the OODB must meet while satisfying the functional requirements"[59]. The functional requirements describe "the capabilities the Open OODB must provide to its users"[66]. The next-generation OODB requirements are a "catch-all" for features that are out of the scope of Open OODB. The Open OODB requirements are presented as a list numbered from R1 through R17. R1 through R12 are the functional requirements and R13 through R16 are the meta requirements. R17 is the lone next-generation OODB requirement. Under each requirement, there are numerous subrequirements which are labeled as R1-2 or R4-5-2, etc. We now review each type of Open OODB requirements.[6]

2.2.1 Open OODB's Meta Requirements

As noted earlier, meta requirements are overall characteristics that must be adhered to while still providing the needed services, or functionalities. Referring to Figure 1, we can see that the meta requirements are requirements on the meta architecture. The meta architecture is discussed in the architecture section. Open OODB has four meta requirements: R13 Openness, R14 Seamlessness, R15 Performance and R16 Industrial Strength. R13 Openness and R14 Seamlessness are referred to as organizational meta requirements while R15 Performance and R16 Industrial Strength are called operational meta requirements. None of these four are specific to OODBs, which is why they have been separated from the rest of the requirements [64]. Next, we summarize each meta requirement.

[6]The complete list of Open OODB requirements is too lengthy for inclusion. For the complete list, see [69].

R13 Openness. This is perhaps Open OODB's key meta requirement. Open is defined as meaning "that the modules of the OODB have well-documented interfaces and that the modules can be changed or replaced to add functionality or improve performance"[69]. Openness is the "characteristic of a system which allows developers or researches to modify or control some part(s) of its architecture or implementation"[60]. The name *Open* OODB was not chosen arbitrarily! All of Openness's subrequirements, R13-1 through R13-11, deal with some aspect of modularity. Examples include: modules need well defined interfaces, modules need to be justified and modules may be extensible, etc. Its worth noting that Open OODB uses the terms openness and extensibility as synonyms [59].

R14 Seamlessness. An "OODB must provide a seamless interface between the programming language and the database"[69]. A seam is an "explicit interface which occurs when performing an action" and " the occurrence of an event involving a specific operation, argument type, or environmental attribute"[60]. Seamlessness refers to the "the absence of a seam between the data model of a transient application and a persistent shared database ..."[60]. Seamlessness can be considered a type of transparency, where operations can be carried out on behalf of a user, but without the knowledge of the user. Subrequirements here deal with different types of transparency. For example, R14-3 states that an OODB must exhibit persistence transparency, R14-6 location transparency and R14-7 replication transparency.

R15 Performance. "An OODB must provide usable performance and the ability to configure, tune, measure, profile performance"[69]. There are five subrequirements of this meta requirement. R15-1 lists a set of required measures such as speed and response time, etc. R15-2 requires an OODB to support performance tuning mechanisms. R15-3 states that an OODB should support usage metering. R15-4 requires support for performance benchmarks and R15-5 states that an OODB must be scal-

able.

R16 Industrial Strength. This group of meta requirements applies to any usable system. Thus, they are "inherited by any OODB"[69]. Here, we find requirements on documentation, portability, robustness and dependencies, etc.

2.2.2 Open OODB's Functional Requirements

Functional requirements are services provided by an implementation that do not preclude the implementation from meeting the meta requirements. The Open OODB requirements document lists twelve functional requirements. First, we list those requirements and then we briefly discuss them.

- R1 Object-oriented Data Model.
- R2 Persistence.
- R3 Concurrent Access.
- R4 Distribution.
- R5 Data Dictionary.
- R6 Query Capability.
- R7 Change Management Facility.
- R8 Class Libraries.
- R9 Integrity and Recovery.
- R10 Security.
- R11 Access to Legacy Data.
- R12 Program and User Interfaces.

R1 Object-oriented Data Model. "An OODB must have an object-oriented data model"[69]. The OO data model is briefly discussed in the introduction to this thesis. Subrequirements of this functional requirement include many of the characteristics of OO programming languages such as object identifiers (OIDs), inheritance and complex objects, etc. OIDs provide a way to uniquely identify objects. Inheritance allows objects to acquire the properties of other objects and complex objects are made up of more than one object.

R2 Persistence. "An OODB must support persistent storage of object instances and classes supported by its OO data model(s)"[69]. Persistence is the ability of data to outlive its creating process. "Persistence assumes that there is a pool of objects outside program scope that can be brought into a program and put back outside program scope when desired"[65]. Example subrequirements include: R2-3 an OODB must be able to store and retrieve objects, R2-4 an OODB must be able to map an object into the proper format (i.e., transient or persistent) and R2-5 an OODB must support object names independent of program names.

R3 Concurrent Access. "An OODB must provide for sharing and controlled concurrent access by multiple users/processes"[69]. Concurrent access is "the access of an object by two or more processes at the same time"[60]. Sample subrequirements of this functional requirement deal with lock primitives and transaction control, etc. Lock primitives provide a means to synchronize concurrent access and transaction control maintains database consistency while allowing concurrent access.

R4 Distribution. "An OODB must support location transparency of objects"[69]. Location transparency means that constructs "can be applied independently of an object's current physical or logical location" and that "details of the mapping of a fragment into a particular object store" are hidden [60]. Two subrequirements

included here are that an OODB must support object transfer between workstations and that applications can access data stored on multiple object servers.

R5 Data Dictionary. "An OODB must be able to store, access, and manipulate meta-data"[69]. Meta data[7] "encodes the definition (structure and behavior) of a class (type, schema) or other auxiliary information about a class." A data dictionary is a collection of meta data. Subrequirements of this functional requirement mainly list the types of information that a data dictionary should be able to represent.

R6 Query Capability. "An OODB must support a query capability"[69]. A query is a "statement written in a non-procedural language specifying *what* data is to be retrieved from a database"[69] without specifying how it is to be retrieved. Sample subrequirements include: R6-2 an OODB to be SQL compatible, R6-5 that the results of queries can be queried and R6-7 that queries must be efficient. SQL (Structured Query Language) is the best known database query language.

R7 Change Management Facility. "An OODB must support a change management facility"[69]. Open OODB defines change management as a "consistent set of techniques that aid in evolution, composition and policy management of the design and implementation of an object or system"[69]. Thus, the change management subrequirements deal with such topics as versions, configurations and schema evolution. A version is a variant of an object's original value and a configuration is a group of consistent, related versions. Schema evolution means that the database is likely to change, or evolve.

R8 Class Libraries. "An OODB should provide support for class libraries"[69]. This functional requirement says that it must be possible to persist objects from class

[7]Meta data is alternatively spelled meta-data and metadata in Open OODB documentation. However, to be consistent with "meta requirement", we prefer "meta data".

libraries and that some pre-existing, or legacy libraries must be supported. An obvious example is the libraries that are associated with C++, such as < streamio.h >, etc.

R9 Integrity and Recovery. R9 requires that "An OODB must support data integrity and recovery"[69]. Under integrity, there are subrequirements stating that pointers and indices must be consistent and that it must be possible to state various integrity constraints. Under recovery, there are subrequirements for atomic transactions, recovery and backup/restore.

R10 Security. "An OODB must support security"[69]. These requirements state some obvious security concerns such as authorization, grant operations and encryption. However, since security is outside the scope of the current effort, this requirements section is undeveloped and preliminary.

R11 Access to Legacy Data. "An OODB must support access to legacy data stored in an SQL-compatible relational database or file system"[69]. This requirement is outside the scope of the current effort, so like security above, it is not well developed.

R12 Program and User Interfaces. "An OODB should support program and user interfaces"[69]. Interfaces are essential to the success of a project like Open OODB. The application program interface subrequirements list the different types of support that an OODB must provide to be accessible to programs. The user interface subrequirements cover many things, but, like R11 Access to Legacy Data above, are outside the scope of the current effort.

2.2.3 Open OODB's Next-Generation OODB Requirements

This requirement type is a "catch-all" for next-generation features that are outside the scope of the Open OODB project. Under this type, there is only one requirement, R17, which states that "these database application requirements are out of

the scope for OODB1 (first release of Open OODB) but the design should not preclude them"[69]. There are only ten subrequirements, including: R17-1 main memory DBMS, R17-2 heterogeneous DBMS, R17-4 rules-based DBMS and R17-8 parallelism. These subrequirements are not reflected in Open OODB's proposed design or implementation, are undeveloped and are not mentioned anywhere else in Open OODB's literature.

2.3 Open OODB's System Architecture

Open OODB's system architecture[8] is partitioned into two pieces: (1) a **meta architecture** and (2) **extenders** [66].[9] "This ... corresponds to the partitioning of the Open OODB requirements into meta and functional requirements"[59]. The meta architecture provides a basis for specifying and implementing extensions and governs module interfaces. The extenders define the actual modules that represent Open OODB's functionalities. Refer to Figure 2 to see how the meta architecture and extenders fit into the overall system architecture. In the rest of this subsection, we first review the Open OODB computational model, then its meta architecture and, finally, its extenders.

It is important to note that this discussion of Open OODB focuses on the *proposed* architecture. Much of the functionalities are, as of yet, unimplemented. This distinction becomes more apparent as we proceed.

2.3.1 Open OODB's Computational Model

"Open OODB's computational model is based on transparently extending the behavior of objects from application programming languages"[59]. To accomplish this,

[8]When we refer to Open OODB's architecture, or current architecture, we mean the architecture as described in documents specifically provided with the alpha release. Since Open OODB is an ongoing project, its architecture has been changed since the alpha release.

[9]In a later draft, the *architecture* is defined as consisting of the *meta architecture* and the *system architecture* In this later definition, the system architecture does essentially what the extenders do as described in the alpha release.

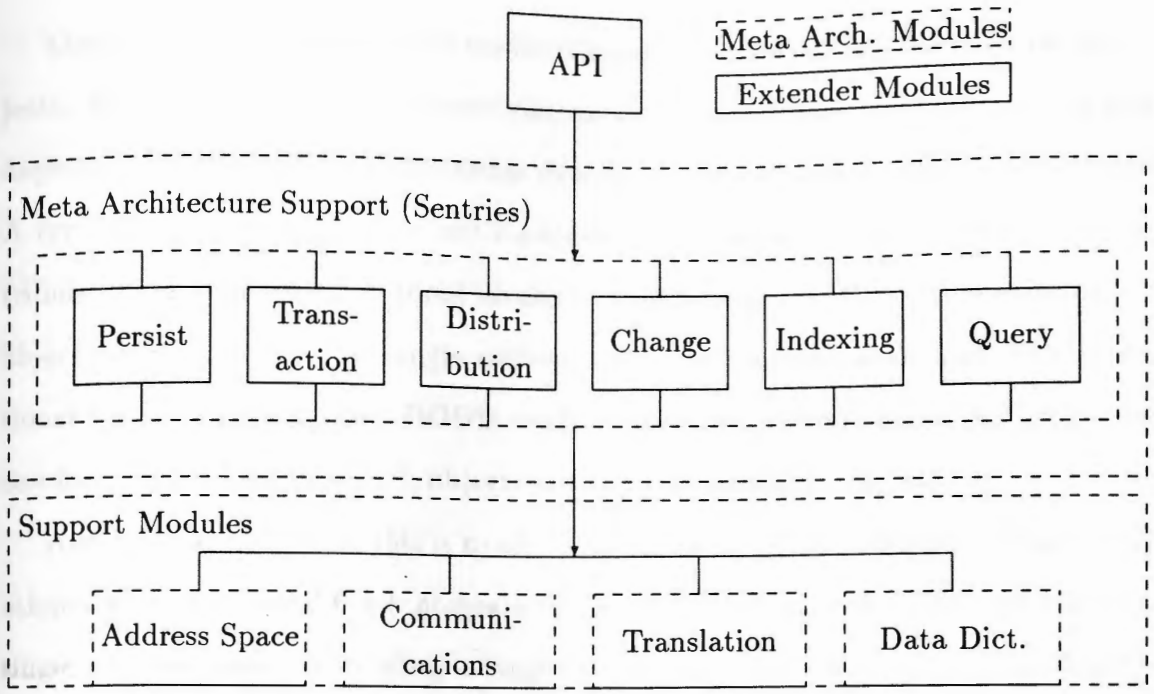

Figure 2: The Open OODB Architecture

Open OODB wishes to avoid making programmers use embedded system calls. Such calls can be viewed as seams, something which violates Open OODB's Seamlessness meta requirement. Thus, the computational model is based on transparently extending C++.

Open OODB is based upon extending C++. An *extension* of a programming language provides additional functionalities to that language. For instance, a useful extension when considering databases is persistence. Traditional C++ programs provide no mechanisms to manage persistent objects, but such mechanisms are crucial to databases. Thus, Open OODB extends C++ to include persistence.

In Open OODB's computational model, all objects exist in a universe of objects. In traditional C++ programs, all objects exist in the same universe. Instead of using the word universe, we could say that all objects in traditional C++ programs exist in the same environment. Therefore, in traditional C++ programming, there is one universe, or environment, in which objects can reside.

Open OODB's computational model is based upon redefining the universe of objects. In Open OODB, the universe can consist of more than one environment and objects have *environmental attributes* describing the object's current environment. A typical environment is an object's address space and a typical environmental attribute contains information about an object's address space. For instance, instead of objects existing only in the single address space environment associated with traditional C++ programs, Open OODB needs at least two address space environments: one for "regular", or transient, objects and one for persistent objects.

Another way to look at this is to say that the cardinality of all the environmental attributes in traditional C++ programs is one. For example, all objects exist in one, single address space. Extending a language increases the cardinality of an object's environmental attributes. For instance, in C++ extended to include persistence, the address space environmental attribute has a cardinality of at least two: the object may be in a transient address space or a persistent one. In essence, the universe of where an object can exist is partitioned into different environments.

When an Open OODB application wishes to persist an object, that object must be "moved" from one environment to another. In order to move, the object must cross the environmental boundary between the regular address space and the persistent address space. The crossing of an environmental boundary represents an extension to C++. Thus, persistence is one of the extensions to C++ that Open OODB implements. Other extensions are performed in the same way. For instance, C++ makes no provision for replication of objects, so Open OODB creates a replication environment. If an object needs to be replicated, it crosses the environmental boundary between non-replicated and replicated objects. All of the extensions are discussed in the upcoming subsection on Open OODB's extenders.

Interactions with objects in Open OODB are called events and events in a traditional C++ program are known as *direct* events. For example, a direct event could

be a simple assignment statement like *Student[1] = new Person();*[10] where *Student[1]* is a variable local to its creating program. If, however, a direct event needs to be extended, it is replaced by a *virtual event*. Thus, a virtual event is an extension to a direct event.[11] In order to extend an event, a mechanism is needed to detect when an event needs extension. In other words, the mechanism must detect when an environmental boundary is to be crossed. In Open OODB, that mechanism is known as a *sentry*. For example, if there is a call to persist an object or class, etc., in an Open OODB application, that call represents an attempt to cross an environmental boundary. A sentry detects the call, and replaces the direct event with a virtual event that can handle persistence. For example, if the program contains a statement like *Persist(Student[1])*, that statement represents an extension to C++. Therefore, a sentry detects the extension and initiates a virtual event.

When a sentry detects the need to extend an event, the sentry passes control to a module designed to perform the needed functionalities. For instance, there are modules to perform persistence and replication, etc. Open OODB calls the different functionalities *policies* and the modules that implement them *policy performers*. Each policy may have more than one performer associated with it. For instance, persistence may be performed in different ways. We may be persisting a single object or a whole set of objects. The mechanism that decides the actual policy to implement is a *Policy Manager* (PM). Therefore, in Open OODB, there could be a Persistence PM to handle persistence and a Replication PM to handle replication, etc. In our example of *Persist(Student[1])*, the sentry would pass control to the Persistence PM which would initiate the required tasks.

To sum up this discussion of Open OODB's computational model, we carry

[10]The statements included are in *pseudo-code*. That is, they are for illustrative purposes only and do not represent any particular language.

[11]Note that virtual events themselves can be extended. This situation could arise if we need to persist a replicated object. Before the persistence virtual event could finish, it would have to be extended with a replication virtual event.

through an example other than persistence. An Open OODB application is executing as a sequence of direct events when a statement calling for the beginning of a transaction is reached. Transactions represent an extension to C++. The need to extend the direct event is trapped by a sentry. The sentry calls the Transaction PM to replace the direct event with a virtual event. The Transaction PM picks an appropriate policy performer to implement the extension, or virtual event. Different performers here may be one to implement two-phase locking or one to implement optimistic concurrency control, etc. When the transaction is complete, control is returned to the original, direct event.

2.3.2 Open OODB's Meta Architecture

"The meta architecture consists of the modules, interfaces, and topology necessary to support the computational model of event extension"[66]. Its purpose is to "support the kinds of variability evidenced in existing systems or envisioned for future ones"[64]. The meta architecture consists of five modules. These five modules are not directly accessible to applications, making them distinct from the extender modules that are discussed in the next subsection. First we list the five modules and then briefly describe them.

- Address Space Managers.
- Communications.
- Translation.
- Data Dictionary.
- Meta Architecture Support.

Address Space Managers. This module allows for objects to be uniformly accessed no matter where they reside. The objects could be in virtual memory or in a

persistent store, etc. To do so, the Address Space Manager (ASM) supports global identifiers and mappings to and from global identifiers to local representations. In an actual implementation, there would probably be multiple ASMs, one for each address space.

Communications. "Open OODB's Communications module is a veneer that normalizes the interfaces to one or more underlying communications mechanisms"[59]. In other words, "when an object moves, it moves via one or more communications media"[65]. The various communications media may use different transport mechanisms. These transport mechanisms may include network protocols and different memory, etc. The object Communications module picks the appropriate transport mechanism, provides uniform interfaces to the transport mechanisms and transfers the bytes.

Translation. Since different address spaces may require different formats, a module is needed to perform the necessary translations. The Translation module converts objects (or whatever the unit of transfer is) into the appropriate format. Note that the Translation module performs the translation, but the actual byte transfer is performed the Communications module.

Data Dictionary. "The Data Dictionary is a globally known repository of data model and type information, instance information, name mappings (of application-specific names to instances), and possibly system configuration and resource utilization information"[59].

Meta Architecture Support. "The Meta Architecture Support module implements the mechanisms to extend events uniformly and defines interface conventions used by other Open OODB modules"[94]. Referring to Figure 2, we can see that to get to the actual extenders, we must first pass through the meta architecture support

module, which has five principle components. The five are: 1) sentries, 2) common interfaces for Policy Managers, 3) pragmas, 4) common exception mechanisms and 5) common references and OIDs.

Sentries, as noted on page 16, detect and trap events. Common interfaces for Policy Managers are achieved by using a common ancestor PM object and inheritance. Pragmas allow flexibility, i.e.: prefetch if it is possible, but do not raise an exception if it is not possible. Common exception mechanisms "define a collection of exception classes and required handlers to provide traceability for exceptions signaled by hidden policies"[59]. Finally, OIDs and references are objects manipulated by all modules, so they need a global representation.

2.3.3 Open OODB's Extenders

Open OODB's behavioral extensions are represented as a group of extenders, or *Policy Manager* modules [64]. The extenders are defined by the functional requirements [65]. When a sentry detects a virtual event, or extension, the sentry passes control to an appropriate PM. Note that there is not one universal PM to handle, for example, transactions. Different applications may have different needs. Therefore, the PMs could be implemented as a type lattice as depicted in Figure 3.

Refer to Figure 3. At the root of the PM type lattice is a generic PM. All other PMs inherit from this root. In this way, all PMs in Open OODB can present a uniform interface. Under the Transaction PM node, there are different subtypes of Transaction PMs to handle the different types of transactions that may be encountered. If the transaction is nested, the Nested Transaction PM is called. If the transaction is cooperative, then the Cooperative Transaction PM is called. Finally, if the transaction is both nested and cooperative, the Nested Cooperative Transaction PM is called.

The following are the six PM modules currently proposed for Open OODB to implement a subset of Open OODB's overall, functional requirements. After the list

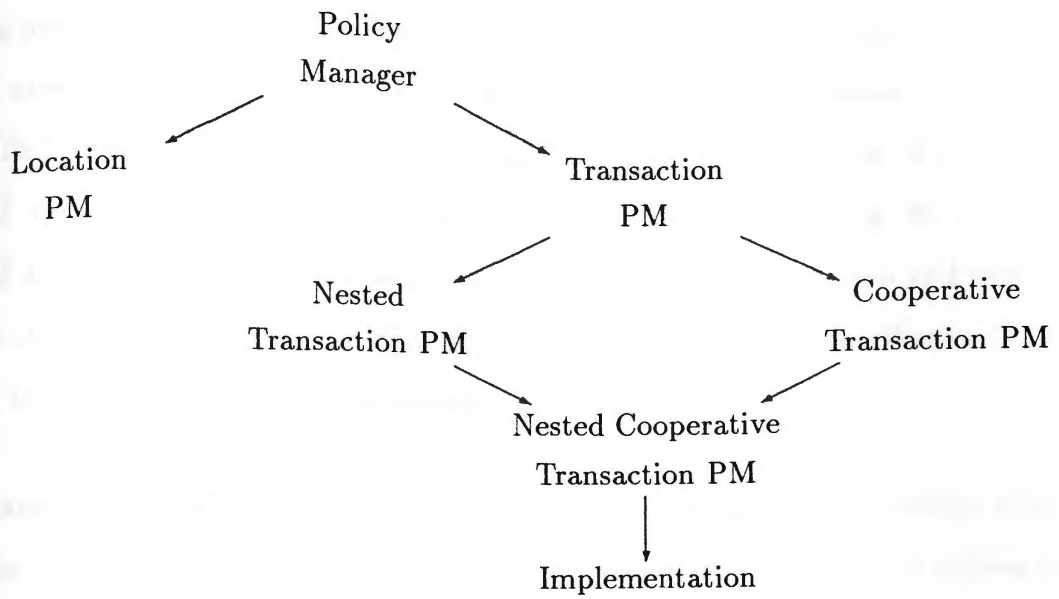

Figure 3: A Policy Manager Type Lattice

is a brief discussion of each of the six.

1. Persistence.
2. Transaction.
3. Distribution.
4. Change Management.
5. Indexing.
6. Query.

Persistence PM. "The Persistence PM provides applications with an interface through which they can create, access, and manipulate persistent objects"[59]. In order to do so, the Persistence PM must be able to allocate and resolve identifiers that uniquely identify objects, or OIDs. These OIDs must be universally interpretable and independent of the object's location. Since OIDs are semantically meaningless to humans, the Persistence PM needs the ability to associate names with objects and OIDs.

For instance, the OID for an object may be 3257, but its name may be "customer". To associate names with objects, the Persistence PM resolves names to OIDs. Since OIDs themselves resolve to objects, names always resolve to objects. The Persistence PM module may use the services of many other modules including the Distribution PM and all of the meta architecture modules. Indeed, "Distribution and persistence are closely related in Open OODB, with persistence simply the distribution of objects ... to address spaces that themselves happen to persist"[59].

Transaction PM. This module "enables *concurrent* access to persistent and transient data ..." [59]. Traditionally, transactions have been designed to adhere to the ACID[12] properties. However, to support a wide range of applications, the traditional ACID properties have to be relaxed. For instance, we may need cooperative transactions, necessitating that the isolation property be relaxed. Therefore, Open OODB allows for more than one type of PM to handle a given extension.

The Transaction PM must "provide the basic operations of begin-transaction, get-object, commit, and abort"[65]. The Transaction PM must also provide and maintain locks and keep track of things like read/write access. The Transaction PM is currently not implemented and transaction control is performed by the Exodus storage manager.

Distribution PM. Also known as the Location PM, this module allows objects to be in separate address spaces unbeknownst to the end-user. The Distribution PM does this by "hiding the distinction between address spaces, thus giving the illusion of a large, flat address space"[65]. Open OODB describes two ways to implement transparent distribution. The first is to let the operations span address spaces. The second is to make sure that the executing function and its arguments are at the same site. Open OODB leaves the choice to the user. The Distribution PM is

[12]The *A* is for atomicity, the *C* for consistency, the *I* for isolation and the *D* for durability

unimplemented and distribution control is left to the Exodus storage manager.

Change Management PM. This module's functionalities are described differently in different documents. In the documentation included specifically with the alpha release, the Change Management PM is described as follows. "The scope of the Change PM module includes three orthogonal subcomponents: *version management*, ... *configuration management*, ... and *dependency management*, which manages relationships between associated representations. *Meta data evolution* is viewed as an important application of the Change PM for managing change of data dictionary information"[59]. In [65], and the later [66], "The Change PM supports the extensions of object versions and object configurations." It appears that dependency management and meta data evolution have been dropped from the Change PM, therefore we do not discuss them.

Version and configuration management are looked at more closely in the DISWG Advanced requirements class where there is a requirement for their support. Even though this module is unimplemented, it is an active research area for the Open OODB project.[13] In its final form, Change Management may well be divided into three separate modules supporting: 1) replicated and/or partitioned objects, 2) versions and 3) configurations [65]. Interestingly, OMG[14] is wrestling with similar problems on how to best manage change management [57].

Indexing PM. This module is "responsible for instantiating indices over sets and for maintaining their consistency as set membership or values of objects in the set change. ... However, even within this rather straightforward looking requirement, there is a wide range of variation"[65]. Different indexing policies include immediate update, commit time update and on-demand update where updates are performed

[13] According to an *email* correspondence with Steve Ford at TI on April 21, 1994, Change Management will not be included in the next release of Open OODB due in the summer of 1994, but in a subsequent release.

[14] A brief review of OMG is provided in the next subsection.

when needed. This module is unimplemented. In OMG's Object Services Architecture (OSA) proposal, indexing is not a separate service, but a subcomponent of a query service.

Query PM. "The Object Query module provides end-users, application programmers and other modules efficient access to large collections based on their content's structural and behavioral properties"[59]. It consists of the query language and the query compiler, which is responsible for query optimization. Query capability is one of Open OODB's most well developed functionalities. However, Open OODB currently has no separate Query PM and the query capabilities are not modularized.

In summary, Open OODB does not implement the Change Management or Indexing PM modules. Also, Open OODB does not currently have functional Transaction and Distribution PM modules as these are both handled by the underlying Exodus storage manager [36]. Finally, Open OODB does not have a separate Query PM module, leaving Persistence as the only PM module implemented so far.

2.4 The Object Management Group

Open OODB's findings parallel the findings of the influential Object Management Group (OMG) [58]. Indeed, one of Open OODB's requirements (R13-9) states that Open OODB modules must conform to OMG's architecture. Open OODB is sponsored by the military; OMG by an industrial consortium with over 200 members [94]. OMG provides an Object Services Architecture (OSA) document [57] which details the services needed in next-generation OODBs. Since Open OODB and OMG have much in common, we reference OMG throughout the evaluation of Open OODB with respect to DISWG's requirements. Therefore, we now provide some background on OMG.

"OMG is dedicated to producing a framework and specifications for commercially

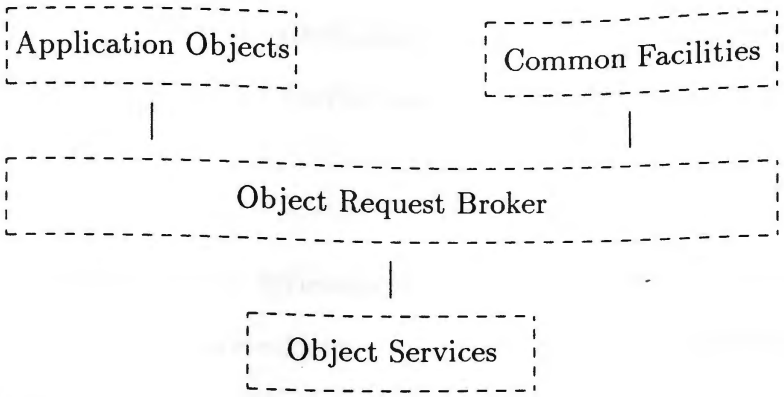

Figure 4: OMG's CORBA Model

available object-oriented environments"[57]. OMG's initial results consist of the OSA which is more specifically referred to as the Common Object Request Broker Architecture (CORBA). At the heart of CORBA is the Object Request Broker (ORB), which brokers requests between application objects and object services. Figure 4 shows the four main parts of OMG's Reference Model.[15]

- The **Object Request Broker** enables objects to make and receive requests and responses in a distributed environment.
- **Object Services** is a collection of services (interfaces and objects) that provide basic functions for using and implementing objects.
- **Common Facilities** is a collection of services that provide general purpose capabilities useful in many applications.
- **Application Objects** are objects specific to particular end-user applications.

OMG's list of required services include: object translation, persistence, concurrency control and transactions, object naming, data dictionary, versioning and configuration management, queries, relationships, events and security.[16] Open OODB sees this list as a superset of its own requirements [59]. This discussion of OMG is

[15]This list of OMG's four parts is taken verbatim from page 2 of [57].
[16]This list is incomplete; for the complete list see [57].

necessarily brief. For a more detailed critique, see [54]. The fact that Open OODB and OMG are obtaining similar results and are working towards similar goals will help both endeavors.

This concludes the section reviewing Open OODB. Many more details can be found throughout the nine subsections which evaluate Open OODB with respect to the DISWG requirements.

3 Research at the University of Rhode Island

URI is interested in extending Open OODB to include new functionalities. Some of these functionalities are real-time features and relationship objects that, among other things, enforce integrity constraints. Both Real-Time and Integrity are DISWG requirements that Open OODB does not currently meet.[17] Thus, we cite URI's work extensively in the sections on real-time, integrity constraints and elsewhere, necessitating this brief review.

This section is organized as follows. First, we examine URI's overall research goal of developing a real-time OODB. This discussion includes a look at RTSORAC [73]. Next, we take a brief look at operating system support needed for real-time databases and DBMSs in general. Then, we look at some of the author's research that formed a background for this thesis.

3.1 Real-Time Databases

"A *real-time database* is a database in which both the data and the operations upon the data may have timing constraints. The RTSORAC (**Real Time Semantic Objects Relationships And Constraints**) group at the University of Rhode Island (URI) has integrated real-time, object-oriented, semantic and active database approaches to

[17] We look at this in more detail in the appropriate subsections.

develop a formal model for the specification of objects, relationships, constraints, updates and transactions"[73]. This group is going to extend Open OODB to include RTSORAC features. In the next two subsections, we briefly present the RTSORAC model and discuss some possible limitations.

3.1.1 The RTSORAC Model

"RTSORAC has three components which model the properties of a real-time, object-oriented database: *objects*, *relationships* and *transactions*. *Objects* represent database entities. *Relationships* represent associations among the database objects. *Transactions* are executable entities which access the objects and relationships in the database"[73]. We next look at each of these three components.

Objects have five components: $< N, A, M, C, CF >$.[18] N is an identifier and A is the set of attributes. In A, it is possible to express an imprecision amount and a timestamp associated with a particular attribute. M is the set of methods. C is the set of constraints used to maintain the object's correct state. C can express both logical and temporal constraints. Logical constraints ensure valid data states under write operations and temporal constraints ensure valid data states under the progression of time. CF is a compatibility function used for concurrency control.

"A relationship is an object defined by $< N, A, M, C, CF, P, IC >$"[96]. The first five components are the same as in an object. P represents the set of objects participating in the relationship. "IC is a set of inter-object constraints placed on objects in the participant set"[96]. IC performs much as C does in an object, only IC can invoke the methods of objects participating in a relationship. The intra-object constraints of an object and the inter-object constraints of a relationship clearly specify integrity constraints, permitting implementation of mechanisms such as triggers and assertions, etc.

[18] Each of these five components has subcomponents. Space considerations preclude us from discussing all of them. A full description of the model is in [73].

A transaction has six components: $< N_t, O, OC, PreCond, PostCond, Result >$. N_t is an identifier and O is the set of operations involved in the transaction. OC are constraints on those operations. $PreCond$ and $PostCond$ are pre- and post-conditions associated with the transaction and $Result$ is the result of the transaction.

Implementation at URI will proceed as follows. There is already a prototype of SORAC (RTSORAC without the real-time features) that uses the ONTOS database system. SORAC will be extended to be real-time and ONTOS will be replaced by Open OODB. Open OODB will be ported to a real-time, POSIX-compliant [92] operating system. Open OODB's extensible, modular nature make it a good candidate for the extensions URI wishes to make.

3.1.2 Potential Limitations of RTSORAC

RTSORAC represents the most developed model we know of for a real-time OODB. However, there are some practicalities that may impede implementation. One concern is that RTSORAC is too far reaching. Perhaps it would be better to concentrate on a subset of the model rather than trying to do so much all at once. Also, it is unclear how scalable the model will be from at least two distinct perspectives. One concern is whether the model could cope with a system including many objects involved in many, complicated relationships. How hard will the many, involved relationships be to express and what impact would there be on performance?

Of course, if the schema designer writes efficient code, then the above potential problem is minimized. A problem that can not be alleviated by "good code", however, is the limitations of the underlying operating system. Typical database activity of one or two hundred transactions would overload state-of-the-art, real-time operating systems. For instance, in an earlier model at URI, it was assumed that every method of an object was a separate thread. Since a transaction may include several method invocations, there could be too many threads active for the operating system to

handle. This is an ongoing area of research at URI and, in the latest model, each method is not an individual thread.

The preceding discussion on operating system support raises a very important question: where does the operating system end and the DBMS begin? We briefly consider this question in the next subsection.

3.2 The Operating System/DBMS Interface

When considering real-time databases, it is impossible to ignore the role of the operating system. Furthermore, the question of exactly where the interface lies between between the operating system and the DBMS can not be ignored in traditional database systems as well. In this subsection, we take a brief look at why this is the case.

A DBMS's performance is intrinsicly tied to the underlying operating system.[19] DBMSs can provide many services including, but not limited to: transaction scheduling and locking, real-time features, recovery algorithms, security checks, distribution management and memory control. Operating systems can also provide many services including, but not limited to: task scheduling and locking, real-time features, recovery algorithms, security checks, distribution management and memory control. The juxtaposition of the previous lists illustrates the problem. There is an obvious redundancy of functionalities.

The key question is: how can this redundancy be lessened or removed? The algorithms used by each system are very similar. The following are a few examples. In a DBMS, one scheme to prevent deadlock is the wait-promote-protocol [2]. In the wait-promote-protocol, if a transaction holds a lock, that transaction's priority is elevated to be the same as the priority of the highest priority transaction waiting for the same lock. In an operating system, priority inheritance does essentially the same

[19] Most of what is discussed in this subsection has been extracted from an unfinished document entitled "Operating System Support for Next Generation Database Management Systems" by Paul Fortier and Joan Peckham.

thing at the task level. Also, DBMSs typically rely on recovery schemes that involve roll-back, as do operating systems. Scheduling algorithms are basically the same at both levels. In real-time databases, transactions have start times and deadlines; in operating systems, tasks have start times and deadlines.

One possibility is to consider the actions of a DBMS to be a refinement of the actions of an operating system. In other words, a transaction could be modeled as a task with some important revisions. To accomplish this, there would have to be new operating system standards that support basic database requirements. The operating system would provide interfaces to the required functionalities. If a certain database requirement needs to be supported, plug a module that fulfills those requirements into the appropriate operating system interface. This type of a system would facilitate the move into the next-generation of database systems.

In summary, cooperation between the operating system and the DBMS is an area that needs further research. "It is important that future operating system designers become more sensitive to database management system's needs"[88]. Many of the DISWG requirements for real-time and fault tolerance can not be met without support from the operating system. Therefore, this brief discussion is referred to in the evaluations of DISWG's real-time and fault tolerance requirements.

3.3 The Author's Individual Work

In this section, some of the author's work that led to this thesis is discussed. This thesis work has gone through three distinct phases, all of which are pertinent to this finished product. Now, each of the three phases is looked at briefly.

Initially, the topic was to add fault tolerance capabilities to the POSIX-compliant [92], real-time Lynx operating system. The intent was to adapt fault tolerance algorithms used in MACH [6], which in turn were adapted from algorithms used in a fault tolerant version of UNIX [11]. The algorithms create shadow processes for all

processes in the system to protect against the failure of any, individual process. The original process and its shadow must be on different nodes. Periodic checkpoints are taken and messages sent after a checkpoint are logged. Upon failure of a process, its shadow is activated and the shadow is "caught-up" using the logged messages. This work with fault tolerance has helped in the evaluation of DISWG's Fault Tolerance requirements class.

When URI was chosen as an alpha test site for Open OODB, the author became part of a group working towards real-time extensions to Open OODB. Part of this project is to port Open OODB to the Lynx real-time operating system. Open OODB uses the Exodus storage manager, which has to be ported to Lynx as well. Since the author was experienced with Lynx due to the fault tolerance work, the new thesis topic became to implement the port of Exodus. Initially, time wqs spent using the facilities of Exodus in order to understand its capabilities. Then, work on the port was started. This proved more difficult than originally thought due to compiler incompatibilities. Lynx uses an early version of the GNU C++ compiler and Open OODB uses a much more recent AT&T release. The port would have involved making tedious, minor changes to possibly hundreds of files and libraries and this was deemed unsuitable for thesis work. However, the work with Exodus, and participation in the real-time database group, helped the author to gain an understanding of the Open OODB system.

The last stage is reflected in this thesis. It was determined that an evaluation of Open OODB with respect to the DISWG requirements was needed to fuel future research efforts at URI. This thesis provides that evaluation.

4 DISWG

The armed services, like most large corporations, rely on the efficient management of data. However, in an armed service like the Navy, life-or-death situations may

depend on such management. To facilitate efficient access, the Navy has embarked on a standardization effort. This effort, initiated by the U.S. Navy Space and Naval Warfare Systems Command (SPAWAR), is called the Next-Generation Computer Resources (NGCR) program. The NGCR Database Management System Interface Standards Working Group (DISWG) was formed by the NGCR in 1992. DISWG "is chartered to identify and help define nonproprietary commercially-based DBMS interface standards for use in the development and maintenance of future mission-critical computing systems"[32].

Towards that goal, DISWG has published a requirements document. DISWG's goal in devising this document is to help guide the Navy's development of DBMS standards. Also, DISWG wants to allow for flexibility so that old standards can evolve and new standards be developed. Thus, DISWG chose to focus its requirements on interfaces. This choice was made to "avoid dictating requirements that would unnecessarily constrain the design or implementation of DBMSs or applications"[32].

DISWG divides its requirements into the following nine requirements classes.

1. General Requirements.
2. Basic Database Management Services.
3. Distribution.
4. Heterogeneity.
5. Real-Time Processing.
6. Fault Tolerance.
7. Integrity.
8. Security.
9. Advanced Database Management Services.

In the next nine subsections, Open OODB is evaluated with respect to each of the nine DISWG requirements classes. Each requirements class evaluation has the same format. First, there is some introductory material providing any necessary background or definitions. Then, we evaluate all the requirements of the class. This evaluation is performed with respect to Open OODB's requirements, its proposed architecture and its implementation. If the requirement is not met by Open OODB, where appropriate, we provide a brief discussion on the feasibility of extending Open OODB to meet the requirement. To maintain consistency, the discussion of extending Open OODB is always included in the evaluation of Open OODB's implementation. As a final note before the evaluations start, Open OODB provides a C++ and a Lisp implementation. We are concerned only with the C++ implementation and all comments, except where specifically noted, refer to the C++ implementation [61].

4.1 DISWG's General Requirements Class

The General requirements address "general goals (e.g., scalability, modularity, extensibility, configurability) of interface standards"[32]. These requirements could be applied to any system, not just to databases. However, many of these requirements highlight the difference between DISWG and Open OODB. DISWG is a list of requirements on interface standards and Open OODB is a specific proposed architecture and implementation. The DISWG requirements need to be as all encompassing as possible, but Open OODB has had to make concrete, implementation decisions. For instance, DISWG may place its requirements on an interface to be independent of any particular language, but Open OODB must pick a language for its implementation. This important difference is referred to throughout the discussions of the requirements in this and other classes. We now evaluate Open OODB with respect to the fifteen requirements of this class.

4.1.1 Public Specification

The NGCR DBMS interface standards shall be based on public specification.

Public specification is defined as "specifications available, without restrictions, for implementation and distribution of an implementation"[32].

Open OODB

Requirements. Open OODB has a matching requirement. In an earlier draft of the DISWG requirements, public specification is referred to as *open* specification. Open OODB's meta requirement R13 is entitled Openness [69]. Open OODB defines openness as "the characteristic of a system which allows developers or researchers to modify or control some part(s) of its architecture or implementation"[60]. This definition strongly matches DISWG's definition of public specification. Also, remembering that DISWG lists a set of requirements on interfaces, it is interesting that Open OODB's R13-1 requirement states that interfaces must be well documented.

Proposed Architecture. Open OODB's proposed architecture meets this DISWG requirement. Open OODB has been designed with openness in mind. Open OODB's goal is to provide a modular framework that allows researchers to design customized OODBs. As stated above, openness and public specification mean very much the same thing.

Implementation. This requirement is met. Open OODB provides documentation and source code with its release. However, due to the preliminary status of Open OODB's implementation, Open OODB's documentation lacks a definitive description on just how to interface with its modules.

4.1.2 Portability

The NGCR DBMS interface standards shall promote the portability of application software, data, and users (end users, application programmers, and database administrators (DBA's)).

DISWG defines portability as "the ease with which software (and, in the broad sense, data and users) can be transferred from one system to another"[91].

Open OODB

Requirements. Open OODB has a matching portability requirement. Indeed, portability is also a goal of OMG and should be a goal of any open system. Open OODB was motivated in large part by the lack of a standard application program interface. This lack has slowed acceptance of OODBs due to portability concerns [70]. Thus, Open OODB sees that portability must be required.

Open OODB's portability requirement is subrequirement R16-5 under R16 Industrial Strength [69]. Open OODB wishes to be independent of any R16-5-1 hardware, R16-5-2 operating systems, R16-5-3 programming languages, R16-5-4 compilers, R16-5-5 storage managers and R16-5-5[20] environments. At a lower level, under R2 Persistence, R2-4-4-1 states that an object's external representation must be portable across compilers, operating systems, and machine architectures. Also, under R6 Query Capability, R6-15 states that the query system must be retargetable across OODBs and programming languages. Under R12 Program and User Interfaces, R12-2-1 states that an OODB should support a user interface that is portable to a wide variety of platforms. Finally, under R7 Change Management, R7-8 says that Open OODB's change management facilities must be portable to other OODBs.

[20]This duplicate requirement number must be a typo in the Open OODB requirements document.

Proposed Architecture. Open OODB's proposed architecture would meet this requirement. Open OODB has been designed with portability in mind.

Implementation. Open OODB's implementation does not meet this requirement. Open OODB is currently tied to the following [58].

- Hardware: Sun SPARCstations.
- Operating System: 4.1.x of SunOS.
- Compilers: AT&T C++ Release 2.1.12, Sun C++ Releases 2.1 and 3.0.1.
- Persistent store: Exodus Storage Manager.
- Languages: C++ and CMU Common Lisp version 16f.

Open OODB is currently attempting to increase portability by adding compilers. At URI, we have had some experience with the portability problems. We wish to add real-time features to Open OODB and towards this end have attempted to port Open OODB to POSIX-compliant, real-time LynxOS. LynxOS only supports GNU C++. We have found this to be a challenging problem.

There are (at least) two portability concerns with respect to language independence. First, Open OODB needs to come up with a standard external C++ representation [63]. Lack of such currently pins objects to the creating machine environment. A consequence of coming up with a standard representation will be an extra translation step and thus, slower performance. Second, the fact that Open OODB takes a programming language specific approach (i.e., persistent C++) makes multiple language support difficult [70].

Open OODB's reliance on the Exodus storage manager creates another set of problems. Exodus performs many basic DBMS services at a lower level than Open

OODB. Among those services are transaction control, locking, recovery and distribution control. Thus, many of the proposed functionalities of Open OODB would, if implemented, be repeated by Exodus with an obvious impact on performance. However, if these are left unimplemented, then portability suffers.

Despite these problems, nothing precludes a future Open OODB implementation from meeting this requirement.

4.1.3 Interoperability

The NGCR DBMS interface standards shall promote the interoperability of DBMSs and applications.

DISWG defines interoperability as "the ability of two or more systems (or, more generally, two or more *components*) to exchange information and to mutually use exchanged information"[91].

Open OODB

Requirements. Open OODB has no specific interoperability requirement. This is an area where the difference between DISWG and Open OODB is highlighted. DISWG is a list of requirements on interfaces and this requirement says that the interfaces should not preclude interoperability. Open OODB represents an architecture and an implementation of a specific DBMS and as such, would not have the same concerns for interoperability.

However, Open OODB requirement R11 states that "an oodb must support access to legacy data stored in an SQL-compatible relational database or file"[69]. Legacy data is defined as "data that already exists and is used and useful in pre-existing applications prior to the introduction of OODB technology"[60]. Requirement R11 is outside the scope of the Open OODB's current effort, but is an area of current research.

Proposed Architecture. This is unmet. Nothing in the proposed architecture addresses the issue of interoperability.

Implementation. This is unmet. However, Open OODB's modular design and well defined interfaces would enhance Open OODB's ability to interoperate with other DBMSs. As noted above, Open OODB is actively pursuing research in this area. Open OODB envisions a CORBA-like environment with one implementation of Open OODB treated as an individual object requesting services from other objects. The other objects could be more Open OODB implementations, or other, possibly heterogeneous, database implementations. Such an environment would meet this requirement.

4.1.4 Supportability

The NGCR DBMS interface standards shall promote maximum life cycle-supportability features for conforming products.

Open OODB

Requirements. This requirement is matched. A subrequirement of the R16 Industrial Strength meta requirement, R16-11 support, states that an OODB must have R16-11-1 quick response to bug reports, R16-11-2 user group meetings, R16-11-3 next release features and schedule available, R16-11-4 continued development in progress and upgrades and, finally, R16-11-5 long-term support.

Proposed Architecture. This requirement is met. The open, extensible and modular nature of Open OODB's architecture allows for supportability.

Implementation. This is met for reasons just stated.

4.1.5 Hardware Independence

The NGCR DBMS interface standards shall be independent of any particular hardware platform.

Open OODB

Requirements. This requirement is matched. Under R16 Industrial Strength, subrequirement R16-5-1 implies that an OODB should be independent of any hardware platform [69].

Proposed Architecture. This is met by the proposed architecture. Open OODB's modular nature and its ability to deal with different types of address spaces and data formats account for hardware differences. Also, Open OODB's proposed Translation module is designed to translate objects between hardware platforms [70].

Implementation. This is unmet by the current implementation. As mentioned previously, Open OODB is currently tied to Sun SPARCstations [58], and the lack of a standard, external C++ representation pins Open OODB to its underlying machine [63]. However, Open OODB has been designed with a goal of hardware independence, which should facilitate ports to other hardware platforms. Also, Open OODB is implemented on UNIX and uses the TCP/IP protocol. The wide acceptance of both enhances portability. Finally, its encouraging that another C++ based DBMS, GemStone [13], supports a wider degree of hardware heterogeneity than does Open OODB.

4.1.6 Operating System Independent

The NGCR DBMS interface standards shall be independent of any particular operating system.

Open OODB

Requirements. This is matched. Under R16 Industrial Strength, R16-5-2 implies that an OODB must be OS (operating system) independent.

Proposed Architecture. The proposed architecture meets this requirement.

Implementation. This is unmet. Open OODB is tied to 4.1.x of SunOS [58]. However, portability and operating system independence are Open OODB requirements, meaning that Open OODB has been designed with those requirements in mind. The proposed Translation module would translate objects between formats required by different operating systems.

Also, Open OODB rests on the very popular UNIX operating system. This certainly enhances operating system portability. For instance, work is underway to at URI to port Open OODB to a POSIX-compliant operating system. POSIX is emerging as the de facto operating system standard and POSIX is based on UNIX.

4.1.7 Network Independent

> *The NGCR DBMS interface standards shall be independent of any particular network.*

DISWG's requirements defer to NGCR's SAFENET standards governing networks.

Open OODB

Requirements. There is no directly matching Open OODB requirement. This is interesting since there is an OODB requirement for almost every other type of independence (R16-5). However, under R4 Distribution, R4-13 requires that the OODB "insulates client-server from network protocol"[69].

Proposed Architecture. Open OODB's proposed architecture meets DISWG's network independence requirement. Network independence is alluded to in architecture documents. The Communication support module is responsible for the actual movement of bytes. This movement is between one or more communications media which may include networks [66]. The Translation module translates objects into different representations required by different communications mechanisms [70]. The Distribution PM chooses the appropriate translation format. Thus, the Communications, Distribution and Translation modules could be implemented to allow DISWG's network independence requirement to be met.

Implementation. This is unmet. However, as stated above, the full implementation of Open OODB's Communications, Translation and Distribution PM modules would allow this DISWG requirement to be met. Also, the fact the Open OODB uses the popular TCP/IP protocol would facilitate extending Open OODB to meet this requirement.

4.1.8 Programming Language Independent

> *The NGCR DBMS interface standards shall be independent of any particular programming language.*

This is essential because the Navy is already using several programming languages and is sure to need to accommodate even more in the near future (i.e., C++). Thus, the next-generation of Navy databases should not be tied to any particular set of languages.

Open OODB

Requirements. Open OODB has no matching requirements. However, several come close. Under R16 Industrial Strength, R16-5-3 requires that an OODB support

C++, Lisp and multiple language interfaces such as Ada, Fortran and C, etc. Therefore, this requirement does not state that Open OODB must be language independent, but that it should support multiple languages.

Also, R1-2, under R1 OO Data Model, requires that this model *may* be one of six different things. One of those six choices, R1-2-2, states that this OO model may be a new object model that is language-independent and independent of C++, Lisp, etc. This requirement addresses the DISWG requirement of language independence, but because it is one of six choices that could be made, does not outright require an OODB to be language independent. The other five choices do not require language independence. However, a subrequirement of one of the five, R1-2-6-3, says that an OODB should not preclude supporting a common OODB semantics across languages. In other words, the query language should not rely on the underlying (or embedding) programming language. This requirement is seen by Open OODB as desirable, but currently unmet.

There are other Open OODB requirements that partially match this DISWG requirement. Under R4 Distribution, R4-3 states that an OODB must support a computing environment that is heterogeneous with respect to the host programming language. This requirement is outside the current scope of Open OODB. Under R13 Openness, R13-7 states that modules must have a language independent architecture and implementation.

Proposed Architecture. This is met. Its modules have been designed to be independent of any particular language.

Implementation. This is unmet. This is a requirement that demonstrates the difference between DISWG and Open OODB. The DISWG requirements are on a standard, meaning that the standard should be programming language independent. An actual implementation, such as Open OODB, has to pick an implementation

language(s). The languages supported by Open OODB are C++ and Lisp, although feasibility studies have shown that Ada 9X could be supported.

The choice of C++ was easy: C++ is emerging as a de facto standard. Many other next-generation databases provide a C++ interface. Examples include Ode [39], GemStone [13] and ObjectStore [48]. Even extended relational systems such as Starburst have begun using C++ [49]. Open OODB has tried to make its architecture language independent, but may encounter difficulties in porting to other languages. This is due to an early implementation decision made by Open OODB. Designers of OODBs have three choices for their OO data model: programming language neutral, a database programming language or programming language specific. Open OODB uses the programming language specific approach which could make interoperability across different language's type systems difficult and limit programming language independence [59].

4.1.9 DBMS Independent

The NGCR DBMS interface standards shall be independent of any particular vendor or implementation.

Open OODB

Requirements. There are no matching requirements.

Proposed Architecture. This is not met. Open OODB *is* a DBMS and, therefore, can not propose an architecture that is independent from itself.

Implementation. This is unmet. This requirement once again highlights the main difference between DISWG and Open OODB.

4.1.10 Scalability

NGCR DBMS interface standards shall accommodate changes in volume of data, number of users, or transaction rates.

Open OODB

Requirements. This DISWG requirement is matched. Under Open OODB's R15 Performance meta requirement, R15-5 states that an OODB should be scalable. Open OODB then presents a list of what should be scalable which does not exactly match DISWG's list. Open OODB mentions the size of the database (number of objects, total space) and the number of users but does not explicitly refer to the transaction rate. However, if the OODB is scalable with respect to size and users, etc., then transaction rate scalability is implicitly covered. In addition, Open OODB's R15-1-6 says that performance measurements should be scalable with respect to large objects, large numbers of objects and large numbers of users.

Proposed Architecture. This is partially met even though scalability is not mentioned in Open OODB's architecture documents. Open OODB is structured as an "object services architecture" connected to many services. These services could include multiple copies of Address Space Managers to handle many remote sites. These services could also include multiple copies of PMs. Not all services have to be in a particular implementation. This allows for configurability and a degree of scalability.

Implementation. This is partially met and nothing in Open OODB's design would preclude it from being fully met. Since scalability is an Open OODB requirement, Open OODB has been designed with scalability in mind. Some of Open OODB's scalable characteristics were just mentioned in the proposed architecture section. Another aspect of scalability is the ability to handle huge stores of data. This is an active research area for Open OODB and others, including OMG. It is interesting to note

that the designers of GemStone state that scalability is one of the three "most difficult aspects of a database management system with regard to development and successful deployment"[13]. However, "It will be necessary to *scale* all DBMS algorithms to operate effectively on databases of the size contemplated by next-generation applications, often several orders of magnitude bigger than the largest databases found today"[84].

4.1.11 Modularity

The NGCR DBMS interface standards should adhere to the design principal of modularity. That is, database functions should be grouped into modules that can be understood independently.

Open OODB

Requirements. Whereas Open OODB does not have a requirement that says that an OODB must be modular, Open OODB's goal is to develop an open, modular OODB toolkit. Thus, a main premise of the Open OODB project is its modularity. All the subrequirements of Open OODB's R13 Openness meta requirement mention various aspects of modularity. For instance, R13-1 says that module instances must be documented, R13-2 deals with configurability of modules in the system and R13-3 says that each module must be justified, etc. Thus, an open system implies a modular system.

Proposed Architecture. The proposed architecture meets this requirement. Open OODB has been designed with modularity in mind. Note that modularity is also a design goal of OMG. Modularity is facilitated by the nature of OO design, i.e., modules are objects with well defined interfaces and hidden, independent implementations. Open OODB's modules should be easily modified or replaced, and indeed,

some groups have removed modules and replaced them with customized ones. For instance, the University of Florida has replaced the persistent Address Space Manager [59].

Implementation. This requirement is partially met. Open OODB has adhered to the design principal of modularity. However, most of its modules are, thus far, only proposed. For instance, the only implemented PM is the Persistence PM and Open OODB's query functionality, one of the most developed parts of the system, is not modular.

4.1.12 Extensibility

> *The NGCR DBMS interface standards should facilitate development and use of extensions: e.g., interfaces should be composable so that they can be combined to create new interfaces and facilities. It should be possible to add new interfaces for new functions.*

Open OODB

Requirements. Open OODB does not have a specific extensibility requirement, but creating an extensible system is a design goal of Open OODB. Extensibility goes hand in hand with openness [59] which is an Open OODB meta requirement. Therefore, much of the flavor of this DISWG requirements class is captured in Open OODB's R13 Openness requirement. For instance, R13-2-1 states that alternative implementations of modules can replace modules and R13-5 states that some modules are extensible.

Proposed Architecture. This is met. Open OODB's architecture has been designed with extensibility in mind. A stated goal of Open OODB is to provide a customizable OODBMS framework which implies extensibility.

Implementation. This is met. A major motivation for this thesis is to provide an evaluation of Open OODB to facilitate researchers at URI making extensions to Open OODB. Open OODB has been picked for this work due, in part, to its extensibility. Extensibility is also a design goal of OMG [57]. The designers of Starburst note that "extensibility can not be retrofitted; it must be a fundamental goal and permeate every aspect of the design"[49].

4.1.13 Uniformity

> *The NGCR DBMS interface standards should be based on a consistent set of unifying, well-defined conceptual models. Interface features should uniformly address aspects such as status return, exceptional conditions, parameter types, and options.*

Open OODB

Requirements. There is no matching Open OODB requirement. However, uniformity is mentioned in the Open OODB literature as is discussed below.

Proposed Architecture. This is met. Open OODB uniformly applies the OO paradigm throughout its design; e.g., PMs are objects. The Open OODB "Meta Architecture Support (MAS) module implements the mechanisms to uniformly extend events, and defines interface conventions used by other Open OODB modules"[59]. The MAS provides a common interface for all PMs as well as a common exception mechanism. To do so, PMs are implemented as a type lattice with all PMs having a common ancestor and, therefore, a common interface [64].

Implementation. Open OODB's implementation should meet this requirement, but the preliminary stage of development of most of Open OODB's modules precludes

us from saying it is met. Certainly, if Open OODB's designers are able to implement their proposed architecture, then this requirement would be met.

4.1.14 Configurability

The NGCR DBMS interface standards shall support the configurability of DBMS implementations so that a given implementation can be tailored to a specific application.

Open OODB

Requirements. Open OODB has a matching requirement. Under R13 Openness, R13-2 states that "different system configurations are possible"[69] and that modules themselves may be reconfigurable. R13-2-1 states that modules may be implemented differently and R13-2-1-3 states that generating new configurations must be efficient. Also, R13-2-2 states that not all modules need to be used in an implementation.

Proposed Architecture. The Open OODB architecture meets this DISWG requirement. Indeed, the main goal of Open OODB is to provide an OODBMS framework that can be custom-tailored to meet particular needs [58]. Open OODB provides a diagram of sample configurations in [59]. Open OODB's modularity facilitates reconfiguration.

Implementation. This is only partially met because many functionalities have not yet been implemented. If Open OODB were fully implemented, then this should be met. However, Open OODB is not sure how to make reconfigurations efficient [69]. Also, Open OODB's designers state that, while they are attempting to be as open as possible, they "are not trying to support complete system reconfiguration at no cost, as we believe it to be infeasible"[67]. Note that configurability is also a goal of OMG [57].

4.1.15 Compatibility with Other NGCR Standards

The NGCR DBMS interface standards should be compatible with other NGCR standards. In particular, they should be implementable on operating systems that conform to the NGCR Operating System Interface (OSIF) Standard [OSIF 93] and (in the case of distributed data) on networks that conform to the NGCR SAFENET Standard [SAFENET 93]. Moreover, the DBMS services specified in the NGCR DBMS interface standards should be accessible from application programs written in Ada and other programming languages in common use for MCCR applications.

Open OODB

Requirement. There is no matching Open OODB requirement. Under R13 Openness, R13-9 states the OODB modules conform to other framework architecture such as that proposed by OMG [69]. However, since Open OODB is not affiliated with the Navy, it has no requirements for meeting NGCR standards.

Proposed Architecture. The proposed architecture does not meet this requirement.

Implementation. This is not met. This is not a goal of the Open OODB project. However, since Open OODB is designed with openness, extensibility and modularity in mind, it should be possible to extend Open OODB to meet this DISWG requirement on an "as needed" basis. At URI, work is underway to port Open OODB to a POSIX-compliant operating system. This port should be facilitated by the fact that Open OODB is built on Unix. POSIX represents an extension of UNIX, the dominant operating system. Additionally, Open OODB has been designed with distribution and network independence in mind. This should allow the NGCR SAFENET standards on networks to be met. Also, Open OODB has completed a feasibility study with

positive results on adding Ada 9X as a supported language. Thus, Open OODB can be extended to meet NGCR standards.

4.2 DISWG's Basic DBMS Requirements Class

"This category represents basic services typically provided by today's general-purpose DBMSs and which must be included in NGCR DBMS interface standards"[32]. Note that these requirements, since they are general purpose, do not provide support for mission-critical computing. Such support is left for other requirements classes to be evaluated shortly.

The evaluation of some of the requirements of this class posed a problem. DISWG lists six distinct types of queries: planned, ad hoc, interactive, embedded, compiled and interpreted. However, the definitions of some of these types are somewhat hazy and this leaves room for doubt as to exactly what the requirements mean. Now, we review each of the twenty-four requirements of this class.

4.2.1 Persistent Data

> *The NGCR DBMS Interface Standard shall provide support for the management of persistent data.*

Persistent data is data which outlasts its creating process [91].

Open OODB

Requirements. This is matched. Open OODB's R2 Persistence functional requirement states that "an oodb must support persistent storage of object instances and classes supported by its object-oriented data model(s)." Subrequirements R2-1 through R2-8 deal with different aspects of persistence. Also, under R14 Seamlessness, Open OODB requirement R14-3 deals with issues involving persistence. For

instance, R14-3-1 requires that persistence must be orthogonal to class/type and R14-3-2 requires that instances can be either transient or persistent.

Proposed Architecture. Open OODB's proposed architecture meets this DISWG requirement. Open OODB seamlessly adds functionalities such as persistence [58]. Applications which need to use persistent objects interface with Open OODB's Persistence PM. The Persistence PM uses the services of many other Open OODB modules such as: the Distribution PM, the Translation, Communications, Data Dictionary and Address Space Managers and the sentries. It may also use the Transaction PM and Change PM [59].

Implementation. This is met. The current implementation of Open OODB hands the responsibility of managing persistent data to Exodus. In essence, Open OODB tells Exodus to persist objects and Exodus does what is required.

4.2.2 Multiple Users

> *The NGCR DBMS interface standards shall provide support for multiple simultaneous users and application programs, all using any combinations of queries, DMLs, DDLs, and DCLs.*

The DML is the Data Manipulation Language, the DDL is the Data Definition Language and the DCL is the Data Control Language.

Open OODB

Requirements. This DISWG requirement is matched. Open OODB's R3 Concurrent Access functional requirement states that an "oodb must provide for sharing and controlled concurrent access by multiple users/processes." Also, under the R1 OO Data Model functional requirement, R1-2-6-2 states that an OODB must be multilingual and allow inter data model sharing [69].

Proposed Architecture. Open OODB meets this DISWG requirement. Users may be simultaneously querying, manipulating, defining or controlling data. Also, Open OODB provides support for multiple languages, C++ and Lisp, being used at the same time.

Implementation. This is met. The fact that Open OODB's DML, DDL, DCL and queries are all written in the same high level language(s) helps.

4.2.3 Conventional Alphanumeric Data Types

The DML(s), DDL(s), and DCL(s) specified in the NGCR DBMS interface standards shall provide the capability to define and manipulate conventional alphanumeric data types, including integer, real, and character string.

Open OODB

Requirements. This is somewhat matched. Under R1 OO Data Model, R1-2-1-3-6 requires that the data model must support first class coverage for types (not just objects). R1-3-12 requires that instances of C types must be supported as first class independently persistent sharable objects. Under R2 Persistence, R2-1-2-2 states that an OODB must support persistence for variables. Note that, since Open OODB data is defined and manipulated through C++, and C++ supports conventional data types, Open OODB supports the same. Thus, this DISWG requirement is implicitly matched by Open OODB, although no Open OODB requirement explicitly matches the wording of DISWG.

Proposed Architecture. The Open OODB architecture meets this DISWG requirement. As stated above, because Open OODB is based on C++ which supports conventional data types, Open OODB supports conventional data types.

Implementation. This is met due to Open OODB being an extension of C++. Open OODB does not, however, meet its own requirements with respect to treating these conventional data types as first class objects. The current implementation does not allow for these types to persist.

4.2.4 Binary Large Objects (BLOBs)

> *The NGCR DBMS interface standards shall provide the capability to define and manipulate BLOBs.*

A binary large object (BLOB) is defined by DISWG as "long, variable-length sequences of bits or bytes used to represent non-conventional data, such as graphics, image, audio, and video objects"[32]. BLOBs are the types of non-conventional data that have helped to promote research into next-generation OODBs [15].

Open OODB

Requirements. This requirement is matched. Under R15 Performance, R15-1-6 requires that an OODB should be scalable with respect to single, large objects. Also, under R1 OO Data Model, R1-1-12 requires that the OO data model must support large objects needed for multimedia support.

Proposed Architecture. This is not directly met because Open OODB has no specific functionalities to deal with BLOBs.

Implementation. This is currently met with limitations. It is met because Open OODB is an extension to C++ which supports pointers of type *void* and *char* to arrays of arbitrary length. One limitation is that Open OODB adds nothing to C++ to specially handle BLOBs. Another is that actual storage in Open OODB is currently handled by Exodus.

Open OODB feels that adding functionalities to handle BLOBs can be done without too much difficulty [69]. For example, another OODB that is an extension to C++, ObjectStore, allows for image data that "that can be stored in very large arrays that span many pages"[48]. ORION uses a *descriptor* object to describe the disk location of the associated long data [43]. BLOBs can be treated in a manner similar to files, with a label similar to a file descriptor [85]. Starburst handles what they call "Long Fields" of up to 1.5 gigabytes with a Long Field Manager with pointers to large pieces of the Long Field [49]. In this way, parts of the BLOB, or Long Field, can be referenced. For example, when dealing with photographic images of faces, each face can be broken down into eyes, the nose, etc., allowing access to pieces of the face, or Long Field. Postgres [86] also provides file oriented access to large objects. In [28], it is proposed to extend SQL to handle BLOBS with special statements such as CREATE< BLOB >, etc.

It must be noted that even though OODBs have been designed with the purpose of manipulating large objects common in multimedia applications in mind, many of the problems of manipulating such data are the same as they would be in other DBMSs. For instance, mechanisms must be devised to allow simultaneous access to different parts of the large object, or BLOB. Also; mechanisms must be devised to allow updates to only part of the BLOB without paging in the whole BLOB and mechanisms to handle the logging necessary for recovery must not necessitate logging the whole BLOB. OODBs appear to offer no advantages over other data models for handling such mechanisms.

4.2.5 Expressiveness of DML

The NGCR DBMS interface standards shall include a DML that allows users to easily isolate various subsets of the data held in a database. In particular, the DML shall provide for data representing multiple tables

(or record types or object types) to be meaningfully joined in a retrieval transaction. The join operation may be expressed declaratively as in relational database systems, or navigationally as in network and some object-oriented database systems.

Open OODB

Requirements. Open OODB has no matching requirements. However, this requirement is implicitly matched as covered below.

Proposed Architecture. The Open OODB architecture allows this DISWG requirement to be met. A motivating factor behind OODBs is the superior modeling capability of the OO paradigm. Open OODB's DML is C++ which is highly expressive.

Implementation. This is met. The DML is C++, which is expressive. Any OODB implemented as an extension to a programming language will have an expressive DML. ObjectStore lists one of its strengths as the expressiveness of C++, its DML, as compared to the capabilities of a language such as SQL [48].

DISWG specifically mentions joins in this requirement. OODBs can perform joins and may significantly reduce the number of joins needed in applications. This is accomplished by making one class an attribute of another, which is "in essence a static specification of a join ..."[46]. In other data models, such as the relational, one table can not be an attribute of another table. However, OODBs do not eliminate the need for joins. Joins are still needed to compare two classes that are disjoint. For instance, we may want to see how many employees are older than the companies for which they work [46].

4.2.6 Planned Queries

The NGCR DBMS interface standards shall provide support for planned queries.

A planned query is a query "for which the need was foreseen"[91]. For instance, the database may be based on a design that anticipates particular queries.

Open OODB

Requirements. Open OODB has no matching requirement.

Proposed Architecture. Open OODB's architecture meets this requirement although allowing for planned queries is not mentioned in the literature.

Implementation. This is met. Open OODB's schemas are in C++ which is highly expressive. Classes and class hierarchies could be set up in order to handle certain queries efficiently. For instance, if it is known that two classes are to be frequently used in conjunction, it may be better to set up the two classes in the same hierarchy as opposed to in different hierarchies. This would help because it should be more efficient to deal with one hierarchy than to join multiple hierarchies.

4.2.7 Ad Hoc Queries

The NGCR DBMS interface standards shall provide support for ad hoc queries.

DISWG defines ad hoc queries as the opposite of planned queries, i.e., queries for which the need was *not* foreseen.

Open OODB

Requirements. There is no Open OODB requirement that states that an OODB must support ad hoc queries in the way that DISWG defines ad hoc queries. However, under R6 Query Capability, R6-13-2 says an OODB must support interactive, ad hoc queries. Open OODB seems to use the terms interactive and ad hoc interchangeably, whereas in DISWG, there is a clear distinction as is seen in the discussion on the next DISWG requirement: interactive queries.

Proposed Architecture. Open OODB's architecture supports ad hoc queries. As just mentioned, in Open OODB's documentation, ad hoc is used synonymously with interactive [59].

Implementation. This is met. Under Open OODB's definition of ad hoc queries as being the same as interactive queries, all Open OODB queries are ad hoc. Under DISWG's definition of ad hoc being the opposite of planned, Open OODB, and perhaps every worthwhile database, meets this requirement.

4.2.8 Interactive Queries

The NGCR DBMS interface standards shall provide support for interactive queries

An interactive query is defined by DISWG as a "query language statement issued as a command by a user"[91].

Open OODB

Requirements. This is matched. Under R6 Query Capability, R6-13-2 states that an OODB must support interactive, ad hoc queries.

Proposed Architecture. The Open OODB architecture meets this DISWG requirement. Open OODB states that "our implementation does not preclude an interactive (ad hoc) query capability"[59].

Implementation. This is met. We assume that the term *interactive* query as used by DISWG is an antonym for canned query, although this is not explicitly stated.

4.2.9 Embedded Queries

> *The NGCR DBMS interface standards shall provide support for embedded queries. Programming languages in which the embedding can take place include at least Ada, FORTRAN, C, and C++.*

Open OODB

Requirements. This is matched under Open OODB's R6 Query Capability functional requirement. R6-2-1 says that OODBs must support an SQL-like language that can be integrated with a host programming language. Also, under R14 Seamlessness, R14-9 requires support for seamless queries.

Proposed Architecture. The current architecture meets this DISWG requirement.

Implementation. This is met as Open OODB supports an SQL-like language embedded in C++ [62, 81]. As a matter of fact, Open OODB currently supports *only* embedded queries [63]. This requirement again points out the main difference between DISWG and Open OODB. DISWG has to allow for many languages, but Open OODB has had to pick a specific language(s) for implementation. Studies have been done on porting Open OODB to Ada with favorable results, and Open OODB's modules have been designed to be language independent. Thus, Open OODB could be

implemented in more languages.

4.2.10 Compiled Queries

The NGCR DBMS interface standards shall provide support for compiled queries.

A compiled query is "a query language statement that is compiled, or translated, into executable code prior to run-time.

Open OODB

Requirements. There is no matching Open OODB requirement.

Proposed Architecture. The Open OODB architecture supports compiled queries.

Implementation. This is met because all Open OODB queries are compiled.

4.2.11 Interpreted Queries

The NGCR DBMS interface standards shall provide support for interpreted queries

DISWG defines interpreted queries as the opposite of compiled queries.

Open OODB

Requirements. There is no matching Open OODB requirement.

Proposed Architecture. Open OODB currently does not meet this requirement. Open OODB and others are working on an interpreted C++, a necessity for the support of interpreted queries.

Implementation. This is not met because C++ can not currently be interpreted. Interpreted queries are considered more flexible, but slower, than compiled queries. The flexibility is due to the possibility of making run time additions [33]. The slowness is due to the time needed to interpret each command individually. However, in [36], it is pointed out that the compile time should be considered as well as the run time.

4.2.12 Transactions

The NGCR DBMS interface standards shall support transactions with conventional ACID properties. They shall include statements for initiating transactions and committing or aborting them.

Open OODB

Requirements. This is matched. Under R3 Concurrent Access, R3-2 says that OODBs must also support some of the following: traditional short transactions, nested transactions, optimistic transactions, multi-threaded transactions, long duration transactions, parallel transactions, compensating transactions, and cooperative transactions. Open OODB allows for the relaxation of the ACID properties. This is evidenced by the cooperative transactions mentioned above, which directly conflicts with ACID's Isolation component [59].

Proposed Architecture. This is met. In addition, the traditional ACID properties may be relaxed. The architecture calls for a Transaction PM that "enables concurrent access to persistent and transient data and supports recovery of changes on these data in the presence of failure"[59].

Implementation. This is met. Manipulations of the persistent store must be done from within transactions which are delineated with familiar looking begin, commit and abort transaction statements. However, the Transaction PM is not currently im-

plemented in Open OODB. This is because Exodus, the underlying storage manager, implements its own transaction control mechanisms and those mechanisms are deeply embedded in Exodus.

Exodus's transaction control does meet this DISWG requirement, which is why we can conclude that Open OODB meets it. Exodus uses an ARIES-based recovery mechanism to help support ACID's Atomicity component. ACID's Consistency component, while ultimately the responsibility of the programmer, is supported with a strict locking mechanism. Transactions in Exodus operate in isolation, supporting ACID's Isolation component. Finally, Exodus provides persistent storage, as well as the ARIES-based recovery scheme, which supports ACID's Durability component.

Both concurrency control and recovery are usually associated with transactions, but are not mentioned in this DISWG requirement. In Open OODB, Exodus handles both. Exodus applies a single concurrency control mechanism to all objects. Objects "are read locked upon read, and the lock is promoted to a write lock if an attempt is made to write the object back"[63]. For recovery, Open OODB relies on Exodus's ARIES [51] based algorithm.

4.2.13 Data Models

The NGCR DBMS interface standards shall provide DML(s), DDL(s), and DCL(s) that support conventional data models, i.e., relational and network.

Open OODB

Requirements. There is no directly matching requirement. However, under the R11 Access to Legacy Data functional requirement, R11-1 says that an OODB "must be able to access and update existing info in conventional relational databases (RDB) as objects"[69].

Proposed Architecture. Open OODB does not currently meet this DISWG requirement. However, Open OODB envisions that, due to its modular, extensible design, its services could be combined to provide relational databases, etc. [58].

Implementation. This is unmet. However, providing such support is an active research area for the Open OODB project. Nothing in Open OODB's design would preclude this requirement from being met. The OO paradigm, with its encapsulated data and method access, seems well suited to provide support for conventional data models.

4.2.14 Conceptual Schema Definition

The NGCR DBMS interface standards shall provide DDL statements for defining and maintaining conceptual schemas.

DISWG defines a conceptual schema as "a description of the conceptual or logical data structures and the relationships among those structures"[32].

Open OODB

Requirements. This DISWG requirement is not directly matched by an Open OODB requirement. However, under R5 Data Dictionary, R5-5-1 states that the Data Dictionary must be able to represent data model information. Included in this information are class/type/schema definitions, type lattice and behavior, etc.

Proposed Architecture. The Open OODB architecture supports meeting this requirement. C++ is used as the DDL and it is very natural to represent schemas in the OO paradigm.

Implementation. This is partially met. The DDL is C++. However, Open OODB makes no mention of relationships between objects as does DISWG in its definition of

conceptual schema. In URI's RTSORAC model, relationships[21] between objects are themselves objects, and can thus be treated similar to any other object [73]. Open OODB could be extended to include relationships such as those in RTSORAC, which would allow this DISWG requirement to be met even more stringently.

4.2.15 External Schema Definition

> *The NGCR DBMS interface standards shall provide DDL statements for defining and maintaining external schemas.*

DISWG states that an external schema and a *view* are equivalent and mean a description of a subset of the database.

Open OODB

Requirements. There is a matching requirement. Under the R1 OO Data Model requirement, subrequirement R1-1-13 states that an OODB may support views. Also, under R6 Query Capability, R6-14 requires support for incrementally updated views.

Proposed Architecture. Open OODB does not currently meet this requirement.

Implementation. This is not met. Indeed, in [46], it is observed that "No OODB today supports views"[46]. Generally, a view is composed of fragments of one or more objects. Thus, a view is a new, previously undeclared object which needs an identifier, and this creates problems for OODBs [7]. This is complicated by the fact that, ideally, views should be updatable. One way to deal with the problem is presented in [3] which discusses views in the O_2 system. They liken views to importing data from a remote database. They use a method that creates virtual classes and imaginary objects that exist only in the view.

[21] Relationships in RTSORAC are discussed in the section on research at URI.

The problem of views in an OODB appears similar to the schema evolution problem. A view creates a new object type and schema evolution can do the same thing. Also, since views are closely tied to queries, the development of an Object SQL would facilitate matters.

4.2.16 Internal Schema Definition

> *The NGCR DBMS interface standards shall provide DDL statements for defining and maintaining internal schemas.*

The internal schema is a description of how data is actually organized in storage. It includes information on "the ordering and size of records, and available access methods (links, indexes)"[32].

Open OODB

Requirements. Open OODB does not have a matching requirement. However, under R5 Data Dictionary, R5-1-4 states that the Data Dictionary must be responsible for physical representation information such as alignment, offsets and formats.

Proposed Architecture. This is partially met. The DDL is C++ which can specify some orderings in main memory (i.e., as with arrays). However, C++'s ability in this area is limited.

Implementation. This is partially met. As noted above, C++ has some ability to order main memory. However, persistent storage is handled by the underlying Exodus storage manager, and as such, is outside of the control of Open OODB's DDL. The designers of the C++ based ObjectStore note that "Applications can improve performance by exercising control over the placement of objects within a database"[48]. ObjectStore allows for application defined clustering of objects that

are frequently used together. ORION supports only a simple clustering scheme where "instances belonging to a user-specified collection of classes are stored in the same physical segment"[43]. "Unfortunately, however, clustering is more of an art than a science at this time"[21].

4.2.17 Identification and Authentication

The NGCR DBMS interface standards shall provide a mechanism for identifying and authenticating users.

Open OODB

Requirements This is partially matched. Under Open OODB's R10 Security functional requirement, R10-2 specifies that an OODB must support authorization. Other subrequirements of R10 hint at identification, but do not explicitly require it.

Proposed Architecture. Open OODB does not currently meet this DISWG requirement. Security is outside the scope of the current effort. Open OODB foresees difficulty in adding security features to its functionalities [69], but envisions security as a dimensional extension to the system architecture [68]. Since DISWG has a separate Security requirements class, further discussion on the topic is deferred until the evaluation of that class.

Implementation. This is not met. Security issues are covered in the evaluation of DISWG's Security requirements class.

4.2.18 Discretionary Access Control

The NGCR DBMS interface standards shall support discretionary access control. The owner of a database object shall be able to specify which

users are authorized to perform which operations on an object. Discretionary access control can, for example, be managed through the definition of views and the granting/revoking of privileges. The Standard shall enable a user to specify access control at various levels of granularity (e.g., in a relational database, at the level of tables or various subset of tables, such as columns or rows).

Open OODB

Requirements. As noted previously, Open OODB requirement R10 deals with security issues. The R10 requirements are not well developed as of yet, and their general nature allow most security requirements to be matched. For instance, R10-6 requires an OODB to support "various sorts of security"[69].

Proposed Architecture. Open OODB does not meet this, or any security requirements, because security falls outside the scope of the current effort. For further information, refer to the evaluation of DISWG's Security requirements class.

Implementation. This is not met. Security is discussed in the section on DISWG's Security requirements class.

4.2.19 Access to Metadata

The NGCR DBMS interface standards shall enable a DBMS to maintain the integrity of a database by disallowing operations, such as certain updates to metadata, that could corrupt the system.

Open OODB

Requirements. There is no directly matching requirement. R5 does require that "an OODB must be able to store, access, and manipulate meta-data"[69], but no

mention is made of maintaining the integrity of the meta data. Also, under R14 Seamlessness, R14-10 requires that "Meta-data ... are data and can be queried, change managed, concurrency controlled, using ordinary mechanisms"[69].

Proposed Architecture. The proposed architecture does not meet this requirement. Although a Data Dictionary module is mentioned, no provision is made for maintaining its integrity with respect to meta data access.

Implementation. This is unmet. Most OODBs do not offer support for meta data management [46]. As will be noted in the discussion of DISWG's Integrity requirements class, although integrity maintenance is not currently included in Open OODB, the basic nature of the OO paradigm should facilitate its future inclusion.

4.2.20 Multiple DBMSs

The NGCR DBMS interface standards shall not preclude the presence of multiple DBMS implementations on the same computer system.

Open OODB

Requirements. There is no matching Open OODB requirement. Open OODB itself is a DBMS, so this again points out the difference between DISWG and Open OODB that has been noted throughout this thesis. DISWG presents requirements on interface standards and Open OODB is an implementation.

Proposed Architecture. Open OODB's proposed architecture meets this requirement simply because nothing precludes it from being met. As stated above, since Open OODB is itself a DBMS, this requirement does not really apply.

Implementation. This is met. Nothing in Open OODB's implementation would prevent it from co-existing with other DBMSs on the same computer system.

4.2.21 Multiple Databases

The NGCR DBMS interface standards shall enable multiple databases, possibly sharing a DBMS, to be implemented on the same computer system (e.g., via a "create database" command).

Open OODB

Requirements. There is no directly matching Open OODB requirement. However, under R4 Distribution, R4-3 requires that "applications can access data stored on multiple object servers"[69].

Proposed Architecture. This is met. Multiple databases could be handled by using multiple ASMs and possibly the services of the Distribution PM, etc. For instance, each database could be considered to be a separate address space with its own ASM. If there are multiple databases distributed on different nodes in the system, then the Distribution PM would be needed.

Implementation. This is met. Currently, Open OODB is the DBMS for Exodus and could be the same for other databases.

4.2.22 Tracing

The NGCR DBMS interface standards shall provide a capability for user-readable error and transaction execution tracing.

Open OODB

Requirements. This is not directly matched. However, under R15 Performance, subrequirement R15-3 states that an OODB should support usage metering. Usage metering is "defined as the kinds of feedback the system should be able to provide"[69].

Proposed Architecture. The Open OODB architecture does not currently meet this requirement. There is no mention of the architecture providing for such capabilities.

Implementation. This is unmet. Nothing would preclude it from being met. However, Open OODB mentions how hard error checking is when using the transparent extension approach. The same problems would probably affect tracing.

4.2.23 Statistical Monitoring

The NGCR DBMS interface standards shall provide the capability to enable and disable statistical monitoring of database usage.

Open OODB

Requirements. Open OODB's R15 Performance meta requirement matches this DISWG requirement. R15 states that "an oodb must provide usable performance and the ability to configure, tune, measure, profile performance"[69]. Specifically, subrequirement R15-3 states that an OODB must support usage metering. Also, under R13 Openness, subrequirement R13-11 states that "it should be possible to install performance meters at module interfaces"[69].

Proposed Architecture. The Open OODB architecture does not currently meet this DISWG requirement.

Implementation. This is not met. Open OODB's modularity and well defined interfaces should allow for for the incorporation of statistical monitoring facilities. Open OODB recognizes the need for such facilities, but is not clear on how to best implement them [69].

4.2.24 Training Mode

The NGCR DBMS interface standards shall support a training mode of operation in which users can exercise the DBMS without damaging the integrity or operational capability of the system.

Open OODB

Requirements. There is no matching Open OODB requirement. One would think think this would be a subrequirement under the R12 User Interface requirements.

Proposed Architecture. This DISWG requirement is not met by Open OODB's architecture.

Implementation. This is not met. The fact that there is not an implemented user interface would certainly hinder any training mode. However, once a user-friendly user interface is devised, nothing in Open OODB's design would preclude a training mode.

4.3 DISWG's Distribution Requirements Class

"The NGCR DBMS interface standards shall support distributed database systems where a distributed database system is defined as [below]"[32]. It is important to note that the requirements in this class are on distributed database *systems*, not on distributed databases. "The Navy and other armed services have large numbers of computer systems interconnected by various local-area and wide-area networks. Effective organization and management of data distributed across these interconnected computer systems is crucial"[32]. In fact, distribution is one of the most important aspects of next-generation computer systems [12, 71]. In the rest of this subsection, we first provide some of DISWG's definitions as they relate to distribution. We next

provide some alternative definitions. We then discuss the benefits of the OO approach and follow with a general discussion of how Open OODB fits in with these requirements. Finally, we perform the evaluations.

4.3.1 Definitions

DISWG cites the following: "A *distributed database* is a collection of data distributed over different computers of a computer network. Each site of the network has autonomous processing capability and can perform local applications. Each site also participates in the execution of at least one global application which requires accessing data at several sites using a communications subsystem"[32]. DISWG states that a distributed DBMS (DDBMS) "permits the management of a distributed database" that does "not accommodate heterogeneity or autonomy." Finally, DISWG defines a distributed database system (DDBS) as "a combination of the distributed database and distributed DBMS." Thus, a DDBS is a collection of autonomous databases which can be accessed through a distributed DBMS. Finally, according to DISWG, "a distributed database system managed by xyz DDBMS would have data at multiple nodes on a computer network, and each node would be running a copy of xyz; this distributed database system would appear to the end user as a single logical database system"[32].

DISWG contrasts a DDBS with a federated database system. In a DDBS, homogeneity is implied. "The DDBS supports just one data model and query language, with an unambiguous schema"[32]. Each node in a DDBS runs a copy of the same distributed DBMS. In a federated database system, heterogeneity is implied. The component databases may vary with respect to data model, query language, schema and DBMS, etc.. Federated database systems are looked at more closely in the subsequent Heterogeneity DISWG requirements class.

Despite DISWG's definitions, "There is no standard definition for a DDBMS"[89]

and there is no standard definition of a DDBS [13]. Alternative definitions vary subtly and drastically. In [10], a distributed DBMS is defined as a collection of sites running one or both of a transaction manager and a data manager. The designers of ORION-2 define their system as a "homogeneous distributed object-oriented system that allows each user a single-system image of the entire, shared database and of the user's own, private database"[45]. In [31], it is observed that perhaps its best to use the term distributed database "in a more general sense to mean a collection of possibly independent or federated database systems." In [26], "A **distributed database** is a collection of data that belongs logically to the same system but is physically spread over the sites of a computer network." Also in [26], DDBSs can be categorized by the degree of homogeneity and local autonomy. One thing is clear: the definition of distribution in the context of databases needs refinement.

4.3.2 OODBs and Distribution

Most "applications that require OODBMS technologies typically arise in distributed environments" and the basic characteristics of the OO paradigm are expected to be of significant assistance with distribution [71]. These helpful OO characteristics include: advanced modeling power [78], message passing and encapsulation, and the extensible nature of OODBs [25]. Encapsulation means that an object's data is protected by a well-defined interface. With object encapsulation, it does not really matter where a computation takes place [65]. Also, inheritance could be used to take advantage of the similarities in the distributed components. However, the designers of GemStone feel that "Distribution is perhaps the most difficult" aspect "of a database management system with regard to development and successful deployment"[13].

A distributed system's components could be thought of as objects or collections of objects [54, 78]. In [71], it is proposed to view the distributed sites as fragments of a composite object. It must be pointed out that both of the just cited methods,

using collections and fragments, are applicable to federated databases as well. This fits in with the way that DISWG defines DDBSs and federations as differing mainly in the degree of heterogeneity.

4.3.3 Open OODB and Distribution

Open OODB is not designed to be a distributed database system according to the DISWG definition. However, Open OODB is designed to be a distributed database. This capability will certainly help to extend Open OODB to be constructed as a DDBS in the DISWG sense. Indeed, allowing multiple copies of Open OODB to participate in a DISWG DDBS is an active research area at TI. The modular, extensible nature of Open OODB and the fact that multiple copies of modules can coexist are plusses in this area. Open OODB envisions an OSA-like architecture with a CORBA-like [57] backplane and multiple Open OODB implementations connected to the backplane.

Thus, since Open OODB does not currently meet these DISWG requirements, the evaluations of the individual requirements in this class are very similar in format. For example, we conclude that Open OODB's proposed architecture and implementation do not meet the requirement, but nothing precludes extending Open OODB to meet the requirement. Therefore, for argument's sake, we discuss how Open OODB would meet DISWG's requirements if those requirements were on a distributed database instead of a distributed database *system*. For instance, the first requirement in this class is on support for queries in a distributed database system. Open OODB does not match that requirement because Open OODB is not a distributed database system. However, Open OODB has a requirement for support of distributed queries in a distributed database. This approach is justified because Open OODB's abilities as a distributed database will enhance extending it to be a DISWG DDBS.

Open OODB's Distribution PM is described on page 21 in the section on Open OODB's extenders. Although the Distribution PM is currently unimplemented [63],

Open OODB is, nonetheless, partially distributed. The Exodus storage manager may be at a different site than Open OODB. Thus, Open OODB and Exodus can communicate over a network, a key component of distribution. Also, the system could accommodate multiple storage managers at different sites. Thus, Open OODB now provides rudimentary distribution. However, if Open OODB's proposed architecture were fully implemented, Open OODB would be fully distributed in the way DISWG defines a distributed database. That is, Open OODB could manipulate multiple, homogeneous databases at multiple sites. This capability should allow Open OODB to be extended to be a DDBS. We now evaluate the eleven requirements of this class.

4.3.4 Distributed Query Processing

> *The NGCR DBMS interface standards shall provide the capability for end users and application programmers to issue queries that access data stored at multiple computer systems.*

Requirements. There are no matching requirements. However, Open OODB's R4 Distribution requirement has a subrequirement, R4-9, that states an OODB must support distributed queries. Also, under the R6 Query requirement, subrequirement R6-12 states that an OODB must support queries over distributed data.

Proposed Architecture. This requirement is unmet because Open OODB's proposed architecture does not provide support for a distributed database system. Since queries exist inside of transactions and distributed transaction support is the next DISWG requirement, a discussion of how the architecture relates to this requirement is deferred until then.

Implementation. This is unmet by the implementation. Open OODB implementations have not been designed to talk to each other. Also, all data in the database

resides at the same site as determined by the Exodus storage manager. However, if Open OODB were fully implemented, then the implementation would support distributed queries as managed by a single implementation of Open OODB. The query capability of Open OODB has been designed to be "independent of the kinds of extensions associated with the objects being queried. This means that it should be possible to query objects independent of whether they are ... local or remote ..."[59]. Therefore, Open OODB could be extended to meet this requirement in the manner discussed in the introduction to these requirements. That is, an implementation of Open OODB could query other implementations of Open OODB across a CORBA-like backplane.

4.3.5 Distribution Transaction Management

The NGCR DBMS interface standards shall provide the capability for end users and application programmers to issue transactions that access data stored at multiple computer systems. Concurrency control and recovery control shall be applied in such a way that the ACID properties of transactions are maintained despite the distribution of data.

Open OODB

Requirements. There are no matching requirements because Open OODB is not a distributed database system. However, subrequirements under the Open OODB's R4 Distribution requirement state that an OODB must support distributed transaction management in a distributed database. For instance, R4-3 states that "applications can access data stored on multiple object servers", R4-2 states that an OODB must allow "object transfer between workstations", R4-5 states that "arguments to messages may be local or remote", R4-6 requires "local and remote invocation of methods" and, finally, R4-8 states that distributed commits must be supported [69].

Proposed Architecture. The proposed architecture does not meet this requirement. However, Open OODB has been designed with support for distributed transactions in a distributed database in mind. There may be multiple Transaction PMs running at any given time. Each transaction, T_i is the responsibility of one Transaction PM, in this case TPM_i. It is the responsibility of TPM_i to make all needed objects available to T_i. If the needed objects are distributed, TPM_i would use the services of the appropriate ASMs and the Distribution PM, etc., to make the appropriate data available.

Implementation. The implementation does not meet this requirement. Also, all data is at one site as determined by Exodus. However, if the appropriate modules were implemented, Open OODB would support distributed transaction management in a distributed database and Open OODB could be extended to meet this requirement. This DISWG requirement also mentions concurrency control and recovery. As noted in the evaluation of the Transaction requirement in DISWG's Basic requirements class, both concurrency control and recovery are handled by Exodus. The flexibility of Open OODB's design certainly allows for this to be changed in the future.

4.3.6 Location Transparency

> *The NGCR DBMS interface standards shall enable the data stored in the distributed database system to be located at multiple interconnected computer systems. To the end user and the application programmer, the distribution of the data is transparent. That is, distributed queries and transactions can be formulated as if the data were not distributed; they have no dependence on the locations of the data that they reference.*

Open OODB

Requirements. There are no matching requirements. However, there are matching requirements in the context of a distributed database under the R14 Seamlessness meta requirement. Specifically, R14-6 requires support for location transparency, which "hides the mapping of a fragment to a particular object store"[69]. From a high-level point of view, requiring Open OODB to be seamless *implies* that location transparency is required.

Proposed Architecture. This is unmet by the proposed architecture. However, in the context of a distributed database, Open OODB's architecture supports location transparency. Open OODB is designed to be a *transparent* extension to C++. Thus, the ASMs and Distribution PM, etc., would be invoked transparently and their actions would take place unbeknownst to the end-user.

Implementation. The implementation does not meet this requirement. Also, all the data is at one site. Implementation of the appropriate modules would allow Open OODB to support location transparency in a distributed database and this functionality could be extended to meet this DISWG requirement.

4.3.7 Fragmentation Transparency

> *The NGCR DBMS interface standards shall enable data in a distributed database system to be partitioned across multiple, interconnected computer systems. For example, in relational DDBMSs, the fragmentation can be horizontal or vertical. Horizontal fragmentation occurs when the rows of a table are distributed across multiple sites, and vertical fragmentation occurs when the columns are distributed. To the end user and the application programmer, fragmentation is transparent. Distributed queries and transactions can be formulated as if data were not fragmented.*

Open OODB

Requirements. This is unmatched. However, under R14 Seamlessness, R14-5 requires fragmentation transparency in the context of a distributed database. Also, R14-5 states that the details of the distribution of object components must be hidden.

Proposed Architecture. The proposed architecture does not meet this requirement. However, in the context of a distributed database, it would. All activities as relates to where and how an object is stored are done transparently without user knowledge.

Implementation. This is not met. Also, all data is at one site as determined by Exodus. However, the fact that Open OODB is designed to be a *transparent* extension to C++ would allow Open OODB to meet this requirement if it were on a distributed database. Open OODB states that "A fragmented object is one that logically exists in several address spaces and is physically composed of a collection of object fragments connected by a connective object that acts as a communication channel. ... The user sees the illusion of a single object"[64]. Thus, Open OODB is being designed with the ability to handle fragmented objects in mind and could be extended to meet this requirement.

4.3.8 Replication Transparency

> *The NGCR DBMS interface standards shall enable the data in the distributed database system to be replicated at the fragment level. For example, in the relational model, this means that tables, as well as horizontal and vertical fragments of tables, can be replicated. To the end user and application programmer, the replication is transparent; distributed queries and transactions can be formulated as if the data were not replicated.*

Open OODB

Requirements. There are no matching requirements. However, in the context of distributed databases, it is matched. Under R14 Seamlessness, R14-7 requires that an OODB must support replication transparency, which means that management of replicas is hidden from the application. Also, under R4 Distribution, R4-10 requires an OODB to support replicated data for fault tolerance and availability. In a discussion on Open OODB's objectives, it is stated that seamlessness adds functionalities such as replication transparency [58].

Proposed Architecture. This is unmet by the current architecture. However, from a distributed database point of view, replication has been considered by Open OODB. Replication is not included in the alpha release of Open OODB, but is an active research area. Open OODB considers replication to be a language extension worthy of being a separate PM [65]. Replication, or multiple instantiation, can improve performance by increasing availability and can enhance fault tolerance capabilities. Indeed, in literature outside of the alpha release, Open OODB includes a Replication PM responsible for keeping track of replicas and making the replicas transparent to the user [66]. In this literature, a reference to a replicated object is trapped by a sentry that passes control to the Replication PM. The Replication PM is responsible for maintaining consistency among replicated objects. OMG also sees replication as a well-defined service module.

Implementation. This is unmet. As just noted, replication in a distributed database is an active research area for Open OODB and the intent is to have a Replication PM. Open OODB is being designed with support for fragmented objects in mind and these fragmented objects could be replicated. Also, the Replication PM would allow replication at the object level, which is a grain of fragmentation with respect to sets, etc. One of the problems in replica management is how to maintain consistency

among replicated objects. This is exacerbated by the fact that all objects in Open OODB have unique OIDs, which makes checks for equality more difficult [66]. This problem helps to justify a separate Replication PM. A fully implemented Replication PM would allow Open OODB to be extended to meet this requirement.

4.3.9 Data Definition

> *The NGCR DBMS interface standards shall provide data definition facilities used to control the distribution, fragmentation, and replication of the data in a distributed database system. By exercising such data definition facilities, the DBA can design a distributed database that meets the reliability, availability, and performance requirements of the application.*

Open OODB

Requirements. There are no matching requirements; Open OODB is not a distributed database system. However, Open OODB's R5 Data Dictionary requirement states that "an OODB must be able to store, access, and manipulate meta-data"[69]. The information required includes: R5-1-8 environment information, R5-1-5 location information, R5-1-9 replication information and R5-1-4 physical representation information"[69].

Proposed Architecture. This is not met. However, in a distributed database implementation of Open OODB, the data dictionary would contain information needed by the Distribution and Replication PMs, etc.

Implementation. This is not currently met because Open OODB is not a distributed database system. Also, the Data Dictionary is only partially implemented and the Distribution and Replication PMs are unimplemented. If Open OODB were

implemented as designed, this would be met in the context of a distributed database and Open OODB could be extended to meet this requirement.

4.3.10 Local Autonomous Processing Capability

The NGCR DBMS interface standards shall enable the local database system to autonomously execute local applications (programs or interactive queries or transactions), i.e., applications that reference only local data. (The data at each local computer system constitutes a local database. Together, the local database and the local instantiation of the DDBMS constitute a local database system.)

Open OODB

Requirements. There are no matching requirements. This requirement highlights the difference between DISWG and Open OODB. DISWG is a set of requirements on interfaces and must address such issues as different databases and their DBMSs cooperating. Open OODB is not designed to be a distributed database system, but rather, a DBMS in and of itself. Thus, local autonomous processing capability is implicitly required in Open OODB.

Proposed Architecture. This is unmet. Open OODB is a DBMS and has not been designed to be a distributed database system. This would imply that there are a number of sites all running their own copy of Open OODB. Together, all the different sites with their own copy of Open OODB would have to be able to appear as one to an end-user. This type of functionality is not covered by the proposed architecture.

Implementation. This is unmet. DISWG has to worry about DBMSs communicating with each other whereas Open OODB is a DBMS. However, since Open OODB resides at one node, it certainly has local autonomous processing capability. Also,

the Exodus storage manager can act as an autonomous DBMS. Finally, as already noted, Open OODB is actively working towards development of a DDBS using a CORBA-like architecture which would allow this requirement to be met.

4.3.11 Continuous Operation

The NGCR DBMS interface standards shall not preclude the continuous operation of the database system.

Open OODB

Requirements. There is no direct match. However, the R16 Industrial Strength meta requirement would seem to require continuous operation of a distributed database.

Proposed Architecture. This is unmet because the requirement is on a DDBS and Open OODB is not a DDBS. In the context of a distributed database, this is met because of the wording of the requirement. Nothing in Open OODB would *preclude* continuous operation. However, there are no special functionalities included in the proposed architecture to ensure continuous operation.

Implementation. This is not met. Also, as a distributed database, Open OODB provides no explicit way to delete data. Thus, the system must be flushed periodically to provide more space as needed. During this time, the database would be unavailable. This should be an easy problem to correct. It is interesting to note that OMG's OSA includes an Object Lifecycle module to explicitly cover things such as object creation and deletion [57]. Finally, in an early release of GemStone, which is a C++ based distributed database, the system had to be stopped to perform backup [13].

4.3.12 Hardware Independent

The NGCR DBMS interface standards shall be hardware independent. They shall allow data to be distributed across heterogeneous computer systems.

Open OODB

Requirements. This is unmatched. However, hardware independence in a distributed database is mentioned in the R4 Distribution requirements. Specifically, R4-3 states that an OODB must allow applications access to data on multiple object servers and R4-3-2 that the servers can be heterogeneous. Also, R4-4 requires a computing environment that is R4-4-2 heterogeneous with respect to R4-4-2-1 machines. However, we again point out that Open OODB is not designed to be what DISWG defines as a distributed database system.

Proposed Architecture. This is not met. However, Open OODB's modular nature and its ability to deal with different types of address spaces and data formats account for hardware differences. Also, the Translation module in Open OODB is designed to translate objects between hardware platforms [70].

Implementation. This is unmet. Refer to the evaluation of the hardware independence requirement in DISWG's General requirements class on page 38 for more details.

4.3.13 Operating System Independent

The NGCR DBMS interface standards shall be operating system independent. They shall allow data to be distributed across heterogeneous operating systems.

Open OODB

Requirements. This is not matched. However, under R4 Distribution, R4-4-2-2 that states that an OODB must support a computing environment that is heterogeneous with respect to operating systems. Also, under R16 Industrial Strength, R16-5 requires that an OODB must be portable and R16-5-2 that it must be O/S (operating system) independent.

Proposed Architecture. This is unmet. However, Open OODB has been constructed with operating system independence in mind, which would allow it to be ported to different systems. Open OODB's open, extensible and modular nature would facilitate any ports.

Implementation. This is unmet by the current implementation. Open OODB is tied to UNIX [69]. However, being tied to UNIX is also a strength due to its wide acceptance. For example, researchers at URI are interested in porting Open OODB to a POSIX-compliant operating system to take advantage of the real-time properties of such an operating system. The fact that POSIX is based upon UNIX should facilitate matters.

4.3.14 Network Independent

> *The NGCR DBMS interface standards shall be network independent. They shall not require computer systems holding the distributed data to be interconnected by any specific communications network.*

Open OODB

Requirements. This is not directly matched by any requirement. Refer to the evaluation of DISWG's General requirement for network independence on page 39.

Proposed Architecture. This is not met by the proposed architecture. Open OODB is not a DDBS. The proposed architecture's network independence capabilities are covered on page 40 in the evaluation of DISWG's General requirement for network independence.

Implementation. This is unmet. However, fully implemented Distribution PM, Communications and Translation modules, etc., would allow this DISWG requirement to be met in the context of a distributed database. These functionalities could be extended to meet this DISWG requirement.

4.4 DISWG's Heterogeneity Requirements Class

"The NGCR DBMS interface standards shall provide facilities to promote the integration and interoperability of distributed, heterogeneous, autonomous database systems"[32]. In this section, we first list some definitions. Then, we motivate why this is an important issue. Next, we introduce some of the obvious problems and proposed solution strategies. We follow with a discussion of how the OO approach may help solve some of the problems. Finally, we evaluate the requirements.

4.4.1 Definitions

Heterogeneous databases "vary with respect to DBMS, data model, query language and/or data definition"[32]. Autonomous database systems are "under separate and independent control"[32]. Interoperability is the problem of making heterogeneous, distributed databases behave as if they formed part of a single database [84].

DISWG defines two ways that autonomous, heterogeneous database systems can be integrated and/or be made to interoperate. In a *multidatabase* system, "a multidatabase language is responsible for achieving interoperability of heterogeneous, autonomous database systems"[32]. In a *federated* system, "users can query using a classical query language against the federated schema with an illusion that he or she is

accessing a single system"[91]. As with distribution, definitions here vary. The terms multidatabase and federated database have been used as synonyms [12, 78]. In [13], federated distribution is defined as "cooperation among several logically independent databases."

4.4.2 Important Issues

Recently, interoperability among heterogeneous databases has been receiving more and more attention. The primary reason is the vast quantity of pre-existing data under the control of DBMSs and outside such control [9]. Also, the development of new data models, platforms and algorithms precludes the prospects of any type of database uniformity. This has caused the National Science Foundation (NSF) to conclude: "It is unreasonable to expect all disciplines to converge on some unifying standard for data model, data language, and communication; heterogeneity will continue to be a complicating factor"[29]. Thus, methodologies need to be developed to access heterogeneous databases and data in an efficient manner.

The attention given to interoperability among heterogeneous databases has caused many problems to become apparent. At the "heart of the interoperability issue" are naming problems [54]. How does one component reveal its naming conventions to another? The different components may use different concurrency control and locking mechanisms, making problems such as deadlock prevention difficult [24]. Varying recovery techniques could impact reliability. Real-time deadlines could be next to impossible. Security could pose some problems that are just unsolvable [89]. The list is extensive!

To illustrate the complexity of the heterogeneity problems, we take a closer look at the issue of local site autonomy. There appears to be a trade-off between local site autonomy and the degree of interoperability [31]. Consider a concurrency control problem. The most accepted way to implement concurrency control in a system of in-

shared with the outside world. Despite the drawbacks of the first approach, it has been more widely used because of the complexity of the second approach.

4.4.3 OODBs and Heterogeneity

The OO paradigm is expected to be of help in the development of heterogeneous systems. As noted in the introduction to DISWG's Distribution requirements class, heterogeneous database systems and DDBSs are very similar. Some would say that heterogeneous database systems are DDBSs with different components. Therefore, the arguments presented in evaluation of the Distribution class about the benefits of the OO paradigm apply here as well. Also applicable is Open OODB's vision of incorporating Open OODB in a CORBA-like system. The databases connected to the CORBA-like system may be heterogeneous.

The problems faced and the lack of solutions are illustrated by the fact that there are only four DISWG Heterogeneity requirements. This is directly related to the preliminary state of research towards solving these problems. With that in mind, we now evaluate the four Heterogeneity requirements. The evaluations of each of the requirements are similar. Open OODB is not a distributed, heterogeneous, autonomous database system. Thus, like the Distribution requirements, the requirements here do not apply to Open OODB. Therefore, none of these requirements are met by Open OODB's proposed architecture or implementation. All we can do is point out that work is underway at the Open OODB project to design a CORBA-like system hooked up to possibly heterogeneous databases.

4.4.4 Remote Database Access

The NGCR DBMS interface standards provide the capability for a user or application program to remotely access heterogeneous databases.

Open OODB

Requirements. This is matched Open OODB's R11 Access to Legacy Data functional requirement.

Proposed architecture. This is not met. This is an active research area for the Open OODB project. Open OODB's extensible design will be of help.

Implementation. This is not met. However, work is underway to provide such access. As already discussed, Open OODB envisions a CORBA-like system serving multiple databases. Those databases could be heterogeneous. The designers of GemStone [13] have developed relational "gateways" that allow the users of GemStone to query remote, relational databases.

4.4.5 Global Transactions

> *The NGCR DBMS interface standards shall provide the capability for a user to execute global transactions. In particular, the standard shall specify application program interfaces for communication between transaction managers and resource managers (i.e., DBMSs).*

Open OODB

Requirements. Open OODB has no matching requirement.

Proposed architecture. This is not met.

Implementation. This is not met. Successful implementation of the previously discussed CORBA-like system would allow this to be met.

4.4.6 Multidatabase Systems

The NGCR DBMS interface standards shall provide multibase language features that enable a user or application program integrated access to multiple, autonomous database systems.

Open OODB

Requirements. There are no matching requirements.

Proposed architecture. This is not met.

Implementation. This is not met. The Open OODB project is not taking the multidatabase approach. This is because the CORBA-like system would not use a multidatabase language that is characteristic of a multidatabase system.

4.4.7 Federated Database Systems

The NGCR DBMS interface standards shall provide facilities for establishing federated database systems.

Open OODB

Requirements. There are no matching requirements. R17 Other DBMS Requirements does mention heterogeneous DBMSs. However, R17 does not require support for heterogeneous DBMSs, it only says that heterogeneous DBMSs are outside the current scope of Open OODB.

Proposed architecture. This is not met.

Implementation. This is not met. The CORBA-like system appears to be closer to a federated database system than to a multidatabase system and its implementation

would allow this to be met.

4.5 DISWG's Real-Time Requirements Class

"Most mission-critical computing systems have a real-time component which must interact with the environment to produce timely and reliable results for successful performance of the mission at hand"[32]. The requirements in this class deal with the characteristics of the real-time components in next-generation DBMSs. "Some of the requirements in this section cannot be met by a DBMS implementation unless its underlying operating system provides certain real-time processing capabilities ..."[32]. In the rest of this section, we first provide some definitions. Next, we look at some important issues in real-time databases. Then, we point to areas of future research and follow with a discussion RTSORAC's incorporation into Open OODB. The section ends with the evaluations of DISWG's requirements.

4.5.1 Definitions

Real-time is "Characterized by the presence of timing constraints. Various levels of real-time can be distinguished as follows:

- **Hard real-time.** Hard real-time means that failure to execute within timing constraints produces catastrophic results.
- **Firm real-time.** Firm real-time means that failure to execute within timing constraints produces no useful results.
- **Soft real-time.** Soft real-time means that failure to execute within timing constraints produces less desirable results than would be produced by meeting timing constraints"[32].

Real-time database systems (RTDBS) can be defined as a database system where transactions are associated with real-time constraints typically in the form of deadlines

[90]. A real-time DBMS can be defined as "A DBMS that manages time-constrained data and time-constrained transactions"[32].

4.5.2 Important Issues in Real-time Databases

"Implementation of RTDBSs is difficult due to the conflicting requirements of meeting deadlines and maintaining data consistency"[90]. It is not enough to simply integrate "concepts, mechanisms, and tools from database systems with those from real-time systems"[75]. In this subsection, we examine some of the important issues raised by RTDBSs.

In non-real-time database systems, performance can be measured in terms of average throughput. However, in a RTDBS, performance measurements are more concerned with the timing constraints of individual transactions [2] and the percentage of transactions that meet their deadlines [98]. In a non-real-time database system, acceptable performance can mean nothing more than the job eventually gets done. In a RTDBS, jobs must be done in a predictable amount of time.

Unfortunately, there is much unpredictability in the traditional database domain [28]. It can not be predicted how often a data reference will result in a main memory access as opposed to a disk access [75]. Since, in many cases, real-time schedules are based on worst-case performance, this I/O problem can severely impact performance [98]. It can not be predicted how often recovery may be needed and how long those recoveries may take. Recoveries usually involve unpredictable rollbacks and restarts [75]. It can not be predicted how the active nature of things like integrity constraints and security checks will affect performance.

There are problems other than predictability presented by RTDBSs. There is no agreed upon data model. There is no agreed upon "best" scheduling algorithm. For instance, the earliest-deadline-first algorithm appears to work best under some circumstances, but in an overloaded system, the least-slack-time algorithm performs

better [2, 98]. Also, there is no "best" concurrency control mechanism [2]. Locks are very undesirable in real-time applications [98], which means that RTDBSs probably do not want to rely on the popular two-phase locking protocol [37]. Various methods of assigning and adjusting priorities are being studied to circumvent this problem [2, 75].

The problems presented by RTDBSs necessitate trade-offs and sacrifices. Serializability requirements will have to be relaxed because of the unpredictability of blocking and restarts [28, 90, 98]. It might be acceptable to enforce external correctness while temporarily allowing internal inconsistencies [90]. Recovery will have to be looked at differently, meaning that it might be better not to recover in certain applications [28]. The traditional ACID properties will have to be relaxed [28, 96] because, for example, transactions must communicate in many real-time applications. Interestingly, relaxation of the ACID properties is also being looked at outside the realm of real-time databases [17]. Transactions may have to be partitioned to allow early commits [28]. Thus, RTDBSs require a rethinking of DBMSs at a very basic level.

4.5.3 The Future of RTDBSs

Most of the work towards developing RTDBSs "uses the relational data model, which has restrictions in representing complex data, constraints and concurrency control"[73]. This necessitates considering other data models such as the OO model. Also, the problem of disk I/O has fueled research into main memory databases [90]. If main memory becomes sufficient to handle large databases, then much of the unpredictability of RTDBSs will go away. Main memory databases also offer the potential for increased speed because they would eliminate time consuming disk I/O [34]. Additionally, since SQL is now the basis for query language standards [82], work is underway to extend SQL with real-time features [28]. SQL is also being extended to include OO features and those two extensions would bring a real-time, next-generation

database that much closer to reality.

4.5.4 RTSORAC and Open OODB

At present, Open OODB has no real-time capabilities and does not meet any of DISWG's real-time requirements. Therefore, the evaluations of most of these requirements are similar in format. We find that Open OODB has no matching requirements and that its proposed architecture and implementation do not meet the requirements.

There are several problems in extending Open OODB with real-time functionalities. One is that Open OODB is based upon transparently extending C++. Thus, many possibly nested extensions can all be going on at once. This situation, which is at the heart of Open OODB's computational model, creates unpredictability. Also, Open OODB currently relies on the Exodus storage manager which has no real-time capabilities. However, the extensible, modular nature of Open OODB and the fact that it uses an SQL-based query language [62, 81], make Open OODB a prime candidate for real-time extensions.

At URI, we are extending Open OODB with RTSORAC features to allow many of DISWG's Real-Time requirements to be met. This fact is alluded to throughout the discussions of the requirements in this class. We know of no other work towards the implementation of a real-time OODB as developed as RTSORAC. For more details on RTSORAC, refer to the section on RTSORAC starting on page 26 or to [73]. At URI, we are also working on porting Open OODB to a real-time, POSIX-compliant operating system. This is essential, because DISWG states in its own introduction to this class that many of these requirements rely on the support of a real-time operating system. OMG is also interested in POSIX compliance [57]. We now evaluate the eleven requirements in this class.

4.5.5 Modes of Real-Time

The NGCR DBMS interface standards shall provide support for hard real-time, firm real-time, and soft real-time modes of operation.

Hard, firm and soft real-time are defined in the introduction to this class.

Open OODB

Requirements. This is not matched.

Proposed architecture. This is not met.

Implementation. This is not met. We first point out that Open OODB would have to be ported to a real-time operating system for this to be met. URI is working towards porting Open OODB to a real-time, POSIX-compliant operating system. Next, not everyone acknowledges that there are three distinct real-time modes. Some omit firm real-time [90]. That being said, the problems presented by hard real-time constraints are different than those presented by firm or soft real-time constraints. Extending Open OODB with the features of RTSORAC would allow the firm and soft requirements to be met. Hard real-time in an OODB is an area that needs more research. Even in non-OO systems, many restrictions have to be placed on transactions and "poor resource utilization may result given the worst-case assumptions made about the activities"[75]. This is because, in hard real-time, all deadlines must be met to avoid catastrophe.

As already noted, many factors hinder predictably meeting deadlines and this problem is exacerbated in hard real-time. In hard real-time, the whole system must be predictable and POSIX does not provide that kind of support. Also, Open OODB does not provide system wide predictability. Problems like locking and disk I/O are compounded by data that is evolving and quickly becoming out-of-date [28]. The

real world is constantly changing, a situation hard to correctly reflect in a database. Sometimes, we may have to accept timely results that are not correct.

Main Memory databases are expected to help towards the implementation of hard real-time OODBs due to their increased predictability [90] and potential for increased speed [34].

4.5.6 Real-Time Transactions

> *The NGCR DBMS interface standards shall provide the capability for users to issue real-time transactions where ACID properties (such as the isolation property, which can be relaxed via the specification of alternative concurrency control correctness criteria) are applied selectively, and where start events, deadlines, periods, and criticality of the real-time transactions are specified.*

Concurrency control correctness criteria is looked at more closely in the next requirement. A start event is "An occurrence in the system (e.g., reaching a specified wall-clock time, activation of a database trigger) constraining the start of a time interval"[32]. A deadline is "An absolute (wall-clock) time constraining the end of a time interval"[32]. "A period establishes regular time intervals of a constant relative time duration where the start of the i^{th} interval is the end of the $i-1^{st}$ interval. A periodic constraint requires that execution appear once and only once within every generated period"[91].

Open OODB

Requirements. This is partially matched under R3 Concurrent Access, by R3-2-9 which requires support for cooperating transactions. This is the same as relaxing the Isolation component of the ACID properties as required by DISWG above. Also, other

subrequirements of R3-2 require support for different concurrency control correctness criteria.

Proposed architecture. This is not met.

Implementation. This is not met. Open OODB's only transaction facility, as we discussed in DISWG's Basic DBMS requirement for transaction support on page 60, is conservative and based on Exodus. Also, Open OODB would have to be ported to a real-time operating system for this to be met, otherwise the expression of deadlines, etc., would be meaningless. Once ported, extending Open OODB with the features of RTSORAC [73] would allow this to be met. RTSORAC allows for the ACID properties to be relaxed and for start events, deadlines and periods to be expressed.

In RTSORAC, "A transaction consists of six components, $< N_t, O, OC, PreCond, PostCond, Result >$"[73], as discussed on page 26. The OC component is where constraints on transactions are represented. These include precedence, execution and timing constraints. The DBMS, when provided with that information, could decide on the criticality of a particular transaction. That decision would then be passed on to the operating system.

The real-time transaction model presented in [2] is similar, but in a non-OO environment. In [2], transactions have three parameters that directly correspond to the three specifics mentioned in this requirement. Those parameters are: "a release time r, a deadline d, and a runtime estimate E"[2]. These three parameters are used to set schedules. It is worth noting that in [2], the attempt is to enforce serializability with combinations of scheduling, locks and priorities. The authors note that enforcing serializability adversely affects performance. In GemStone [13], though not real-time, serializability may be relaxed by allowing dirty reads. With dirty reads, the ACID properties' Isolation and Consistency components are relaxed.

4.5.7 Concurrency Control Correctness Criteria

The NGCR DBMS interface standards shall provide the capability for users to specify concurrency control correctness criteria.

Concurrency control correctness criteria is "The criteria that establish the allowable interleavings of concurrent execution sequences. Serializability is an example of a typical concurrency control correctness criteria"[32]. Under serializability, a schedule is said to be serializable if it is equivalent to some serial schedule [26].

Open OODB

Requirements. This is not matched.

Proposed architecture. This is not met. Concurrency control mechanisms would most likely be included in the Transaction PM which is currently unimplemented. However, nothing in Open OODB's design precludes this from being met in the future.

Implementation. This is not met. Open OODB uses the strict locking mechanisms of Exodus to enforce serializability as a correctness criteria. We already pointed out that many do not think serializability is adequate to handle real-time needs [28, 90, 98]. In the RTSORAC model, one component of each object is CF or a compatibility function [73]. This function allows for concurrency on the granularity of individual methods. A boolean value is used to represent whether concurrent execution is allowed for every pair of an object's methods. The designer of an object is responsible for the definition of the compatibility function. Note that CF handles intra-object concurrency control. It is unclear how to handle inter-object concurrency control.

In [2], various concurrency control methods are compared. This is done by pairing different scheduling algorithms with different locking mechanisms in different types of systems. The scheduling algorithms include earliest-deadline-first, least-slack-time and first-come-first served.[22] The locking mechanisms are variations on two-phase locking obtained by using different priority schemes including wait, wait-promote and highest. The systems are disk and main memory resident, as well as normal and overloaded. The authors conclude that in a disk resident system, which represents the current reality, a combination of least-slack-time and wait-promote works best.

4.5.8 Temporal Consistency

The NGCR DBMS interface standards shall provide the capability for users to specify data temporal consistency constraints.

Temporal consistency is "A property of data. Data exhibits temporal consistency if it meets specified timing constraints"[32].

Open OODB

Requirements. This is unmatched.

Proposed architecture. This is unmet.

Implementation. This is unmet. RTSORAC uses an object's C, or constraint, component to maintain temporal consistency. "Each constraint is of the form $< N_c, AttrSet, Pred, ER >$" where N_c is the name of the constraint and $AttrSet$ is a subset of the object's attributes [73]. $Pred$ is a boolean that can be used to express the temporal consistency of an object's data by "referring to the value, time, and imprecision fields of the attributes"[73]. The ER, or enforcement rule, is triggered

[22] We do not discuss these algorithms here, the reader is referred to [2] for a complete treatment.

when the boolean value of *Pred* evaluates to false. Open OODB extended with this RTSORAC functionality would meet this DISWG requirement.

In [75], it is concluded that the effect of maintaining temporal consistency on transaction timing is an open question.

4.5.9 Real-Time Scheduling

> *The NGCR DBMS interface standards shall provide DBMS real-time scheduling that attempts to maximize meeting timing constraints and criticality (the synthesis of these two requirements is left undefined here) of transactions, as well as attempting to maintain both logical and temporal consistency of data. The NGCR DBMS interface standards shall require that real-time scheduling support analysis of predictable timing behavior (e.g., by bounding priority inversion).*

Open OODB

Requirements. This is not matched.

Proposed architecture. This is unmet.

Implementation. This is unmet. This requirement points to a problem brought up earlier in this thesis: where is the interface between the operating system and DBMS? If the DBMS schedules real-time transactions, how is this affected by the operating system scheduling tasks? Clearly, it would be most efficient to have the scheduling performed only once. At any rate, Open OODB would have to be ported to a real-time operating system in order to meet this requirement.

This requirement mentions many things; we look at them one at a time. The first thing mentioned in this requirement is maximizing meeting timing constraints. URI is porting Open OODB to the real-time, POSIX-compliant, Lynx operating

system. Additionally, the scheduling capabilities of Lynx have been improved upon to allow more tasks to meet deadlines [79]. Next mentioned is the maintenance of logical and temporal consistency while meeting real-time schedules. The capabilities of RTSORAC as concerns logical and temporal consistency will be looked at in other requirements in this class. We have already noted that the impact of meeting real-time schedules while preserving consistencies is an open research area [75].

The last subject mentioned is support for analysis of predictable timing behavior, "e.g., by bounding priority inversion." This again points to the operating system/DBMS interface. If a DBMS is granting locks and using a priority scheme to bound inversion, what is the impact of the operating system also assigning priorities and locks? This is an open question.

This requirement could be met by extending Open OODB with URI's work on the Lynx operating system and RTSORAC. However, it is clear that a more efficient implementation could be realized if operating systems were designed with support for DBMSs in mind.

4.5.10 Bounded Logical Imprecision

The NGCR DBMS interface standards shall allow logical imprecision of data; it shall provide the capability to constrain these imprecisions.

DISWG defines logical imprecision is "the degree to which data fails to meet integrity constraints"[32].

Open OODB

Requirements. This is not matched.

Proposed architecture. This is unmet.

Implementation. This is unmet. Extending Open OODB with RTSORAC features would allow this to be met. RTSORAC has several mechanisms that bound logical imprecision. The C, or constraint, component of an object, as previously detailed in the temporal consistency requirement of this class, is one mechanism. Also, each object has an A, or attribute component. Each "attribute is characterized by $< N_a, V, T, I >$", where N_a is the name and V the value [73]. T is a timestamp and I "is used to store the amount of imprecision associated with the attribute"[73]. The object's C component can write constraints on I. Furthermore, it has been proven that the amount of this imprecision can be bounded [16].

4.5.11 Bounded Temporal Imprecision

The NGCR DBMS interface standards shall allow temporal imprecision of data; it shall provide the capability to constrain these imprecisions.

DISWG defines temporal imprecision as "The degree to which data fails to meet timing constraints"[32].

Open OODB

Requirements. This is unmatched.

Proposed architecture. This is unmet.

Implementation. This is unmet. Extending Open OODB with RTSORAC features would allow this to be met. As just discussed in the evaluation of the last requirement, each attribute of an object in RTSORAC may have a T, or a timestamp, component. This component would be watched over by the object's C, or constraint, component.

A slightly different approach is taken in [75] where temporal consistency is considered to have two components: absolute consistency and relative consistency. In

[75] data, d, is a triple: $(value, avi, timestamp)$. "d_{value} denotes the current state of d, and $d_{timestamp}$ denotes the real-time when the observation relating to d was made. d_{avi} denotes *d's absolute validity interval*, i.e., the time interval following $d_{timestamp}$ during which d is considered to have absolute validity"[75]. If data items are used to derive a new data item, a *relative consistency set*, R, is formed. Each R has a *relative validity interval* or R_{rvi} which defines the maximum allowable difference between the timestamps of the data involved.

4.5.12 Main Memory Data

The NGCR DBMS interface standards shall provide the capability to specify that certain parts of the database should be maintained exclusively in main memory. The NGCR DBMS interface standards shall require that the DBMS still be responsible for maintaining persistence of this main memory data.

Open OODB

Requirements. This is partially matched. Under R3 Concurrent Access, R3-3-1 requires "controllable commits – (e.g., we can force everything to stay in memory for performance reasons)"[69]. Also, under R17 Other DBMS Requirements, R17-1 requires support for a main memory DBMS [69]. However, there is no requirement for maintaining the persistence of this main memory data.

Proposed architecture. This is not met.

Implementation. This is not met. This requirement is on the DBMS maintaining *persistence of parts* of the database in main memory. Use of the ASM abstraction allows parts of a database maintained by Open OODB to reside in main memory.

However, the user does not control which parts and main memory data is not persisted beyond the lifetime of a program unless sent to a persistent store.

On a more general level, the algorithms to maintain parts of the database in main memory would have to be similar to the algorithms to maintain the entire database in main memory. Main memory databases are outside the current scope of Open OODB and it is "hard to predict the amount of rework involved to add this capability"[69]. We mentioned main memory databases briefly in the introduction to this class. Main memory databases hold promise for RTDBMSs due to their increased predictability over disk based systems [90] and the potential for increased performance [34]. However, implementation of large, main memory databases awaits technological advances and it is unclear when they will become a reality. When they do, nothing in Open OODB's design precludes it from taking advantage of the added speed and predictability that would be provided.

4.5.13 Time Fault Tolerance

> *The NGCR DBMS interface standards shall support time fault tolerance. That is, violations of transaction timing constraints and data temporal consistency constraints are faults and shall be treated as such by the fault-tolerance capabilities of the standard, as specified [...previously].*

Open OODB

Requirements. This is unmatched.

Proposed architecture. This is unmet.

Implementation. This is unmet. In order to meet this requirement, Open OODB would have to be ported to a real-time operating system that can detect time faults. Also, there would have to be some way to express time faults in the database.

Fault tolerance, like real-time, is an issue that raises the operating system/DBMS interface question. Certainly, transaction recovery is different than task recovery. However, even the best transaction recovery algorithms are dependent on the underlying operating system providing adequate fault tolerance support. For instance, what good is the best logging algorithm if the operating system, due to a fault, jumbles up the files involved? We look at fault tolerance more closely in evaluation of the next DISWG requirements class: Fault Tolerance.

4.5.14 Resource Utilization Limits

> *The NGCR DBMS interface standards shall allow the specification of worst-case resource utilization limits (at least, CPU time, memory, devices, and data objects) for transactions. Violations of these limits are faults and shall be treated as such by the fault-tolerance capabilities of the standard, as specified [...earlier].*

Open OODB

Requirements. This is unmatched.

Proposed architecture. This is unmet.

Implementation. This is unmet. Once again, where is the division between the operating system and the DBMS? Limits for CPU time, memory and devices are best handled by the operating system, while limits on data objects are best handled by the DBMS. Therefore, to meet this requirement, the underlying operating system would have to tabulate resource limits and allow that information to be used by the DBMS. Also, the DBMS would have to provide support for the definition of data limits such as the maximum size of a set, etc. Once a fault of this type is detected, a

fault tolerance mechanism must be invoked. Fault tolerance is looked at more closely in its own requirements class.

4.5.15 Compilable DML

The NGCR DBMS interface standards shall provide a compilable DML that yields a minimal run-time burden.

Open OODB

Requirements. There is no matching requirement.

Proposed architecture. This is met. The DML is C++, which is compilable.

Implementation. This is met as noted above.

4.6 DISWG's Fault Tolerance Requirements Class

"The requirements in this section specify fault management capabilities which must be provided by NGCR DBMS Interface Standards"[32]. DISWG's requirements are on military systems possibly involved in life-or-death situations. These "Mission-critical database systems must be reliable"[32]. As with the Real-Time requirements class, DISWG states that "Some of the requirements specified herein cannot be met by a DBMS implementation unless its underlying operating system provides certain fault management capabilities ..."[32]. Obviously, the best database recovery[23] scheme is ineffective if the underlying operating system, due to a fault, mismanages files. In the rest of this section, we first review some definitions. Next, we examine some popular fault tolerance strategies. Then, we look at Open OODB and fault tolerance. Finally, Open OODB is evaluated with respect to DISWG's requirements.

[23] Typically the term *recovery* is used in database literature instead of the term fault tolerance, which has more often been used with respect to operating systems. However, in the DISWG requirements, database fault tolerance encompasses database recovery and the necessary operating system support.

4.6.1 Definitions

The DISWG Fault Tolerance requirements are based on *failure* of a *DBMS component* that could be viewed as a *fault* in the overall system. A DBMS component is "A physical device or logical execution entity used or controlled by the DBMS. DBMS components include, by are not limited to, data objects, transactions, database managers, database sites, communication media among database sites, processors, memory, and secondary memory"[32]. A failure is when a DBMS component "deviates from its specified behavior"[32]. A fault is a failed component "from the viewpoint of higher-level components that encompass it"[32]. Fault tolerance is "The ability of a component to maintain its specified correct behavior in the presence of faults (e.g., failures of subcomponents)"[32].

4.6.2 Popular Strategies

In this subsection, we discuss two of the most popular database fault tolerance schemes. To initiate this discussion, we first look at exactly what is meant by a failure. There are three types of failures [51, 53]. Perhaps the most typical failure is a transaction abort due to errors, deadlock or initiated by the user for some reason. The second type is a system crash and the third type is media, or device failure.

DBMS recovery techniques typically involve one of two strategies [53]. The most common strategy is based upon keeping logs of a transaction's actions. Upon failure, these logs can be used to bring the database back to a previous state known to be consistent. The most popular of these strategies is the write-ahead log based ARIES [51] recovery method. In this strategy, effects of transactions must be logged before being entered into the database. Upon failure, a *redo* is performed to reestablish the state of the database at the time of failure. Then, an *undo* is performed to bring the database back to a consistent state. Many systems use an ARIES based or ARIES-like strategy including ObjectStore [48], ORION [43] and the Exodus storage manager.

The second strategy is shadow-paging. In shadow-paging, all a transaction's updates are performed on a copy of the appropriate database page. The locations of the copy and the original page are kept in tables. Therefore, upon failure, the copy can be discarded and the table that points to the original can be used to restore the database to a consistent state. "The advantage of shadow paging is that it makes undoing the effect of the executing transaction very simple"[26]. "One problem with the shadowing approach is ... it requires maintaining a very large page table"[53] and another is that it is "difficult to keep related database pages close together on disk without complex storage management strategies"[26]. GemStone uses a shadow-paging recovery scheme [13].

4.6.3 Important Issues

Redundancy is the key to any recovery or fault tolerance scheme. Logging is a form of redundancy. A transaction's results are not only written to a page, but also redundantly written to a log. Shadow-paging also involves redundancy at the page level. A more obvious form of redundancy can occur at the hardware level. Redundant disks and CPUs, etc., are used to protect mission-critical systems.

Redundancy at the hardware level introduces a new level of complexity: distribution. With databases, the main problem with redundancy and distribution is to make sure that all data copies remain consistent while not overburdening the system with replication management [1]. Typical replicated data management schemes involve forming groups of replicas and voting [76]. Distribution in DBMSs has been cited as one of the most difficult next-generation functionalities to implement [13]. Thus, we have the paradox that replicated, distributed components enhance fault tolerance, but complicate the needed algorithms.

Another next-generation concern is implementing fault tolerance under real-time constraints. Fault tolerance is achieved by redundancy, which immediately increases

overhead. Also, we certainly can not accurately predict how many failures will occur, and each failure further increases overhead. Thus, there is a performance trade-off and an unpredictable factor in bounding execution time. That unpredictability is in addition to the fact that, as we noted on page 91 in the introduction to DISWG's Real-Time requirements class, traditional recovery methods do not satisfy real-time needs [28].

The issue of fault tolerance begs the same question that real-time does: where does the operating system end and the DBMS begin? In a fault tolerance scheme proposed for UNIX [11], a checkpoint of a consistent state is taken between tasks. A record is kept of all changes to that state. Upon failure, the system is rolled back to the consistent state and then the changes are redone. In a typical database recovery scheme, a checkpoint of a consistent state is taken between transactions. A log is kept of all changes made to that state. Upon failure, the system is restored to the consistent state and the changes redone.

The parallels are obvious. However, transactions are not supported by operating systems due to the high overhead of maintaining consistency through serializability.[24] Despite that fact, due to the similarities of the fault tolerance algorithms in the operating system and in the DBMS, more cooperation between the two could greatly improve efficiency.

4.6.4 Open OODB and Fault Tolerance

Open OODB does not meet any of the requirements in DISWG's Fault Tolerance class. Open OODB does not implement a recovery strategy, but instead relies on the ARIES based algorithms of the underlying Exodus storage manager. Open OODB's recovery requirements are undeveloped and do not even merit a separate category (they are lumped in with integrity). Also, Open OODB rests on the UNIX operating

[24] This is from an unfinished paper by Paul Fortier and Joan Peckham entitled *Operating System Support for Next Generation Database Management Systems.*

system which does not have fault tolerance capabilities. In addition, Open OODB's computational model with (possibly nested) transparently extended events would make recovery "very complicated"[68].

Therefore, the evaluations of the DISWG Fault Tolerance requirements are, for the most part, all the same. The DISWG requirement is found to be unmatched and the requirement is found to be unmet by both the proposed architecture and the implementation. Also, we note that many of the requirements need underlying operating system support.

We suggest the definition of an additional Policy Manager to handle fault tolerance. The Fault Tolerance PM could do many of things called for in these DISWG requirements. Included under the scope of a Fault Tolerance PM could be setting fault limits, initiating tests and collection of fault information, etc. These functionalities are orthogonal to the functionalities of the rest of the PMs. These functionalities should also be accessible to application programs. Orthogonality and application accessibility are two of the main justifications for the development of a separate PM in Open OODB. Of course, Open OODB would still have to be ported to a fault tolerant operating system. The Fault Tolerance PM would have an interface into the fault tolerance mechanisms of the operating system. We now evaluate the nine requirements in this class.

4.6.5 Collection of Fault Information

The NGCR DBMS interface standards shall specify the fault information (e.g., the component that failed, the number of times the fault occurred, when the faults occurred) to be collected. The standard shall also specify a minimal set of faults for which the specified information shall be collected. This set shall include, but is not limited to, the following faults:

- *Database constraint violations (e.g., range constraints, referential in-*

tegrity constraints, temporal consistency constraints).

- *Transaction timing faults.*
- *Transaction resource utilization violations.*

Open OODB

Requirements. This is not directly matched. Under R5 Data Dictionary, R5-1-10 requires support for mechanisms to collect and maintain information such as configurations and replications.

Proposed architecture. This is not met.

Implementation. This is not met. Open OODB would have to be extended in several ways in order for this to be met. First, Open OODB would have to provide support for integrity constraints. We look at this problem in more detail in the evaluation of DISWG's Integrity requirements class. Second, Open OODB would have to be extended with real-time features to be able to express things like transaction timing faults. Third, Open OODB would need an operating system with more fault tolerance capabilities than UNIX. Finally, a mechanism would be needed to coordinate the specification of fault information that is to be collected from the DBMS and from the operating system. One way to do this would be to define a Fault Tolerance PM as discussed earlier.

Extending Open OODB with features of RTSORAC [73] would help. RTSORAC supports the expression of both integrity constraints and real-time constraints. The Fault Tolerance PM could watch for violations of theses constraints.

4.6.6 Retrieval of Fault Information

The NGCR DBMS interface standards shall provide for the retrieval of DBMS fault information.

Open OODB

Requirements. This is unmatched.

Proposed architecture. This is unmet.

Implementation. This is unmet. Information on some faults, particularly hardware, would best be collected by the operating system. Therefore, to meet this requirement, an interface from the DBMS to that information in the operating system would be needed. In Open OODB's case, that would mean that either UNIX is extended to collect fault information and allow a DBMS access to that information, or Open OODB would have to be ported to a different operating system. As suggested in the introduction to this class, a separate Fault Tolerance PM should be considered for Open OODB. This PM could perform such duties as retrieving fault information.

4.6.7 Initiation of Diagnostic Tests

The NGCR DBMS interface standards shall provide for the initiation of DBMS diagnostic tests.

Open OODB

Requirements This is unmatched.

Proposed architecture. This is unmet.

Implementation. This is unmet. Some tests, particularly on hardware components, should be performed by the operating system. Currently, Open OODB rests on UNIX, which does not provide the necessary support to meet this requirement. Therefore, either UNIX would have to be extended or Open OODB would have to be ported to a new operating system. Diagnostic tests would also be needed on Open

OODB modules such as Policy Managers. Many modules are currently unimplemented. However, Open OODB's design goals of well-defined modules and interfaces would facilitate the implementation of appropriate tests.

4.6.8 Retrieval of Results of Diagnostic Tests

The NGCR DBMS interface standards shall provide for the retrieval of the results of DBMS diagnostic tests.

Open OODB

Requirements. There is no matching requirement.

Proposed architecture. This is unmet.

Implementation. This is unmet. Diagnostic tests on hardware should be handled by the operating system. To meet this requirement, the operating system would have to make the results of those tests available to the DBMS via an interface. The Fault Tolerance PM of the DBMS, as suggested in the introduction to this class, would access those results as well as the results of tests performed on software components of the DBMS.

4.6.9 Operational Status

The NGCR DBMS interface standards shall provide access to the operational status of DBMS components.

Open OODB

Requirements. This is unmatched.

Proposed architecture. This is unmet.

Implementation. This is unmet. The operational status of hardware components should be tracked by the operating system. As for software components of the DBMS, we recommend the development of a Fault Tolerance PM as described in this class's introduction. This Fault Tolerance PM would need an interface into the fault tolerance mechanisms of the underlying operating system.

4.6.10 Fault Detection Thresholds

The NGCR DBMS interface standards shall provide for the specification of fault detection thresholds, which shall include, but not be limited to, the number of faults that if detected within a certain amount of time is treated as a failure (e.g., the number of retry attempts of aborted transactions before a failure of that is reported).

Open OODB

Requirements. There are no matching requirements.

Proposed architecture. This is unmet.

Implementation. This is unmet. Once again, operating system support would be needed to meet this requirement. Also, a Fault Tolerance PM would facilitate meeting this requirement. Allowing applications access to the Fault Tolerance PM would allow users to fine-tune fault thresholds on a needs-be basis. For instance, in some applications, we may not wish to retry many transactions, whereas in others, we may wish to retry all transactions.

4.6.11 Specification of Fault Responses

The NGCR DBMS interface standards shall provide for the specification of actions to be taken at the occurrence of a fault. They shall support at

least the following actions:

- *Restart of a specified set of transactions at a database's specified past state or with only a specified part of the database replaced by its past state.*
- *Rollback of specified transactions that have started, but no yet committed, so that their effects are not realized in the database.*
- *Use of specified backup components as primary components (e.g., other versions of the database).*
- *Providing notification of a fault to a specified set of DBMS components to allow them to initiate recovery.*
- *Providing notification of a fault to a specified location outside of the DBMS.*
- *Reconfiguration of DBMS components (see next requirement).*

Furthermore, the NGCR DBMS interface standards shall allow for each of these actions to be applied selectively. Also, these actions may fall under time-constrained execution described in the "Real-Time Processing" section.

Open OODB

Requirements. This is unmatched. However, some Open OODB requirements provide a partial match. Under R5 Concurrent Access, R5-1-9 requires mechanisms for collection of replication and configuration information. Also, under R16 Industrial Strength, R16-1 requires the system to be fault tolerant, R16-1-1 requires logging and R16-1-2 requires automatic recovery after crashes. R16-2 requires "backup, restore and archiving"[69]. Also, under R9 Data Integrity and Recovery, a system must support R9-5 recovery and R9-6 backup and restore.

Proposed architecture. This is unmet.

Implementation. This is unmet. To meet this requirement, operating system support would be needed. As we have noted earlier in this evaluation of the Fault Tolerance class, UNIX would have to be extended or replaced with a fault tolerant operating system. Also, a separate Fault Tolerance PM would be a good way to extend Open OODB to meet this requirement.

This requirement mentions several distinct issues involved in fault response. The first issue is the use past states and rollbacks. We have already mentioned that Open OODB considers this a very complicated issue due to transparent extensions [68].

Many of the other issues could only be met by cooperation between the DBMS and the operating system. For instance, hardware reconfiguration in the face of hardware failure should be handled by the operating system. Selectivity could be handled by bundling fault tolerance capabilities into a PM with an applications interface. Real-time processing, as we noted in the introduction to this class, is an open question with respect to fault tolerance.

4.6.12 Reconfiguration

> *The NGCR DBMS interface standards shall support dynamic reconfiguration of the DBMS components based on reconfiguration of the underlying operating system and hardware. Reconfiguration includes, but is not limited to, enabling/disabling components, adding/deleting components as members of specified groups and reassigning resources to components. Reconfiguration must be allowed as a response to a fault, as in the previous requirements, or at the discretion of certain DBMS components.*

Open OODB

Requirements. This is not directly matched. However, under R5 Data Dictionary, R5-1-7 requires that the data dictionary maintain configuration information. Also, Open OODB has been designed with configurability in mind and has many requirements in this area. We discussed this at length in the configurability requirement of DISWG's General class on page 47.

Proposed architecture. This is unmet.

Implementation. This is unmet because UNIX, Open OODB's operating system, does not provide the support called for in this requirement. Open OODB does provide limited support for enabling and disabling software components. For instance, transactions can be aborted, and, theoretically, different address spaces can be added or deleted. However, none of this is done "based on reconfiguration of of the underlying operating system and hardware", to quote the requirement.

Hardware reconfiguration would have to be tracked by the operating system. The DBMS, possibly in the form of a Fault Tolerance PM, would need access to this information. The DBMS would also need to monitor software configuration. Open OODB has been designed with configurability in mind, which should facilitate reconfiguration for fault tolerance purposes at the software level.

4.6.13 Replicated Components

The NGCR DBMS interface standards shall not preclude the use of replicated components.

Open OODB

Requirements. This is not directly matched. However, under R4 Distribution, R4-10 requires support for replicated data for fault tolerance.

Proposed architecture. This is met due to the wording of the requirement, e.g., nothing in Open OODB's proposed architecture *precludes* the use of replicated components. Open OODB's designers have proposed a Replication PM outside of the scope of the alpha release to handle replicated data objects [67]. Replicated hardware could be supported by the services of the ASMs, the Distribution PM, and the Communications and Translation modules. Even though nothing in Open OODB's proposed architecture has been designed to specifically handle replicated hardware, this functionality is not precluded by Open OODB's design.

Implementation. This is unmet. Replicated hardware would be best handled by the operating system. The DBMS would need to interface that replication information. Open OODB's proposed Replication PM could handle replicated data. However, it is not clear how to best handle replicated software components, such as PMs, for fault tolerance purposes. Also unclear is how to handle replicated transactions (transactions are DBMS components).

4.7 DISWG's Integrity Requirements Class

The Navy, just like business and industry, bases important decisions on data retrieved from databases. "In order for the *right* decisions to be made, stored data used for such decisions must be correct and consistent. That is, the *integrity* of the stored data must be upheld"[32]. In this introduction, we first look at some of the problems involved in expressing integrity constraints, and then look at how the OO paradigm may be of help. Next, we take a brief look at integrity constraints in RTSORAC [73]. Finally, we perform the evaluations.

Data integrity must be maintained under pressure from concurrency control, recovery and other factors. The constant checking involved in maintaining data integrity can impact performance. Also, constraint checking could cascade, with unpredictable results [93]. Many constraints may not be computable, and some may not even be

expressible [24].

4.7.1 OODBs and Integrity

The "role of integrity constraints in OODB's is not very well-defined"[93]. Consequently, Open OODB lumps data integrity and recovery together into one, rather undeveloped requirement. This is reflected in the proposed architecture. For example, while there may be a matching Open OODB requirement for a particular DISWG requirement, none of these DISWG requirements are met by the proposed architecture of Open OODB. Open OODB envisions that the sentry mechanism can be used to enable integrity extensions.

The nature of the OO paradigm appears to lend itself well to the expression of integrity constraints. We can look at two types of integrity constraints. *Intra*-object constraints exist within an object and *inter*-object constraints exist between objects. Some intra-object integrity constraints are automatically captured through the type system and class hierarchies. For example, every student object is also a person object [39]. Also, intra-object data values can be protected or watched using constraint methods. Since objects can be constructed so that nothing but its own methods can touch its data, the implementation appears straightforward. Thus, the expression of intra-object constraints in an OODB is very natural.

The expression of inter-object constraints is not so straightforward. Some feel that these inter-object constraints should not be encapsulated within the object [39, 93]. In the RTSORAC[25] [73] model proposed at URI, intra-object constraints may be expressed within an object and inter-object constraints within *relationship* objects. Relationship objects in RTSORAC "represent aggregations of two or more objects"[73]. In [23], details on how to implement relationship objects are provided. Formal models of relationship objects are provided in [73, 96]. It is also worth noting that OMG sees

[25] More details on RTSORAC are provided in the evaluation of DISWG's Real-Time requirements class and in the review of current research at URI.

relationships as first class objects in its OSA [57]. Extending Open OODB to include relationship objects would allow for most of DISWG's Integrity requirements to be met. We now evaluate the eight requirements presented here with respect to Open OODB.

4.7.2 Domains

> *The NGCR DBMS interface standards shall provide the capability to define domains and to declare attributes as having values drawn from specified domains.*

A domain is a pool "of legal values for an attribute of an object ..."[32].

Open OODB

Requirements. There is no matching requirement.

Proposed Architecture. This is met. The C++ type system mechanism establishes domains as well or better than most DDLs. However, there is no explicit mention of domains in Open OODB's literature.

Implementation. This is met. As just noted, C++'s type mechanism provides support for domain definition. Also, where needed, it should be straightforward to express any additional domain constraints. The basic nature of the OO paradigm encapsulates data, and that data can be monitored by methods to make sure it stays in the exceptable range of values. ORION allows for domain checking to be including in an object [43]. In RTSORAC [73], domains can be expressed as intra-object constraints. Therefore, it should be possible to do the same in Open OODB.

4.7.3 Keys

The NGCR DBMS interface standards shall provide the capability to declare a specified attribute or set of attributes as a key.

A key is an attribute, "or set of attributes, that uniquely identifies an object within a class ..."[32].

Open OODB

Requirements. There is no matching requirement.

Proposed Architecture. This is unmet by the proposed architecture. Open OODB makes no mention of keys, or of special functionalities to handle keys.

Implementation. This is unmet by the implementation. Keys are more of an issue in other data models, such as the relational model. Keys can be seen as artificial since they necessitate all records to have a unique value [23]. The unique OIDs in an OODB offer more flexibility than keys. For instance, whereas keys in the relational model do not allow for two rows of a table to have the same values, OIDs allow for two objects to have the same values. OIDs give objects an identity independent of their value [15, 23]. This allows for more natural modeling of the real world where different things can have the same value.

However, OIDs do not eliminate the need for keys. OIDs are determined by the system and have no semantic meaning with regard to the object they identify. Therefore, "It is more convenient for the user to be able to fetch one or more objects using user-defined keys"[46]. The advantage of the OODB approach is that keys do not have to be unique because OIDs can be used to determine uniqueness. Thus, instead of using keys in the traditional sense, objects are searched over the domains of attributes [45].

4.7.4 Referential Integrity Constraints

The NGCR DBMS interface standards shall provide the capability to declare referential integrity constraints.

"Referential integrity holds when all referenced objects exist in the database"[32].

Open OODB

Requirements. This is a matched requirement. Under R9 Integrity and Recovery, R9-4 states that it must be possible to state referential integrity constraints. Also, under R1 OO Data Model, R1-1-14-5 states that an OODB must support some of a given list of functionalities. Included in that list is referential integrity constraints.

Proposed Architecture. The proposed architecture does not meet this requirement. This type of functionality is not explicitly proposed in Open OODB's architecture.

Implementation. Open OODB's implementation does not meet this requirement. Open OODB's designers state that this requirement could be met by its proposed architecture with minimal rework [69]. References "to objects are manipulated by all Open OODB modules" and thus, "are a part of the common glue defined by the meta architecture"[68]. One problem here is that its much easier to determine to where an object reference points than it is to determine from where that reference came [64]. This problem makes it difficult to tell if a deleted object was pointed to by another object(s).

One approach to solving the referential integrity problem in an OODB is the approach taken in Ode [39]. In Ode, each reference is implemented as an object that uses inverse pointers: if A points to B, then B must point to A. Ode's designers see three ways to implement referential integrity upon an object delete. A NULL value

could be placed in the reference pointer, the referencing object could be deleted or the delete action could be aborted. In each reference object, the application programmer includes which of the three actions to take upon deletion of a particular object. A similar method involving inverse pointers and user input is used in ObjectStore [48]. Finally, GemStone [13], based on C++, fully supports referential integrity constraints.

The reference objects described by Ode, ObjectStore and even Open OODB, are specific examples of the more general relationship object presented in RTSORAC [73]. In a relationship object, referential integrity constraints can be expressed, as can other inter-objects constraints.

4.7.5 Assertions

The NGCR DBMS interface standards shall provide the capability to declare assertions.

An assertion is a "constraint, possibly involving multiple classes of objects (and thus more powerful than a domain), on the data values that can be stored in a database. If the assertion fails to hold for an insert or update being submitted to the database system, then the operation is not performed, and an error is reported to the operator of that operation"[32]. Thus, an assertion is a statement of fact. For example, consider a hierarchy with the superclass ship. We have different types, or subclasses of ships such as freighters and tankers, etc. An assertion could be that that the speed of any ship can not exceed 100 knots. If it does, something is wrong because ships just can not go that fast.

Open OODB

Requirements. Open OODB has a matching requirement. Under R1 OO Data Model, R1-1-14-2 states that an OODB may support assertions. Also, under R5

Data Dictionary, R5-1-1 states that the data dictionary must be able to represent data model information such as assertions.

Proposed Architecture. This is not met by the proposed architecture.

Implementation. As of yet, this is unmet by Open OODB's implementation, but minimal rework is predicted to incorporate the features needed to meet this requirement [69]. The encapsulation of data inherent in the OO paradigm is expected to help with this, and other, integrity constraints. One way to handle inter-object constraints like assertions in an OODB is through the use of relationship objects. These objects express the relationships between other objects or tell how different objects reference each other. This is an integral part of ongoing research at URI. In RTSORAC [73] and also in [96], relationship objects are described in detail and in [23], implementation specifics concerning assertions are provided.

4.7.6 Triggers

> *The NGCR DBMS interface standards provide the capability to specify triggers.*

A trigger is a "mechanism for specifying that a sequence of database updates is to be performed upon the occurrence of a given event (e.g., access to a given object). Triggers can be used to propagate updates to maintain database consistency and can be used to maintain constraints. For example, a trigger could initiate a position update if a ship's speed changes.

Open OODB

Requirements. This requirement is matched. Under R1 OO Data Model, R1-1-14-6 states that the OODB may include trigger integrity constraints.

Proposed Architecture. This is unmet by the proposed architecture. There is no provision to include a trigger mechanism.

Implementation. This is unmet by the implementation. There is little discussion in OODB literature on triggers and other integrity constraints. Most OODBs do not support triggers [46]. One OODB, ORION, mentions triggers only to say that they are unimplemented [43]. The fact that methods watch over data in the OO paradigm should facilitate incorporation of triggers. Implementation of triggers in Open OODB can be patterned after work underway at URI. Triggers can be expressed in relationship objects that are designed to specifically watch over other objects [23]. A formal model of the relationship object is presented in [73] and [96]. A similar approach is taken in Jasmine [38]. Triggers in Jasmine are implemented as *demon* classes which watch for such actions as references, insertions and updates, etc. When a watched for action is detected by the demon class, an appropriate action, or method, is triggered.

4.7.7 Alerters

> *The NGCR DBMS interface standards shall provide capability to specify alerters.*

An alerter is a "mechanism for specifying that a message is to be sent to a specified process or user upon the occurrence of a given event (e.g., access to a given object). For example, if a ship changes course, all other ships could be automatically notified.

Open OODB

Requirements. There is no matching requirement.

Proposed Architecture. This is not included in the proposed architecture.

Implementation. This is not met by the implementation. However, Open OODB's designers do not predict much rework will be needed to incorporate alerters. Alerters could be modeled using RTSORAC's relationship objects [73], although there is no specific mention made of alerters.

4.7.8 Enabling/Disabling of Constraint Enforcement

The NGCR DBMS interface standards shall provide the capability to enable and disable the enforcement of specified integrity constraints. To restrict access to the capability, privileges shall be associated with this capability.

Open OODB

Requirements. This is unmatched.

Proposed Architecture. This is unmet. As stated earlier, DISWG's Integrity requirements are unmet by Open OODB's proposed architecture. Therefore, the integrity constraints, which are not there, can not be enabled/disabled or restricted.

Implementation. The implementation, while not meeting this requirement, does nothing to preclude it being met in future implementations. The modular nature of Open OODB's design should facilitate building in mechanisms to adjust constraint checking. This requirement also mentions privileges. This topic is covered in the discussion of DISWG's Security requirements class.

4.7.9 Null Values

The NGCR DBMS interface standards shall provide the capability to store and retrieve null values. Null-valued attributes shall be ignored in the computation of aggregate functions. For example, an averaging query

shall exclude null values from its computation of the average value of an attribute.

Open OODB

Requirements. There is no matching requirement.

Proposed Architecture. This is not met by the current architecture. There is no mention of treatment of null values.

Implementation. This is not met by the current implementation for reasons just stated. This appears to be more of a concern of other data models, such as the relational data model, that rely on keys, etc. In such models, it is important to make sure that no key has a null value. OODBs do not rely on keys. The DISWG requirement specifically mentions an averaging algorithm. Since Open OODB is an extension of C++, treatment of null values in such an algorithm could be handled in the same way as they would be in any C++ program. The treatment of null *pointers* in an OODB is an entirely different subject. For instance, what if we are dealing with a collection of composite objects, not all of which are complete? This is an open research area and currently would have to be done on an ad hoc basis.

4.8 DISWG's Security Requirements Class

DISWG offers no explicit definition of database security. Instead, twenty-two definitions are presented that deal with various aspects of security. Then, DISWG list its twenty security requirements. When considering the needs of the next-generation, the problems presented by security are not well understood. Due to this and perhaps other reasons, DISWG has deemphasized its security requirements.[26] Additionally, security is outside the scope of Open OODB's current effort. Therefore, our evalu-

[26] Minutes of the April 1994 Meeting, Alexandria, VA.

ation of this class is different than that of DISWG's eight other classes. We do not evaluate each DISWG requirement individually, but instead discuss security issues generally.

The rest of this section is organized as follows. We first discuss some of the most common security policies and strategies to date. Then, we look at OODBs and security. Next, we review what the Open OODB project has to say about security. We conclude with a brief look at some of the challenges that next-generation needs pose for security. For completeness, we include a list of the twenty DISWG requirements in this class.

4.8.1 Security Policies and Strategies

"Computer security is concerned with the ability of a computer system to enforce a security policy governing the disclosure, modification, or destruction of information"[50]. We now take a brief look at the two most popular security policies: *mandatory* access control and *discretionary* access control. "Mandatory security (or multilevel security) policies restrict access to classified information to cleared personnel"[50]. Basically, all data is classified as either top secret, secret or confidential, etc., and users are cleared based on their trustworthiness [72]. Mandatory access control can be summarized by two rules:

1. A subject S is not allowed to read data of access class c unless $class(S) \geq c$, and

2. A subject S is not allowed to write data of access class c unless $class(S) \leq c$.

The first rule means that subjects can not access data unless they are cleared to access that data. The second rule means that a subject can not write data that could then be accessed by other subjects with less clearance.

Mandatory security has primarily been used in military applications, many of which are very important. This does not imply that there are no problems with

mandatory security. Indeed, computer systems at both NASA and NATO have been breached [72], despite the fact that every read and write is checked under mandatory security [40]. Also, users at different clearances are presented different views of the same database. This has an adverse impact on integrity concerns such as null keys and dangling pointers, etc. [50, 72].

"Discretionary security policies ... define access restrictions based on the identity of the users (or groups), the type of access (e.g., select, update, insert, delete), the specific object being accessed, and perhaps other factors"[50]. "To support [discretionary access control], *user-role based security (URBS)* has been proposed, which dictates that the responsibilities of the end-users within the application be the guiding factor when assigning privileges"[20]. Thus, one way to implement discretionary policies is to associate both users and data with roles. A key problem with the discretionary approach is that it does not prevent malicious access [72].

There are other security strategies varying from armed guards to locked rooms. One of the most effective security measures taken aboard submarines is to *close the hatch*! As a final note on security strategies, security algorithms in general can be seen as very similar to the algorithms for integrity constraints, i.e., they both protect data [26, 47].

4.8.2 OODBs and Security

The OO paradigm could be of help in the area of security. Data hiding should support data security [10]. Some see OODBs shifting the emphasis from mandatory policies to discretionary policies [20, 50, 72]. This is due to how well the OO paradigm can be used to model user-role based discretionary access control. Roles can be associated with methods and users with roles [20]. Users can also be associated with groups and groups with roles, easing the burden on the implementor [55].

There are some concerns. Inheritance, one of the strengths of the OO paradigm,

tends to complicate matters. Security could be implemented in an OODB with inheritance by allowing access to be inherited to inherited methods, but not to subclass defined methods [30]. Another concern is that the analysis needed to associated users, groups and methods with roles could be very complicated and fragile [55]. For instance, any role changes could necessitate a complete reanalysis. Also, OODBs are more complex that simpler, relational databases. Therefore, security algorithms associated with OODBs may be more complex [56]. Finally, there is no agreement on just what an OODB should be, so how can there be any agreement on how security should be handled in an OODB environment [72]?

4.8.3 Open OODB and Security

Security is outside the current scope of the Open OODB project and it is hard to predict the amount of rework that would be needed to incorporate security [69]. Some feel that security must be designed into, rather than added onto, a database [72]. Nonetheless, adding security features is an ongoing research area for the Open OODB project.

Open OODB does not ignore the issue of security. Open OODB's functional requirement R10 states that "an OODB must support security"[69]. R10 is rather undeveloped and preliminary, with only seven subrequirements. To illustrate, R10-2 requires support for "authorization" while R10-5 requires support for "security authorization and access control"[69]. What is the difference between authorization and security authorization? Also, R10's subrequirements call for security support in a very general manner. For instance, R10-6 requires support for "Various sorts of security"[69].

Open OODB's designers state that security constraints could be a dimensional extension to the system architecture on a par with persistence and replication, etc. [64]. Even though "security access control is often bundled with transaction concurrency

control ... it is clearly something that is orthogonal"[65]. Nonetheless, Open OODB's designers are considering joining security and concurrency control into a single Policy Manager with both being checked at the transaction boundary [65].

4.8.4 The Next-Generation and Security

In the field of database security, there are still more concerns than controls and security systems lag behind the technologies they need to protect [72]. Security is seen as a major failing in current DBMSs [84]. "In spite of its importance, the issue of security has been relegated to a secondary status by researchers"[40]. One of the complications with security is that all the levels in a computer database system must be secure. In [72], four levels are discussed: hardware, operating system, DBMS and communication. The open systems interconnection (OSI) reference model contains seven layers [97]. All seven layers must be secure for the system to be secure. For instance, at the application level, resource access must be protected and at the transport level, encryption may be needed [97].

Clearly, to move into the next-generation of databases, this situation must change. There will be more information of more importance accessible to more users. However, next-generation needs present many challenges. The simple fact that networks are expanding highlights the need for increased security [72]. Distribution makes the security problem more difficult [88], but security is particularly critical in a distributed environment [31]. Heterogeneity complicates matters [88], causing [89] to conclude, "it is clear that the steps to achieving secure interoperability are by no means straightforward, and we believe that some of them are impossible." Also, real-time deadlines may be unenforceable under the constant checking of security constraints. We now list twenty requirements included in DISWG's Security requirements class. Remember, as mentioned above, Open OODB is not evaluated with respect to these requirements.

4.8.5 Multilevel Security

The NGCR DBMS interface standards shall have the ability to handle multilevel security.

4.8.6 Labeling

The NGCR DBMS interface standards shall provide the support for (1) labeling data and information (response, query, transaction, metadata, etc.), (2) handling different types of labeling granularity, (3) labeling DBMS subjects, (4) handling application-specific labeling constraints, and (5) exporting and importing labeled data and information.

4.8.7 Mandatory Access Control

The NGCR DBMS interface standards shall support a security policy based on subject and object labels. They shall also support the manipulation of the labels based on security policy.

4.8.8 Discretionary Access Control

The NGCR DBMS interface standards shall support (1) a mechanism for the enforcement of discretionary access control based on users and groups and (2) the manipulation of access rights to specifically include or exclude access based on users or groups. They shall also provide controls to limit propagation of access rights.

4.8.9 User Role-Based Access Control

The NGCR DBMS interface standards shall support the identification of users based on roles. They shall also support access control based on roles of users and transactions to be carried out.

4.8.10 Integrity

The NGCR DBMS interface standards shall support features which can be used to validate the accuracy of data and information.

4.8.11 Consistency

The NGCR DBMS interface standards shall support the enforcement of (1) application independent integrity constraints, (2) application specific semantic integrity constraints, (3) concurrency control techniques, and (4) recovery techniques, all without compromising security via covert covert channels or otherwise.

4.8.12 Identification and Authentication

The NGCR DBMS interface standards shall provide a protected mechanism to authenticate users' identities.

4.8.13 Security Auditing

The NGCR DBMS interface standards shall support (1) the generation of audit records that uniquely identify the users, events, and objects being operated upon, (2) the storage and maintainability of audit data, and (3) manipulation of audit data.

4.8.14 Least Privilege

The NGCR DBMS interface standards shall support the principle of least privilege.

4.8.15 Trusted Path

The NGCR DBMS interface standards shall support a trusted communi-

cation path between the user and the DBMS exclusively activated by the user.

4.8.16 Trusted Recovery

The NGCR DBMS interface standards shall provide procedures and/or mechanisms to assure that, after a failure, recovery without a security compromise is obtained.

4.8.17 Inference and Aggregation

The NGCR DBMS interface standards shall support features to control unauthorized inferences and aggregation problems.

4.8.18 Multilevel Data Model

Different MLS/DBMSs utilize different multilevel relational data models. These include (1) models with polyinstantiation, (2) models with security constraints, and (3) models based on schema design. The NGCR DBMS interface standards shall have the ability to handle different types of multilevel data models.

4.8.19 SQL Extensions

Different MLS/DBMSs propose differing extensions to SQL. These include extensions to support polyinstantiation and extensions to support security constraints. The NGCR DBMS interface standards shall provide the ability to support differing extensions.

4.8.20 OS Interface

The NGCR DBMS interface standards shall provide the ability to handle

any type of multilevel secure operation system or MLS/DBMS design.

4.8.21 Network Interface

The NGCR DBMS interface standards shall provide the ability to handle any type of distributed architecture.

4.8.22 Heterogeneity

The NGCR DBMS interface standards shall provide the ability to handle heterogeneity with respect to query languages, query processing and optimization, transaction processing, data models, and security policies.

4.8.23 Next-Generation MLS/DBMS

Next-generation MLS/DBMSs include secure object-oriented DBMSs, secure deductive DBMSs, and MLS/DBMSs which can handle multimedia data types as well as data processing in real time. The NGCR DBMS interface standards shall have the capability to handle different kinds of MLS/DBMSs, multimedia data types, and real-time transaction processing algorithms.

4.8.24 Trusted Database Interpretation

The NGCR DBMS interface standards shall have the ability to handle the evolution of the Trusted Database Interpretation.

4.9 DISWG's Advanced DBMS Requirements Class

"This category captures the directions in which DBMSs need to evolve in order to be able to support applications beyond traditional business data processing. Such applications tend to involve the management of complex data and rules. ... Much of

this advanced functionality is typically associated with *object-oriented DBMSs* and *knowledge base management systems*"[32]. Obviously, many of the OO requirements included here are matched and met. However, Open OODB falls short of the knowledge base requirements. Now, we evaluate the eighteen requirements of this class.

4.9.1 Persistent Objects

> *The NGCR DBMS interface standards shall provide database management support for persistent objects in accordance with the concept of object-oriented database management discussed [previously].*

Open OODB

Requirements. This is matched by R2 Persistence. R2 states, "an oodb must support persistent storage of object instances and classes supported by its object-oriented data model(s)"[69]. Since Open OODB *is* an OODB, it has much stronger requirements on persistent objects than does the more general DISWG. Subrequirements R2-1 through R2-8 deal with many important issues in object persistence.

Proposed Architecture. Open OODB's proposed architecture meets this requirement. Open OODB is modeled as a persistent, OO programming language [66]. Open OODB handles persistence with a separate Persistence PM, but mentions that persistence could be handled by a combination of other PMs.

Implementation. Open OODB's implementation allows this requirement to be met for reasons similar to those just discussed. The Persistence PM is the only fully modularized PM implemented in Open OODB [63]. It must be noted, however, that the low level storing and retrieving of objects is done by the Exodus storage manager.

4.9.2 Object Identifiers

The NGCR DBMS interface standards shall provide the capability to associate OIDs with objects and the capability to establish a relationship between objects by reference to an OID.

Open OODB

Requirements. Open OODB has a matching requirement. Under functional requirement R1 OO Data Model, R1-1 states that the OO data model must support R1-1-1: object identity and inter-object references. R1-1-15 requires that the inter-object reference scheme must be extensible. Also, under functional requirement R2 Persistence, R2-2 states that an OODB must "support object identity and interobject reference"[69]. Additional subrequirements under R2-2 strengthen the required support for OIDs.

Proposed architecture. Open OODB's proposed architecture meets this DISWG requirement. "Object identity is a crucial aspect of object-oriented systems"[68]. All proposed Open OODB modules can manipulate OIDs and references to objects. Many "behavioral extensions are modeled and/or implemented in terms of references to objects ..." and a "reference consists of an object identifier (OID) that names the object, and a mechanism for mapping from the OID to the object's state"[68]. Open OODB, however, never explicitly uses the term "relationship" that is in the DISWG requirement.

Implementation. This is met. As just noted, OIDs are basic to the OO paradigm and many objects in Open OODB are reached through references. The DISWG definition also mentions relationships. In Open OODB, references to objects are themselves objects. These reference objects could be incorporated into the relationship objects as presented in RTSORAC [73]. Relationships can be considered to be generalizations of

references. That is, all references are relationships, but relationships are much more than just references.

4.9.3 Collection Data Type Constructors

The NGCR DBMS interface standards shall provide collection data type constructors that enable users to define collection-valued attributes.

Open OODB

Requirements. Open OODB has a matching requirement. Under R1 OO Data Model, R1-1-11, states that the data model must support the following: "collection, complex objects, composite objects ..."[69]. R1-1-11-1 further refines collections as sets, lists, sequences and arrays, etc. Also, under the R6 Query Capability functional requirement, R6-1 states that OODBs must support "a set data type and user-defined sets of class-compatible object instances"[69].

Proposed Architecture. Open OODB's proposed architecture meets this requirement.

Implementation. Open OODB's implementation meets this requirement. Open OODB provides support for sets which are implemented as linked lists.

4.9.4 User-Defined Data Types

The NGCR DBMS interface standards shall provide a mechanism that enables users to define their own data types.

Open OODB

Requirements. There is no explicit, matching requirement. However, since Open OODB is an extension of C++, this requirement is implicitly matched.

Proposed Architecture. This is met for the same reason presented above.

Implementation. This is met because Open OODB is an extension to C++.

4.9.5 Sorting Order

The NGCR DBMS interface standards shall provide the capability to specify a sorting order, at least one of a set of pre-defined sorting orders, for given data types, including at least the traditional character data type.

After initially reading this requirement, it was hard to determine exactly what DISWG meant. Does sorting order imply indexing? However, this must not be the case because the term "indexing" is well defined in database jargon. Indeed, it is interesting to note that "indexing" is never mentioned in the DISWG requirements. This must be an oversight because objects are usually referenced through some sort of index [85].

Open OODB

Requirements. There is no matching Open OODB requirement.

Proposed Architecture. This is met. Although Open OODB provides no special functionalities to do sorting, Open OODB is an extension to C++. Therefore, sorting routines could be written as needed.

Implementation. This DISWG requirement is met for the reason just stated.

4.9.6 Temporal Data

The NGCR DBMS interface standards shall provide for management of temporal data, i.e., data augmented by a time point (or interval) at which its value applies.

Open OODB

Requirements. There is no directly matching requirement. However, under R5 Data Dictionary, R5-1-6 requires support for timestamps.

Proposed Architecture. This requirement is unmet. The Open OODB project makes no mention of support for temporal data.

Implementation. This requirement is unmet. Temporal data is not a concern of Open OODB. In [84], it is pointed out that no current commercial system supports temporal data in a general way. Also, in [74], it is observed that "there is still no common definition for an object data model and no common consensus over what features are expected in an object database system, let alone a temporal object database system." Work is underway to develop both a Temporal SQL, and a list of common definitions for temporal databases [41].

However, it would be straightforward to add an attribute to an object expressing the time at which it was written. For instance, in the RTSORAC model [73], an attribute of an object is represented as a tuple with four fields. More formally, an attribute is characterized by $< N_a, V, T, I >$. N_a is the attributes name, V its value and T a timestamp used to check the temporal consistency of V, the value. I, the tuple's last field, represents imprecision and is discussed in a subsequent DISWG requirement for uncertain data. However, simply expressing a time attribute is one thing, efficiently using it in the context of a temporal database is another. There are many open questions in this area [74].

4.9.7 Spatial Data

The NGCR DBMS interface standards shall provide for management of spatial data, i.e., data augmented by a spatial location at which its corresponding object exists.

Open OODB

Requirements. There is no matching requirement.

Proposed Architecture. This facility is not included in Open OODB.

Implementation. Open OODB's implementation does not meet this requirement. It would be straightforward to include attributes for spatial data in an object. Manipulation of those spatial attributes is, however, another matter. Efficient manipulation of spatial data has been identified as a primary concern for next-generation databases [84].

Indeed, a major motivating factor behind the development of OODBs is the ability to handle complex, spatial data such as maps. A map could be modeled as a collection of smaller maps, or objects. Each object would have spatial data attributes to tell how to put them together to form the larger map. New methods to organize and query data must be developed to handle spatial data [84]. Currently, such methods are handled on an ad hoc basis [35].

4.9.8 Uncertain Data

> *The NGCR DBMS interface standards shall provide for the management of uncertain data, i.e., data augmented by an indication of the likelihood that its value is accurate.*

Open OODB

Requirements. There is no matching requirement.

Proposed Architecture. This is not a design issue for Open OODB.

Implementation. This is unmet. Research on uncertainty in the relational model has focused on *fuzzy* sets [52]. However, "Reasoning under uncertainty, especially when a conclusion must be derived ..." needs further research [84].

Uncertain data is an area of research at URI and will most likely be included in any extension to Open OODB. As discussed previously, attributes in the RTSORAC model have an imprecision field, I, which is used "to store the amount of imprecision associated with the attribute ..."[73]. Furthermore, it is proven in [16] that this imprecision can be bounded. Extending Open OODB with this feature of the RTSORAC model would allow this DISWG requirement to be met.

4.9.9 Derived Attributes

> *The NGCR DBMS interface standards shall provide a mechanism that enables users to define derived attributes.*

A derived attribute is an "attribute whose value is defined procedurally or declaratively rather than stored explicitly"[91]. For example, instead of representing a person's age as an explicit value, the age could be represented as a function that subtracts the person's birth date from the current date. DISWG lists derived attributes as one of the distinguishing characteristics of an OODB.

Open OODB

Requirements. There is no matching requirement. However, as stated above, derived attributes are a characteristic of an OODB. Thus, we can say that Open OODB implicitly matches the requirement because it has the necessary characteristic.

Proposed Architecture. Open OODB's proposed architecture meets this DISWG requirement for the reason stated above.

Implementation. This is met. Open OODB is an extension of C++, which allows for the declaration and procedural derivation of attributes.

4.9.10 Composite Objects

The NGCR DBMS interface standards shall support composite objects.

A composite object is an object that contains other objects [32]. This definition can be refined to say that a composite object is an object with a hierarchy of exclusive, component objects [8]. "A component of a composite object may have at most one containing object"[43]. A composite object can be used to model the "part-of" or ownership concepts. It "may be viewed as a scope containing several contained objects"[60]. DISWG views composite object support as a distinguishing characteristic of an OODB [32]. Open OODB states that, "Objects that are considered atomic by an application are often realized as a composite of many other objects. Such a composite object is called a configuration"[65].

Open OODB

Requirements. This requirement is matched. Under R1 OO Data Model, R1-1-11 states that an OODB may support composite objects.

Proposed Architecture. Open OODB's proposed architecture allows this requirement to be met. Support for composite objects is basic to the OO paradigm, and Open OODB is based on the OO paradigm. Also, Open OODB's Change Management PM provides support for configurations, some of which are composite objects.

Implementation. This is partially met. When an object is persisted in Open OODB, objects pointed to by the persisted object will also be persisted. This is done to a depth of one level. For instance, consider an object A which points to an

object B and object B points to an object C. If object A is persisted, object B will also be persisted, but object C will not be persisted. However, there are no other functionalities in Open OODB geared specifically towards manipulating composite objects.

ObjectStore handles composite, or complex objects with relationships [48]. In ObjectStore's relationships, if one object points to another, there is also an inverse pointer. The relationship is responsible for maintaining the integrity of these pointers when faced with a deletion, etc. Thus, in ObjectStore, the ability to handle complex objects is similar to the problem of maintaining referential integrity. ORION superimposes an IS-PART-OF relationship over the appropriate references in a composite object [44]. In ORION, "direct support for composite objects as a unit for one type of semantic integrity, physical clustering, and locking" is provided [43]. In the extended relational system Starburst, complex objects can be treated in a manner similar to that used in treating BLOBs. That is, the whole complex object is stored contiguously with a Manager that maintains pointers into the complex object. Finally, composite, or complex, objects could certainly be modeled using the relationship facilities of RTSORAC [73].

4.9.11 Object Type Hierarchies

> *The NGCR DBMS interface standards shall implement inheritance and provide mechanisms for the establishment of object type hierarchies based on inheritance.*

DISWG recognizes that hierarchies and inheritance are intrinsic to OODBs [32].

Open OODB

Requirements. This requirement is matched by Open OODB. Under R1 OO Data Model, R1-1-5 requires that an OODB support inheritance. The fact that Open

OODB is an extension to C++ implies that this requirement is matched, since C++ supports both inheritance and hierarchies.

Proposed Architecture. Open OODB's proposed architecture meets this DISWG requirement. As just mentioned, support for hierarchies and inheritance is characteristic of OODBs.

Implementation. Open OODB's implementation meets this requirement due the nature of the OO paradigm.

4.9.12 Object Encapsulation

> *The NGCR DBMS interface standards shall provide a mechanism to associate a procedure with an object in support of object encapsulation.*

Object encapsulation is the "hiding of attributes and implementation details of an object from the client (or user) of the object"[32]. Object encapsulation is a distinguishing characteristic of an OODB.

Open OODB

Requirements. This is matched under the R1 OO Data Model requirement. R1-1-2 states that an OODB must support encapsulation.

Proposed Architecture. This requirement is met by the proposed architecture.

Implementation. This requirement is met by the implementation. Encapsulation is part of the basic nature of the OO paradigm.

4.9.13 Versions and Configurations

The NGCR DBMS interface standards shall support versions and configurations.

A version is "a variant of the initial value of an object"[60]. DISWG says that a version is "a mechanism that can be used for concurrency control, recovery control and configuration management"[91]. A configuration is "an enhancement to versions. A configuration is a group of mutually consistent versions of related objects"[32]. Both versions and configurations are basic to OODBs [32].

Open OODB

Requirements. This requirement is matched by the R7 Change Mangement functional requirement which states that an OODB must support both R7-1 version management and R7-2 configuration management. Since versions and configurations are basic to the OO paradigm, Open OODB's requirements for their support are strengthened by subrequirements. For instance, management of both versions and configurations must be able to be turned on or off and must be able to deal with persistent and transient objects. Also, under R5 Data Dictionary, R5-1-7 requires that the data dictionary contain configuration information and R5-4 requires that it maintain information on configuration relationships.

Proposed Architecture. Open OODB's proposed architecture allows this requirement to be met. The Change Management PM is responsible for versions and configurations. Versions and configurations are extensions to C++. This means that when either are used, they cause a virtual event.[27] This virtual event is the invocation of the Change Management PM, which then chooses the next appropriate action(s) [68].

[27] Remember that a virtual event seamlessly replaces a direct event when an extension is detected by a sentry. Refer to the section describing Open OODB's computational model for more details.

Implementation. This is unmet because the Change Management PM is not implemented. Indeed, most OODBs do not support versioning [46] and "While there has been much discussion and many proposals for proper version and configuration models in different domains, little has been implemented"[84]. Open OODB has worked towards incorporating these facilities [59]. Typically, versions in an OODB can be represented by disconnected, directed, acyclic graphs with the direction representing successor/predecessor relationships and the nodes representing different versions [4]. The problem then becomes how to reconcile the differences between the versions in the graph. ObjectStore takes a *merge* approach which lets the user define what to do on an ad hoc basis [48]. ORION treats versions somewhat differently. In ORION, there is a distinction between *working* and *transient* versions [43]. A working version is stable and can not be updated. A transient version may be deleted, updated or promoted to a working version. It is worth noting that ORION's designers are not satisfied with their implementation of version management.

Space is a also major consideration in version management because the version graphs can grow large. Most systems allow for users to decide whether a particular object can be versioned, instead of allowing all objects to be versioned. Versions pose complicated problems and are an active research area. Since configurations are refinements of versions, the same arguments apply to them.

4.9.14 Archival Storage

The NGCR DBMS interface standards shall provide management of archival storage.

Open OODB

Requirements. This requirement is matched by a subrequirement of the R16 Industrial Strength meta requirement. R16-2-1 states that the database administrator

utilities must include backup, restore and archiving capabilities.

Proposed Architecture. This is unmet as there is no mention of the management of archival storage in Open OODB's architecture literature. Note that OMG includes an "archival mapping" service to handle mappings in and out of archival storage. At present, OMG is not sure if archival storage should be a separate service, or a component of persistence or change management.

Implementation. This is unmet. There is no special functionality in Open OODB to handle archival storage. This would have to be handled by a systems administrator in the same way any files are backed up. In [85], it is pointed out that support for archival storage is rare in current OODBs. Archival storage is essential for a system to recover from disk crashes. In [43], it is noted that ORION does not provide support for recovery from disk crashes because it does not provide support for archival dumping.

OMG is working towards defining an archival mapping service and Open OODB could use their ideas as a basis for implementation. Note that another C++ based DBMS, ObjectStore, "provides backup to long-term storage media such as tapes, allowing full dumps as well as continuous archive logging"[48]. A main consideration here is to determine an efficient archival storage organization and representation. For instance, consider a three stage memory hierarchy: main memory, disk memory and archive memory. Objects in main memory use pointers and can be efficiently organized using AVL trees. Objects on disk use OIDs and are better accessed using B++ trees. Similar issues must be resolved for archival storage.

4.9.15 Schema Evolution

The NGCR DBMS interface standards shall support schema evolution. That is, they shall provide facilities that enable users "to modify a schema with minimum impact on existing applications"[Cattell 91]. Such facilities

may include a "run-time capability to rename, add, and drop properties of object types, and to add and drop object types themselves"[Cattell 91].

Open OODB

Requirements. There is a matching requirement under the R7 Change Management functional requirement. R7-4 requires that schema evolution must be supported. Open OODB defines schema evolution as the "process of manipulating the Data Dictionary to add, remove or reconfigure existing class definitions"[60]. A subrequirement of the R5 Data Dictionary functional requirement, R5-1-6, states that the Data Dictionary must be able to represent schema version information. A subrequirement of the R15 Performance meta requirement, R15-1-13, states that the cost of schema evolution must be measured.

Proposed Architecture. The proposed architecture would meet this DISWG requirement. Users using C++ could "manipulate schema meta data in the data dictionary"[69].

Implementation. This is not met. Most OODBs do not support schema evolution [46] and there is no clear cut path on how to incorporate schema evolution into an OODB [69]. Schema evolution is outside the current scope of Open OODB, but is an active research area. Indeed, an objective of the Open OODB project is to seamlessly add functionalities such as schema evolution [58]. Open OODB's designers have observed that schemas, as well as many other database aspects (i.e., platforms), are subject to change simply due to the passing of time [59].

Schema evolution is also listed as work in progress in ObjectStore, which like Open OODB, is an extension to C++ [48]. Another OODB, O_2, tackles the problem of class deletion, a subset of schema evolution, by not allowing the deletion if the class has any instances or if any other classes depend on the class. The developers

of ORION [43] present a more detailed look at schema evolution. For example, if a class C is deleted, any subclasses of C now become subclasses of C's former parent. Any method or attribute that was inherited from C would be dropped. All objects of class C are automatically deleted. If C were in the domain of another class, C's former parent would be inserted. Finally, all references to C must be modified by the user. That last step, involving undesirable user interaction, highlights the problems involved in implementing schema evolution.

A different problem surfaces in another area covered by schema evolution: object migration to a new class. We have already mentioned that Open OODB recognizes the difficulty in telling from where a reference came. If an object is to be deleted or to join a new class, all references to that object must be updated. ORION sees the solution to this problem as very expensive. The problems presented by schema evolution are complex, and we have only scratched the surface. However, ORION's techniques, or ones similar, are applicable to Open OODB.

4.9.16 Long Transactions

The NGCR DBMS interface standards shall support long transactions.

DISWG defines a long transaction is a "transaction representing a computation that may take up to hours, days or even longer to complete. Such transactions are incompatible with conventional transaction concurrency control and recovery control policies and mechanisms"[91]. Open OODB defines a long *duration* transaction as "a transaction or collection of transactions that survive system crashes and do not block short duration transactions"[60]. In an article by the designers of ObjectStore, long transactions are defined as, "extended editing sessions on private, checked-out versions ..."[48].

Open OODB

Requirements. There is a matching requirement under the R3 Concurrent Access functional requirement. R3-2 states that an OODB must support *some* of a list of functionalities. One member of that list is R3-2-5: long duration transactions.

Proposed Architecture. Open OODB's proposed architecture does not meet this requirement. However, nothing would preclude it from being met.

Implementation. Open OODB's implementation does not meet this requirement. Long transactions are not supported by most OODBs [46]. Open OODB states that support for long transactions is not fully implemented [69]. Whether a transaction is short or long, it must do the same thing: "ensure database consistency in the presence of simultaneous accesses to the system by multiple users and in the presence of system crashes"[46].

A key problem with long duration transactions is their potential to lock up resources for long durations. If long transactions are broken down into smaller transactions, like the *saga* approach, then recovery becomes more of a problem [18]. User-defined compensation routines have to be written as a solution. Another method proposes using triggers and rules as a way to modularize control flow and, thus, to help implement long transactions [18]. Extending Open OODB with RTSORAC could help with long transactions due to the potential to relax locking in RTSORAC.

4.9.17 Rule Processing

> *The NGCR DBMS interface standards shall support rule processing, including the enforcement of assertions, the initiation of triggers and alerters, and deductive query processing.*

Open OODB

Requirements. This is partially matched. Under R17 Other DBMS Requirements, R17-4 mentions, but does not require support for, rules based systems. This requirement is outside of Open OODB's current scope. Also, subrequirements of Open OODB's R1 OO Data Model functional requirement match this partially. Specifically, R1-1-14 requires support for integrity constraints and R1-1-14-3 rules, R1-1-14-2 assertions and invariant conditions, R1-1-14-6 triggers and R1-1-14-4 user-defined validation procedures. Under R9 Integrity and Recovery, R9-5 requires that an OODB to support integrity constraints. The R5 Data Dictionary requirement states that information about rules and assertions be represented in the data dictionary.

Proposed Architecture. This requirement is not met by Open OODB's proposed architecture. Rules based systems are currently out of Open OODB's scope.

Implementation. This requirement is not met by the implementation. Open OODB is uncertain of the amount of rework that would be required to meet this requirement [69] and rules processing is out of the scope of the current Open OODB project. In [83], the authors note that most systems already have "simple-minded" rules to support referential integrity and "special purpose" rules for protection and integrity constraints. The authors conclude that "It is clear to us that all DBMS's need a rules system."

4.9.18 Domain Specific Standards

The NGCR DBMS interface standards shall provide enhanced portability and interoperability of MCCR applications by adopting and endorsing domain-specific standards (at least for data representation or format) for those data types that are expected to be common in both MCCR applications and also in commercial applications. These data types include the

following:

- *Text*
- *Documents*
- *Graphics*
- *Image*
- *Audio*
- *Video*
- *Multimedia*
- *Geographic*

The NIST Application Portability Profile (APP) [NIST 93] lists several data interchange standards, covering many of these data types. In addition, [Gallagher and Sullivan 92] propose generic abstract data type packages for many of these data types. They are working with ANSI and ISO SQL standardization committees to determine the best process and mechanism for standardization.

Open OODB

Requirements. There is no matching requirement. However, a subrequirement of the R16 Industrial Strength meta requirement, R16-10, mentions standards. R16-10-1 requires an OODB to be available from multiple sources. R16-10-2 requires an OODB to be available as a X3/IEEE/ISO standard. R16-10-3 states that an OODB must be CAD Framework Initiative compliant. R16-10-4 requires that an OODB meet standards for Unix, SQL, C++ and Motif, etc.

Proposed Architecture. The proposed architecture does not meet this requirement. However, its modular nature would facilitate extending Open OODB to meet

a specific domain standard.

Implementation. The implementation does not meet this requirement. However, nothing would preclude Open OODB from meeting it in the future. Indeed, the development of OODBs has been motivated in large part by the need to efficiently manipulate multimedia data. "Objects can contain very large values, such as audio, video, or text"[14].

5 Conclusion

This thesis evaluated Open OODB with respect to DISWG's requirements. This evaluation was performed from three distinct perspectives: Open OODB's requirements, its proposed architecture and its implementation. For each DISWG requirement, we first determined if Open OODB had a matching requirement. Then, we decided if Open OODB's proposed architecture would meet the requirement. Finally, we evaluated Open OODB's implementation with respect to the requirement. If a requirement was found to be unmet, we included a brief discussion either on how Open OODB could be extended to meet the requirement or on the topic covered by the requirement.

In the rest of this conclusion, we summarize the evaluations of each of DISWG's requirements classes. These summaries follow the same evaluation format used throughout this thesis. However, instead of looking at each individual requirement, we look at each DISWG requirements class with respect to Open OODB's requirements, proposed architecture and implementation. We end with some overall observations of Open OODB.

5.1 DISWG's General Requirements Class

These are requirements that any computer system in general, not just databases, should meet. Example requirements include portability, different types of independence (i.e., hardware, operating system, etc.), modularity and extensibility.

Open OODB

Requirements. We found that about one half of these requirements are matched. The main reason that more are not matched is due, once again, to the main difference between DISWG and Open OODB. DISWG wants to accommodate all types of databases in its requirements, whereas Open OODB is a specific implementation. For instance, DISWG requires support for six different types of queries, but Open OODB only requires support for two types. Also, DISWG requires support for different data models, i.e., relational, network and hierarchical, but Open OODB supports the OO data model.

Proposed Architecture. We found that many of these are met due to the open, extensible nature of Open OODB. Requirements are unmet for mainly two reasons. The first is that they are outside the scope of Open OODB's current effort. For instance, there are two requirements for security in this class and security is not currently included in Open OODB. The second is due to the main difference between Open OODB and DISWG.

Implementation. We found that most of these are met. Most of these would be met by any database, which is DISWG's intent. Most of the unmet requirements are unmet for the two reasons just stated in the summary of the evaluations of Open OODB's proposed architecture.

5.3 DISWG's Distribution Requirements Class

These requirements are for a distributed database system which DISWG defines as multiple, distributed copies of a DBMS managing distributed databases. Open OODB does not fit this definition. Consequently, none of these requirements are matched or met. However, Open OODB is designed to be a distributed database. Also, the Open OODB project is currently working on an architecture using a CORBA-like

backplane that would fit this definition. Therefore, we evaluated these requirements as if they were for distributed databases. We assume that the same properties that would allow Open OODB to meet requirements for distributed databases would allow it to be extended to meet those for distributed database systems.

Open OODB

Requirements. We found that Open OODB would match most of these requirements if they were for distributed databases. Two unmatched DISWG requirements of note are for continuous operation and for network independence.

Proposed Architecture. We found that all of these requirements would be met if they were for distributed databases.

Implementation. We found that almost none of these requirements are met by Open OODB's current implementation, even when making the mentioned assumption. While Open OODB's proposed architecture allows for flexibility, in the implementation, concrete choices have to be made. For instance, Open OODB's modules have a proposed design that should be hardware independent. However, those modules are currently tied to the machine upon which they are implemented. If Open OODB were fully implemented as proposed, then these requirements would all be met in the context of a distributed database. Furthermore, Open OODB's extensible nature should allow it to be incorporated into a CORBA-like system, which would then meet these DISWG requirements for a distributed database system.

5.4 DISWG's Heterogeneity Requirements Class

These requirements are in an undeveloped state due to the current lack of understanding of the problems posed by heterogeneity. Also, these requirements highlight the difference between DISWG and Open OODB. DISWG has to allow for interoper-

ating, heterogeneous systems, while Open OODB is a particular system. Nonetheless, as just mentioned in the summary of DISWG's Distribution requirements class, the Open OODB project is working on incorporating Open OODB into a CORBA-like system. The other DBMSs in this system could be Open OODB implementations, which would allow DISWG's Distribution requirements to be met, or they could be heterogeneous DBMSs, which would allow these Heterogeneity requirements to be met.

Open OODB

Requirements. We found that Open OODB requires support for remote database access. Otherwise, these are unmatched.

Proposed Architecture. We found none of these are met. The proposed architecture, as it stands in the alpha release, has no special functionalities to handle interoperation with heterogeneous databases.

Implementation. These are unmet. Implementation of the previously discussed CORBA-like system would allow these to be met.

5.5 DISWG's Real-Time Requirements Class

These are all requirements that a next-generation, real-time database should meet. Since Open OODB is not real-time, it does not match or meet these requirements. At URI, we are developing RTSORAC: a next-generation, real-time OODB. In our evaluations of this class, we mentioned that extending Open OODB with RTSORAC features would allow for these requirements to be met.

Open OODB

Requirements. We found that these are all unmatched.

Proposed Architecture. We found these unmet.

Implementation. We found these are unmet with one exception: DISWG requires a compilable DML which Open OODB's C++ implementation meets.

5.6 DISWG's Fault Tolerance Requirements Class

These requirements are for fault management capabilities, an area of particular concern for the mission-critical database systems in the Navy. However, no DBMS currently meets these requirements. For these to be met in the future, there would have to be more cooperation between operating systems and DBMSs. We proposed the development of a Fault Tolerance Policy Manager in Open OODB to handle the DBMS's responsibilities. To handle the operating system's responsibilities, either UNIX would have to be extended to include fault tolerance capabilities, or Open OODB would have to be ported to a fault tolerant, operating system.

Open OODB

Requirements. We found that these requirements are unmatched. Open OODB does require that the "System must support recovery"[69]. However, that statement is too vague to say that any of DISWG's Fault Tolerance requirements are matched.

Proposed Architecture. We found that none of these are met. The proposed architecture has no special functionalities to handle fault tolerance. Open OODB relies on the Exodus storage manager for all of its recovery capabilities.

Implementation. We found that these are all unmet. We proposed a Fault Tolerance PM to handle Open OODB's fault tolerance responsibilities and noted that the underlying operating system must provide support.

5.7 DISWG's Integrity Requirements Class

These are requirements to ensure that the database provides users with correct information. The role of integrity in OODBs is an active research area and Open OODB contains no special functionalities to enforce integrity constraints. Consequently, we found that most of these requirements are unmatched and unmet. We noted that extending Open OODB with relationships as proposed in URI's RTSORAC model would allow most of these requirements to be met.

Open OODB

Requirements. We found that the majority of these are unmatched. However, Open OODB does require support for referential integrity, assertions and triggers.

Proposed Architecture. We found that these are unmet as Open OODB includes no functionalities to handle integrity constraints.

Implementation. We found that these are mostly unmet. However, OODBs appear to be a natural vehicle for the expression of integrity constraints. Extending Open OODB with relationships such as those proposed in RTSORAC would allow most of these to be met.

5.8 DISWG's Security Requirements Class

These are requirements that ensure that users do not gain unauthorized access to information. We noted that DISWG has deemphasized its Security requirements and that security is outside the scope of Open OODB. Therefore, we did not perform a detailed evaluation of DISWG's twenty Security requirements. Instead, we talked about security issues in general. Thus, there are no summaries of our findings on Open OODB's requirements, proposed architecture and implementation as there are for DISWG's eight other classes.

5.9 DISWG's Advanced Requirements Class

These requirements encompass the functionalities of non-traditional DBMSs such as OODBs and knowledge base management systems. Support is required for many of the basic characteristics of OODBs. For instance, support is required for OIDs, object encapsulation, composite objects and versions, etc. However, this class is very much a mixed-bag. There are also requirements for support of temporal data, spatial data, uncertain data, archiving, long transactions and rules. Thus, its hard to imagine any database matching or meeting substantially more than one half of these requirements.

Open OODB

Requirements. We found that Open OODB matches the requirements that directly pertain to OODBs. Additionally, Open OODB has matching requirements for archiving, long transactions and rules.

Proposed Architecture. We found that the requirements that deal with OODBs are met. None of the others are met.

Implementation. We found that the implementation meets most of the requirements that relate to OODBs. The unmet OODB requirements include support for versions, configurations and schema evolution. These are all active research areas at the Open OODB project. The other, non-OODB requirements, are all unmet. In our evaluations, we briefly discussed the main issues of topics such as temporal data, spatial data, archiving and long transactions. These are all active research areas.

5.10 Overall Observations of Open OODB

Open OODB is the leading effort to develop a next-generation OODB. We have identified the following overall strengths and weaknesses. Open OODB's main weakness is that its all encompassing scope has impeded implementation. It is hard to determine

exactly how all the many, different modules should interact. Also, it is very difficult to decide which functionalities deserve separate modules and which functionalities may coexist in a module. Another weakness is Open OODB's dependence on Exodus for transaction control, distribution control and recovery. This dependency has caused Open OODB to omit these important functionalities from its current implementation.

Open OODB's main strength is the thought put behind its proposed architecture. Its extensible, modular design with well defined functionalities allows it to be customized for special purposes. Much of this strength is derived from the basic nature of the OO paradigm. In an OO design, well defined interfaces, modularity and object encapsulation come almost for free. Due to this strength, URI is finding Open OODB to be a convenient testbed for the extensions we wish to make.

References

[1] Abbadi, A., D. Skeen and F. Cristian. An Efficient, Fault-tolerant Protocol for Replicated Data Management. in [87].

[2] Abbott, R. and H. Garcia-Molina. Scheduling Real-time Transactions: A Performance Evaluation. *ACM Transactions on Database Systems*, Vol. 17, No. 3, 1992.

[3] Abiteboul, S. and A. Bonner. Objects and Views. *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data*, Vol. 20, Issue 2, 1991.

[4] Ahmed, R. and S. Navathe. Version Management of Composite Objects in CAD Databases. *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data*, Vol. 20, Issue 2, 1991.

[5] Atkinson, M., et al. The Object-Oriented Database System Manifesto. in [87].

[6] Babaoğu, Ő. Fault Tolerant Computing Based on Mach. *ACM Operating Systems Review*, Vol. 24, No. 1, 1990.

[7] Bancilhon, F. and W. Kim. Object-Oriented Database Systems: In Transition. *SIGMOD Record*, Vol. 19, No. 4, 1990.

[8] Banerjee, J., H.Chou, J. Garza, W. Kim, D. Woelk, N. Ballou and H. Kim. Data Model Issues for Object-Oriented Applications. in [87].

[9] Barbara, D. Extending the Scope of Database Services. *SIGMOD Record*, Vol. 22, No. 1, 1993.

[10] Bertino, E. Data Hiding and Security in Object-oriented Databases. *Proceedings of the Eighth IEEE International Conference on Data Engineering*, 1992.

[11] Borg, A., W. Blau, W. Graetcsh, F. Herrmann and W. Oberle. Fault Tolerance under UNIX. *ACM Operating Systems Review*, Vol. 7, No. 1, 1989.

[12] Breitbart Y. Multidatabase Interoperability. *SIGMOD Record*, Vol. 19, No. 3, 1990.

[13] Butterworth, P., A. Otis and J. Stein. The GemStone Object Database Management System. *Communications of the ACM*, Vol. 34, No. 10, 1991.

[14] Cattell, R. Introduction to Special Section on Next Generation Database Systems. *Communications of the ACM*, Vol. 34, No. 10, 1991.

[15] Cheng, J. and A. Hurson. Effective Clustering of Complex Objects in Object-Oriented Databases. *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data*, Vol. 20, Issue 2, 1991.

[16] Cingiser DiPippo, L. and Fay Wolfe, V. Distributed Object-Based Real-Time Concurrency Control with Bounded Imprecision. *University of Rhode Island, Technical Report TR93-227*, 1994.

[17] Chrysanthis, P. and K. Ramamritham. ACTA: A Framework for Specifying and Reasoning about Transaction Structure and Behavior. in [87].

[18] Dayal, U., M. Hsu and R. Ladin. Organizing Long-Running Activities with Triggers and Transactions. in [87].

[19] Decorte, G., A. Eiger, D. Kroenke and T. Kyte. An Object-Oriented Model for Capturing Data Semantics. *Proceedings of the Eighth IEEE International Conference on Data Engineering*, 1992.

[20] Demurjian, S., M.-Y. Hu, T. Ting and D. Kleinman. Towards an Authorization Mechanism for User-Role Based Security in an Object-Oriented Design Model.

Proceedings of the 1993 Phoenix Conference on Computers and Communications, 1993.

[21] de Paula, E. and M. Nelson. Clustering in Object-Oriented Databases. *OOPS Messenger*, Vol. 3, No. 3, 1992.

[22] Deux, O., et al. The O_2 System. *IEEE Transactions on Knowledge and Data Engineering*, March 1990.

[23] Doherty, M. Implementing Relationships in an Object-Oriented Database. *Master's Thesis, University of Rhode Island*, 1992.

[24] P. Drew, R. King, D. McLeod, M. Rusinkiewicz and A. Silbershatz. Report of the Workshop on Semantic Heterogeneity and Interoperation in Multidatabase Systems. *SIGMOD Record*, Vol. 22, No. 3, 1993.

[25] Eliasen, F. and R. Karlsen. Interoperability and Object ID. *SIGMOD Record*, Vol. 20, No. 4, 1991.

[26] Elmasri, R. and S. Navathe. *Fundamentals of Database Systems.* Benjamin/Cummings Publishing Co., Inc., Redwood City, Ca., 1989.

[27] Fang, D., S. Ghanderharizadeh, D. McLeod and A. Si. The Design Implementation and Evaluation of an Object-based Sharing Mechanism for Federated Database Systems. *Proceedings of the Ninth IEEE International Conference on Data Engineering*, 1993.

[28] Fourtier, P. and J. Prichard. Concepts for a Real-time Structured Database Query Language. *Unpublished, University of Rhode Island*, 1994.

[29] French, J., A. Jones and J. Pfaltz. NSF Workshop on Scientific Database Management. *SIGMOD Record*, Vol. 19, No. 4, 1990.

[30] Gal-Oz, N., E. Gudes and E. Fernandez. A Model of Methods Access Authorization in Object-oriented Databases. *Proceedings of the 19th IEEE Very Large Databases Conference*, 1993.

[31] Garcia-Molina, H. and B. Lindsay. Research Directions for Distributed Databases. *SIGMOD Record*, Vol. 19, No. 4, 1990.

[32] Gordon, K. Requirements for Military Database Management Systems. *Next-Generation Computer Resources (NGCR) Technical Document No. 010 ver. 1.0*, NGCR SPAWAR 331-2, 2451, Crystal Dr., Alexandria, Va. 22245, November 15, 1993.

[33] Graefe, G. and W. McKenna. The Volcano Optimizer Generator: Extensibility and Efficient Search. *Proceedings of the Ninth IEEE International Conference on Transactions on Data Engineering*, 1993.

[34] Gruenwald, L. and S. Liu. A Performance Study of Concurrency Control in a Real-Time Main Memory Database System. *SIGMOD Record*, Vol. 22, No. 4, 1993.

[35] Guenther, O. and A. Buchmann. Research Issues in Spatial Databases. *SIGMOD Record*, Vol. 19, No. 4, 1990.

[36] Hanson, E., T. Harvey and M. Roth. Experiences in DBMS Implementation Using an Object-oriented Persistent Programming Language and a Database Toolkit. *Object-Oriented Programming: Systems, Languages and Applications*, 1991.

[37] Hung, S. and K. Lam. Locking Protocols for Concurrency Control in Real-Time Database Systems. *SIGMOD Record*, Vol. 21, No. 4, 1992.

[38] Ishikawa, H. and K. Kubota. An Active Object-Oriented Database: A Multi-Paradigm Approach to Constraint Management. *Proceedings of the 19th Very Large Databases Conference*, 1993.

[39] Jagadish, H., and X. Qian. Integrity Maintenance in an Object-Oriented Database. *Proceedings of the 18th Very Large Databases Conference*, 1992.

[40] Jajodia, S. and R. Sandhu. Database Security: Current Status and Key Issues. *SIGMOD Record*, Vol. 19, No. 4, 1990.

[41] Jensen, C. et al. A Glossary of Temporal Database Concepts. *SIGMOD Record*, Vol. 21, No. 3, 1992.

[42] Kilov, H. A Review of Object-oriented Papers. *SIGMOD Record*, Vol. 18, No. 4, 1989.

[43] Kim, W., J. F. Garza, N. Ballou and D. Woelk. Architecture of the Orion Next Generation Database System. *IEEE Transactions on Knowledge and Data Engineering*, March 1990.

[44] Kim, W., E. Bertino and J. Garza. Composite Objects Revisited. *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, Volume 18, Number 2, 1989.

[45] Kim, W., N. Ballou, J. Garza and D. Woelk. A Distributed Object-Oriented Database System Supporting Shared and Private Databases. *ACM Transactions on Information Systems*, Vol. 9, No. 1, 1991.

[46] Kim, W. Object-Oriented Database Systems: Promises, Reality, and Future. *Proceedings of the Nineteenth International Conference on Very Large Data Bases*, 1993.

[47] Krishnamurthy, E. and A. Mcguffin. On the Design nd Implementation of Secure Database Transactions. *SIGSAC Review*, Vol. 10, Nos. 2 & 3, 1992.

[48] Lamb, C., G. Landis, J. Orenstein and D. Weintreb. The ObjectStore Database System. *Communications of the ACM*, Vol. 34, No. 10, 1991.

[49] Lohman, G., B. Lindsay, H. Pirahesh and K. B. Schiefer. Extensions to Starburst: Object Types, Functions and Rules. *Communications of the ACM*, Vol. 34, No. 10, 1991.

[50] Lunt, T. and E. Fernandez. Database Security. *SIGMOD Record*, Vol. 19, No. 4, 1990.

[51] Mohan, C., D. Haderle, B. Lindsay, H. Pirahesh and P. Schwarz. ARIES: A Transaction Recovery Method Supporting Fine-granularity Locking and Partial Rollbacks Using Write-ahead Logging. in [87].

[52] Motro, A. Accommodating Imprecision in Database Systems: Issues and Solutions. *SIGMOD Record*, Vol. 19, No. 4, 1990.

[53] Mourad, A., W. Fuchs and D. Saab. Database Recovery Using Redundant Disk Arrays. *Proceedings of the Eighth IEEE International Conference on Data Engineering*, 1992.

[54] Nicol, J., C. Thomas, T. Wilkes and F. Manola. Object Orientation in Heterogeneous Distributed Computing Systems. *IEEE Computer*, June 1993.

[55] Nyanchama, M. and S. Osborn. Role-Based Security, Object-oriented Databases & Separation of Duty. *SIGMOD Record*, Vol. 22, No. 4, 1993.

[56] Olivier, M. and S. Von Solms. A Taxonomy for Secure Object-Oriented Databases. *ACM Transactions on Database Systems*, Vol. 19, No. 1, 1994.

[57] Object Management Group, Inc. Object Services Architecture, Revision 6.0, Document 92.8.4. *Object Management Group, Inc., Framingham Corporate Center, 492 Old Connecticut Path, Framingham, Massachusetts, 01701-4568*, August, 1992.

[58] Open OODB Project. Open OODB Executive Summary Release 0.2 (Alpha). *Texas Instruments, Inc., P.O. Box 655474, M/S 238, Dallas, Texas, 75265*, 1993.

[59] Open OODB Project. Open OODB Technical Overview Release 0.2 (Alpha). *Texas Instruments, Inc., P.O. Box 655474, M/S 238, Dallas, Texas, 75265*, 1993.

[60] Open OODB Project. Open OODB Glossary. *Texas Instruments, Inc., P.O. Box 655474, M/S 238, Dallas, Texas, 75265*, 1993.

[61] Open OODB Project. Open OODB C++ API User Manual Release 0.2 (Alpha). *Texas Instruments, Inc., P.O. Box 655474, M/S 238, Dallas, Texas, 75265*, 1993.

[62] Open OODB Project. Open OODB Query Language User Manual Release 0.2 (Alpha). *Texas Instruments, Inc., P.O. Box 655474, M/S 238, Dallas, Texas, 75265*, 1993.

[63] Open OODB Project. Open OODB Release Notes Release 0.2 (Alpha). *Texas Instruments, Inc., P.O. Box 655474, M/S 238, Dallas, Texas, 75265*, 1993.

[64] Open OODB Project. Open OODB Meta Architecture Document. *Texas Instruments, Inc., P.O. Box 655474, M/S 238, Dallas, Texas, 75265*, 1993.

[65] Open OODB Project. Open OODB System Architecture Document. *Texas Instruments, Inc., P.O. Box 655474, M/S 238, Dallas, Texas, 75265*, 1993.

[66] Open OODB Project. Open OODB Architecture Specifications Document. *Texas Instruments, Inc., P.O. Box 655474, M/S 238, Dallas, Texas, 75265*, 1993.

[67] Open OODB Project. ARPA Open Object-Oriented Database Architecture Specification. *Texas Instruments, Inc., P.O. Box 655474, M/S 238, Dallas, Texas, 75265,* 1993.

[68] Open OODB Project. ARPA Open Object-Oriented Database Module Specification Meta Architecture Support. *Texas Instruments, Inc., P.O. Box 655474, M/S 238, Dallas, Texas, 75265,* 1993.

[69] Open OODB Project. Open OODB Requirements Document. *Texas Instruments, Inc., P.O. Box 655474, M/S 238, Dallas, Texas, 75265,* 1993.

[70] Open OODB Project. The Open Object-Oriented Database: Obtaining Database Functionality by Extension. *Readings in Object-Oriented Systems and Applications.*, IEEE Press, 1993.

[71] Özsu, M., U. Dayal and P. Valduriez. WORKSHOP REPORT International Workshop on Distributed Object Management. *SIGMOD Record*, Vol. 22, No. 1, 1993.

[72] Pangalos, G. Database Systems Security. *University of Rhode Island, Technical Report TR93-225*, 1993,

[73] Peckham, J., V. Fay Wolfe, J. Prichard and L. Cingiser DiPippo. RTSORAC: Design of a Real-Time Object-Oriented Database System. *University of Rhode Island, Department of Computer Science, Technical Report 94-231*, 1994.

[74] Pissinou, N., K. Makki and Y. Yesha. On Temporal Modeling in the Context of Object Databases. *SIGMOD Record*, Vol. 22, No. 3, 1993.

[75] Ramamritham, K. Real-Time Databases. *International Journal of Distributed and Parallel Databases*, 1992.

[76] Rangarajan, S., S. Setia and S. Tripathi. A Fault-Tolerant Algorithm for Replicated Data Management. *Proceedings of the Eighth IEEE International Conference on Data Engineering*, 1992.

[77] Saltor, F., M. Castellanos and M. Garcia-Solaco. On Canonical Models for Federated DBs. *SIGMOD Record*, Vol. 20, No. 4, 1991.

[78] Scheuermann, P., et al. Report on the Workshop on Heterogeneous Database Systems. *SIGMOD Record*, Vol. 19, No. 4, 1990.

[79] Senerchia, J. A POSIX-Compliant Dynamic Real-Time Scheduler. *Master's Thesis, University of Rhode Island*, 1993.

[80] Sheth, A., and J. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Survey*, Vol. 22, No. 3, 1990.

[81] Sreewastav, K. Prototype of the Object Query Language (OQL): an Associative Query Facility for Zeitgeist. *Texas Instruments, Inc.*, 1990.

[82] Stonebraker, M., et al. Third Generation Database System Manifesto. *SIGMOD Record*, Vol. 19, No. 3, 1990.

[83] Stonebraker, M., and G. Kemnitz. The POSTGRES Next-Generation Database Management System. *IEEE Transactions on Knowledge and Data Engineering*, March 1990.

[84] Stonebraker, M., A. Silbershatz and J. Ullman. Database Systems: Achievements and Opportunities. *Communications of the ACM*, Vol. 34, No. 10, 1991.

[85] Stonebraker, M. Managing Persistent Objects in a Multi-Level Store. *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data*, Volume 20, Issue 2, 1991.

[86] Stonebraker, M. and M. Olson. Large Object Support in POSTGRES. *Proceedings of the Ninth IEEE International Conference on Data Engineering*, 1993.

[87] Stonebraker, M., editor. *Readings in Database Systems*, Morgan Kaufmann Publishers, San Mateo, California, 1994.

[88] Stonebraker, M. Operating System Support for Database Management. in [87].

[89] Thuraisingham, B. Secure Interoperability of Trusted Database Management Systems. *SIGSAC Review*, Vol. 10, Nos. 2 & 3, 1992.

[90] Ulusoy, Ő. Current Research on Real-time Databases. *SIGMOD Record*, Vol. 21, No. 4, 1992.

[91] United States Navy. NGCR DISWG Draft Glossary and Acronyms. *NGCR DISWG Requirements Subgroup*, June 17, 1993.

[92] United States Navy. POSIX Delta Document for the Next-Generation Computer Resources Operating System Interface Standard (Version 2). *NUWC-NPT Technical Document 10,076*, June 1, 1992.

[93] Urban, S., A. Karadimce and R. Nannapaneni. The Implementation and Evaluation of Integrity Maintenance Rules in an Object-Oriented Database. *Proceedings of the Eighth IEEE International Conference on Data Engineering*, 1992.

[94] Wells, D., J. Blakeley and C. Thompson. Architecture of an Open Object-Oriented Database Management System. *IEEE Computer*, 1992.

[95] Wilkinson, K., P. Lyngbaik and W. Hasan. The Iris Architecture and Implementation. *IEEE Transactions on Knowledge and Data Engineering*, March 1990.

[96] Wolfe, V., L. Cingiser and J. Peckham. A Model for Real-Time Object-Oriented Databases. *Proceedings of the Tenth IEEE Workshop on Real-Time Operating Systems and Software*, 1993.

[97] Wright, M. Security Services in the OSI Reference Model. *SIGMOD Review*, Vol. 10, No. 1, 1992.

[98] Yu, P., K. Wu, K. Lin and S. Son. On Real-Time Databases: Concurrency Control and Scheduling. *Proceedings of the IEEE*, Vol. 82, No. 1, 1994.

Bibliography

Abbadi, A., D. Skeen and F. Cristian. An Efficient, Fault-tolerant Protocol for Replicated Data Management. in [87].

Abbott, R. and H. Garcia-Molina. Scheduling Real-time Transactions: A Performance Evaluation. *ACM Transactions on Database Systems*, Vol. 17, No. 3, 1992.

Abiteboul, S. and A. Bonner. Objects and Views. *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data*, Vol. 20, Issue 2, 1991.

Ahmed, R. and S. Navathe. Version Management of Composite Objects in CAD Databases. *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data*, Vol. 20, Issue 2, 1991.

Atkinson, M., et al. The Object-Oriented Database System Manifesto. in [87].

Babaoğu, Ő. Fault Tolerant Computing Based on Mach. *ACM Operating Systems Review*, Vol. 24, No. 1, 1990.

Bancilhon, F. and W. Kim. Object-Oriented Database Systems: In Transition. *SIGMOD Record*, Vol. 19, No. 4, 1990.

Banerjee, J., H.Chou, J. Garza, W. Kim, D. Woelk, N. Ballou and H. Kim. Data Model Issues for Object-Oriented Applications. in [87].

Barbara, D. Extending the Scope of Database Services. *SIGMOD Record*, Vol. 22, No. 1, 1993.

Bertino, E. Data Hiding and Security in Object-oriented Databases. *Proceedings of the Eighth IEEE International Conference on Data Engineering*, 1992.

Borg, A., W. Blau, W. Graetcsh, F. Herrmann and W. Oberle. Fault Tolerance under UNIX. *ACM Operating Systems Review*, Vol. 7, No. 1, 1989.

Breitbart Y. Multidatabase Interoperability. *SIGMOD Record*, Vol. 19, No. 3, 1990.

Butterworth, P., A. Otis and J. Stein. The GemStone Object Database Management System. *Communications of the ACM*, Vol. 34, No. 10, 1991.

Cattell, R. Introduction to Special Section on Next Generation Database Systems. *Communications of the ACM*, Vol. 34, No. 10, 1991.

Cheng, J. and A. Hurson. Effective Clustering of Complex Objects in Object-Oriented Databases. *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data*, Vol. 20, Issue 2, 1991.

Cingiser DiPippo, L. and Fay Wolfe, V. Distributed Object-Based Real-Time Concurrency Control with Bounded Imprecision. *University of Rhode Island, Technical Report TR93-227*, 1994.

Chrysanthis, P. and K. Ramamritham. ACTA: A Framework for Specifying and Reasoning about Transaction Structure and Behavior. in [87].

Dayal, U., M. Hsu and R. Ladin. Organizing Long-Running Activities with Triggers and Transactions. in [87].

Decorte, G., A. Eiger, D. Kroenke and T. Kyte. An Object-Oriented Model for Capturing Data Semantics. *Proceedings of the Eighth IEEE International Conference on Data Engineering*, 1992.

Demurjian, S., M.-Y. Hu, T. Ting and D. Kleinman. Towards an Authorization Mechanism for User-Role Based Security in an Object-Oriented Design Model.

Proceedings of the 1993 Phoenix Conference on Computers and Communications, 1993.

de Paula, E. and M. Nelson. Clustering in Object-Oriented Databases. *OOPS Messenger*, Vol. 3, No. 3, 1992.

Deux, O., et al. The O_2 System. *IEEE Transactions on Knowledge and Data Engineering*, March 1990.

Doherty, M. Implementing Relationships in an Object-Oriented Database. *Master's Thesis, University of Rhode Island*, 1992.

P. Drew, R. King, D. McLeod, M. Rusinkiewicz and A. Silbershatz. Report of the Workshop on Semantic Heterogeneity and Interoperation in Multidatabase Systems. *SIGMOD Record*, Vol. 22, No. 3, 1993.

Eliasen, F. and R. Karlsen. Interoperability and Object ID. *SIGMOD Record*, Vol. 20, No. 4, 1991.

Elmasri, R. and S. Navathe. *Fundamentals of Database Systems.* Benjamin/Cummings Publishing Co., Inc., Redwood City, Ca., 1989.

Fang, D., S. Ghanderharizadeh, D. McLeod and A. Si. The Design Implementation and Evaluation of an Object-based Sharing Mechanism for Federated Database Systems. *Proceedings of the Ninth IEEE International Conference on Data Engineering*, 1993.

Fourtier, P. and J. Prichard. Concepts for a Real-time Structured Database Query Language. *Unpublished, University of Rhode Island*, 1994.

French, J., A. Jones and J. Pfaltz. NSF Workshop on Scientific Database Management. *SIGMOD Record*, Vol. 19, No. 4, 1990.

Gal-Oz, N., E. Gudes and E. Fernandez. A Model of Methods Access Authorization in Object-oriented Databases. *Proceedings of the 19th IEEE Very Large Databases Conference*, 1993.

Garcia-Molina, H. and B. Lindsay. Research Directions for Distributed Databases. *SIGMOD Record*, Vol. 19, No. 4, 1990.

Gordon, K. Requirements for Military Database Management Systems. *Next-Generation Computer Resources (NGCR) Technical Document No. 010 ver. 1.0*, NGCR SPAWAR 331-2, 2451, Crystal Dr., Alexandria, Va. 22245, November 15, 1993.

Graefe, G. and W. McKenna. The Volcano Optimizer Generator: Extensibility and Efficient Search. *Proceedings of the Ninth IEEE International Conference on Transactions on Data Engineering*, 1993.

Gruenwald, L. and S. Liu. A Performance Study of Concurrency Control in a Real-Time Main Memory Database System. *SIGMOD Record*, Vol. 22, No. 4, 1993.

Guenther, O. and A. Buchmann. Research Issues in Spatial Databases. *SIGMOD Record*, Vol. 19, No. 4, 1990.

Hanson, E., T. Harvey and M. Roth. Experiences in DBMS Implementation Using an Object-oriented Persistent Programming Language and a Database Toolkit. *Object-Oriented Programming: Systems, Languages and Applications*, 1991.

Hung, S. and K. Lam. Locking Protocols for Concurrency Control in Real-Time Database Systems. *SIGMOD Record*, Vol. 21, No. 4, 1992.

Ishikawa, H. and K. Kubota. An Active Object-Oriented Database: A Multi-Paradigm Approach to Constraint Management. *Proceedings of the 19th Very Large Databases Conference*, 1993.

Jagadish, H., and X. Qian. Integrity Maintenance in an Object-Oriented Database. *Proceedings of the 18th Very Large Databases Conference*, 1992.

Jajodia, S. and R. Sandhu. Database Security: Current Status and Key Issues. *SIGMOD Record*, Vol. 19, No. 4, 1990.

Jensen, C. et al. A Glossary of Temporal Database Concepts. *SIGMOD Record*, Vol. 21, No. 3, 1992.

Kilov, H. A Review of Object-oriented Papers. *SIGMOD Record*, Vol. 18, No. 4, 1989.

Kim, W., J. F. Garza, N. Ballou and D. Woelk. Architecture of the Orion Next Generation Database System. *IEEE Transactions on Knowledge and Data Engineering*, March 1990.

Kim, W., E. Bertino and J. Garza. Composite Objects Revisited. *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, Volume 18, Number 2, 1989.

Kim, W., N. Ballou, J. Garza and D. Woelk. A Distributed Object-Oriented Database System Supporting Shared and Private Databases. *ACM Transactions on Information Systems*, Vol. 9, No. 1, 1991.

Kim, W. Object-Oriented Database Systems: Promises, Reality, and Future. *Proceedings of the Nineteenth International Conference on Very Large Data Bases*, 1993.

Krishnamurthy, E. and A. Mcguffin. On the Design nd Implementation of Secure Database Transactions. *SIGSAC Review*, Vol. 10, Nos. 2 & 3, 1992.

Lamb, C., G. Landis, J. Orenstein and D. Weintreb. The ObjectStore Database System. *Communications of the ACM*, Vol. 34, No. 10, 1991.

Lohman, G., B. Lindsay, H. Pirahesh and K. B. Schiefer. Extensions to Starburst: Object Types, Functions and Rules. *Communications of the ACM*, Vol. 34, No. 10, 1991.

Lunt, T. and E. Fernandez. Database Security. *SIGMOD Record*, Vol. 19, No. 4, 1990.

Mohan, C., D. Haderle, B. Lindsay, H. Pirahesh and P. Schwarz. ARIES: A Transaction Recovery Method Supporting Fine-granularity Locking and Partial Rollbacks Using Write-ahead Logging. in [87].

Motro, A. Accommodating Imprecision in Database Systems: Issues and Solutions. *SIGMOD Record*, Vol. 19, No. 4, 1990.

Mourad, A., W. Fuchs and D. Saab. Database Recovery Using Redundant Disk Arrays. *Proceedings of the Eighth IEEE International Conference on Data Engineering*, 1992.

Nicol, J., C. Thomas, T. Wilkes and F. Manola. Object Orientation in Heterogeneous Distributed Computing Systems. *IEEE Computer*, June 1993.

Nyanchama, M. and S. Osborn. Role-Based Security, Object-oriented Databases & Separation of Duty. *SIGMOD Record*, Vol. 22, No. 4, 1993.

Olivier, M. and S. Von Solms. A Taxonomy for Secure Object-Oriented Databases. *ACM Transactions on Database Systems*, Vol. 19, No. 1, 1994.

Object Management Group, Inc. Object Services Architecture, Revision 6.0, Document 92.8.4. *Object Management Group, Inc., Framingham Corporate Center, 492 Old Connecticut Path, Framingham, Massachusetts, 01701-4568*, August, 1992.

Open OODB Project. Open OODB Executive Summary Release 0.2 (Alpha). *Texas Instruments, Inc., P.O. Box 655474, M/S 238, Dallas, Texas, 75265,* 1993.

Open OODB Project. Open OODB Technical Overview Release 0.2 (Alpha). *Texas Instruments, Inc., P.O. Box 655474, M/S 238, Dallas, Texas, 75265,* 1993.

Open OODB Project. Open OODB Glossary. *Texas Instruments, Inc., P.O. Box 655474, M/S 238, Dallas, Texas, 75265,* 1993.

Open OODB Project. Open OODB C++ API User Manual Release 0.2 (Alpha). *Texas Instruments, Inc., P.O. Box 655474, M/S 238, Dallas, Texas, 75265,* 1993.

Open OODB Project. Open OODB Query Language User Manual Release 0.2 (Alpha). *Texas Instruments, Inc., P.O. Box 655474, M/S 238, Dallas, Texas, 75265,* 1993.

Open OODB Project. Open OODB Release Notes Release 0.2 (Alpha). *Texas Instruments, Inc., P.O. Box 655474, M/S 238, Dallas, Texas, 75265,* 1993.

Open OODB Project. Open OODB Meta Architecture Document. *Texas Instruments, Inc., P.O. Box 655474, M/S 238, Dallas, Texas, 75265,* 1993.

Open OODB Project. Open OODB System Architecture Document. *Texas Instruments, Inc., P.O. Box 655474, M/S 238, Dallas, Texas, 75265,* 1993.

Open OODB Project. Open OODB Architecture Specifications Document. *Texas Instruments, Inc., P.O. Box 655474, M/S 238, Dallas, Texas, 75265,* 1993.

Open OODB Project. ARPA Open Object-Oriented Database Architecture Specification. *Texas Instruments, Inc., P.O. Box 655474, M/S 238, Dallas, Texas, 75265,* 1993.

Open OODB Project. ARPA Open Object-Oriented Database Module Specification Meta Architecture Support. *Texas Instruments, Inc., P.O. Box 655474, M/S 238, Dallas, Texas, 75265,* 1993.

Open OODB Project. Open OODB Requirements Document. *Texas Instruments, Inc., P.O. Box 655474, M/S 238, Dallas, Texas, 75265,* 1993.

Open OODB Project. The Open Object-Oriented Database: Obtaining Database Functionality by Extension. *Readings in Object-Oriented Systems and Applications.*, IEEE Press, 1993.

Özsu, M., U. Dayal and P. Valduriez. WORKSHOP REPORT International Workshop on Distributed Object Management. *SIGMOD Record*, Vol. 22, No. 1, 1993.

Pangalos, G. Database Systems Security. *University of Rhode Island, Technical Report TR93-225*, 1993,

Peckham, J., V. Fay Wolfe, J. Prichard and L. Cingiser DiPippo. RTSORAC: Design of a Real-Time Object-Oriented Database System. *University of Rhode Island, Department of Computer Science, Technical Report 94-231*, 1994.

Pissinou, N., K. Makki and Y. Yesha. On Temporal Modeling in the Context of Object Databases. *SIGMOD Record*, Vol. 22, No. 3, 1993.

Ramamritham, K. Real-Time Databases. *International Journal of Distributed and Parallel Databases*, 1992.

Rangarajan, S., S. Setia and S. Tripathi. A Fault-Tolerant Algorithm for Replicated Data Management. *Proceedings of the Eighth IEEE International Conference on Data Engineering*, 1992.

Saltor, F., M. Castellanos and M. Garcia-Solaco. On Canonical Models for Federated DBs. *SIGMOD Record*, Vol. 20, No. 4, 1991.

Scheuermann, P., et al. Report on the Workshop on Heterogeneous Database Systems. *SIGMOD Record*, Vol. 19, No. 4, 1990.

Senerchia, J. A POSIX-Compliant Dynamic Real-Time Scheduler. *Master's Thesis, University of Rhode Island*, 1993.

Sheth, A., and J. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Survey*, Vol. 22, No. 3, 1990.

Sreewastav, K. Prototype of the Object Query Language (OQL): an Associative Query Facility for Zeitgeist. *Texas Instruments, Inc.*, 1990.

Stonebraker, M., et al. Third Generation Database System Manifesto. *SIGMOD Record*, Vol. 19, No. 3, 1990.

Stonebraker, M., and G. Kemnitz. The POSTGRES Next-Generation Database Management System. *IEEE Transactions on Knowledge and Data Engineering*, March 1990.

Stonebraker, M., A. Silbershatz and J. Ullman. Database Systems: Achievements and Opportunities. *Communications of the ACM*, Vol. 34, No. 10, 1991.

Stonebraker, M. Managing Persistent Objects in a Multi-Level Store. *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data*, Volume 20, Issue 2, 1991.

Stonebraker, M. and M. Olson. Large Object Support in POSTGRES. *Proceedings of the Ninth IEEE International Conference on Data Engineering*, 1993.

Stonebraker, M., editor. *Readings in Database Systems*, Morgan Kaufmann Publishers, San Mateo, California, 1994.

Stonebraker, M. Operating System Support for Database Management. in [87].

Thuraisingham, B. Secure Interoperability of Trusted Database Management Systems. *SIGSAC Review*, Vol. 10, Nos. 2 & 3, 1992.

Ulusoy, Ő. Current Research on Real-time Databases. *SIGMOD Record*, Vol. 21, No. 4, 1992.

United States Navy. NGCR DISWG Draft Glossary and Acronyms. *NGCR DISWG Requirements Subgroup*, June 17, 1993.

United States Navy. POSIX Delta Document for the Next-Generation Computer Resources Operating System Interface Standard (Version 2). *NUWC-NPT Technical Document 10,076*, June 1, 1992.

Urban, S., A. Karadimce and R. Nannapaneni. The Implementation and Evaluation of Integrity Maintenance Rules in an Object-Oriented Database. *Proceedings of the Eighth IEEE International Conference on Data Engineering*, 1992.

Wells, D., J. Blakeley and C. Thompson. Architecture of an Open Object-Oriented Database Management System. *IEEE Computer*, 1992.

Wilkinson, K., P. Lyngbaik and W. Hasan. The Iris Architecture and Implementation. *IEEE Transactions on Knowledge and Data Engineering*, March 1990.

Wolfe, V., L. Cingiser and J. Peckham. A Model for Real-Time Object-Oriented Databases. *Proceedings of the Tenth IEEE Workshop on Real-Time Operating Systems and Software*, 1993.

Wright, M. Security Services in the OSI Reference Model. *SIGMOD Review*, Vol. 10, No. 1, 1992.

Yu, P., K. Wu, K. Lin and S. Son. On Real-Time Databases: Concurrency Control and Scheduling. *Proceedings of the IEEE*, Vol. 82, No. 1, 1994.