To: PUBLIC DISTRIBUTION
Date: 17–September–1991
Subject: Distribution of OODBTG Final Technical Report, and Request for Comment

The Object-Oriented Database Task Group (OODBTG) was organized by the ASC X3/SPARC Database Systems Study Group in 1989 to gather information on Object Database Management Systems and to recommend standards needed in this area.

We have now completed our Final Technical Report (attached). The report contains

- Recommendations for Standards in Object Information Management
- Reference Model for Object Data Management
- Glossary of Object Data Management Terms
- Report on a Survey of Object Data Management Systems
- Report on Workshops on Object Data Management Standardization
- Bibliography

At our quarterly meeting in Minneapolis, MN, on August 2, 1991, we voted unanimously to release the Final Technical Report.

At this time, we are soliciting public comment in written form by December 1, 1991, for consideration at our January, 1992, meeting in Florida. Your comments may influence the standardization process for object information management systems. Please send your comments to our Correspondence Secretary, Elizabeth Fong, at the address listed below.

Sincerely,
William Kent
Acting Chairman, X3/SPARC/DBSSG/OODBTG
Hewlett-Packard Laboratories
1501 Page Mill Road
Palo Alto, CA 94303-0969

SEND WRITTEN COMMENTS BY DECEMBER 1, 1991, TO:

Elizabeth Fong
National Institute of Standards and Technology
Building 225, Room A266
Gaithersburg, MD 20899
Tel: 301-975-3250
Fax: 301-590-0932
Email: fong@ecf.ncsl.nist.gov

X3/SPARC/DBSSG/OODBTG FINAL REPORT

17–September–1991

Editors:

Elizabeth Fong
National Institute of Science and Technology
Building 225, Room A266
Gaithersburg, MD 20899
Tel: 301-975-3250, Fax: 301-590-0932
Email: fong@ise.ncsl.nist.gov

William Kent
Hewlett Packard Laboratories
1501 Page Mill Road
P.O. Box 10490
Palo Alto, CA 94303-0969
Tel: 415-857-8723, Fax: 415-852-8137
Email: kent@hplabs.hp.com

Ken Moore
Digital Equipment Corporation
Mail Stop NUO1-1/A11
55 Northeastern Blvd.
Nashua, NH 03062
Tel: 603-884-6044, Fax: 603-884-0829
Email: moore@databs.enet.dec.com

Craig Thompson
Texas Instruments Incorporated
PO Box 655474, MS 238
Dallas, TX 75265
Tel: 214-995-0347, Fax: 214-995-0304
Email: thompson@csc.ti.com

CONTENTS

FIGURES

Preface

This is a report produced by the Object-Oriented Databases Task Group (OODBTG) of the Database Systems Study Group (DBSSG). The DBSSG is one of the advisory groups to the Accredited Standards Committee X3 (ASC/X3), Standards Planning and Requirements Committee (SPARC), operating under the procedures of the American National Standards Institute (ANSI).

The OODBTG was established in January 1989. Consistent with usual practice when confronted with a complex subject, DBSSG charged the OODBTG to investigate the subject of Object Database Management systems (ODMs) with the objective of determining which, if any, aspects of such systems are, at present, suitable candidates for the development of standards. The program of work for OODBTG has included a **Survey of ODM Systems**, two public workshops on ODM standardization, an **ODM Reference Model** document, and a **Report on Recommendations for Standards in Object Information Management**. Reports on these tasks, taken together, constitute the **Final Technical Report** of OODBTG.

The scope of the term **object data management** (**ODM**)[1] includes object models and database management systems whereas the term **object information management** (**OIM**) broadens this scope to additionally include the use of objects in programming languages, network management, design methodologies, user interfaces, and related areas. This broadening of scope reflects our finding that common object concepts cross these domains. In this report, we use the term **Object** rather than **Object-Oriented**. For historical reasons, the name of the task group remains **OO**DBTG.

This technical report represents the work of many individuals who attended quarterly OODBTG meetings held in conjunction with DBSSG meetings, and several ad-hoc subgroup meetings held in the east coast, west coast, and midwestern regions. The technical work represents the careful distillation of direct contributions by the members of OODBTG. The opinions and ideas expressed here are not necessarily endorsed by all members nor by the members' sponsoring organizations.

Tim Andrews (Ontologic) served as chairman of OODBTG with William Kent (Hewlett-Packard) serving as vice-chairman throughout 1989 and 1990. William Kent has served as acting chairman with Craig Thompson (Texas Instruments) serving as vice-chairman in 1991. Elizabeth Fong (NIST) has served as correspondence secretary. Editors of OODBTG reports are:

Gordon Everest	University of Minnesota
Elizabeth Fong	National Institute for Standards and Technology
Magdy Hanna	University of St. Thomas
William Kent	Hewlett-Packard
Haim Kilov	Bellcore
Ken Moore	Digital Equipment Corporation
Allen Otis	Servio Corporation
Edward Perez	Texas Instruments
Mark Sastry	Honeywell

[1] ODMs are also known as **Object-Oriented Data Base management systems**, or **OODBs**.

Craig Thompson	Texas Instruments

The following individuals have attended at least one of the OODB Task Group quarterly or regional meetings and/or have made technical contributions to this report:

Tim Andrews	Ontologic
Victoria Ashby	MITRE
Douglas Barry	Itasca Systems Inc.
José Blakeley	Texas Instruments
Roger Burkhart	Deere and Company
Stephanie Cammarata	Rand Corporation
Chris Dabrowski	NIST
Richard T. Due	Thomsen Due and Associates Ltd.
Larry English	Information Impact International Inc.
Gordon Everest	University of Minnesota
Elizabeth Fong	NIST
Jeff Galarneau	Itasca Systems
Roy Gates	Rand Corporation
John Gersting	IUPUI
Ed Greengrass	National Security Agency
Pat Hagey	P.H. Hagey Consulting Ltd.
Magdy Hanna	University of St. Thomas
William Harvey	Robert Morris College
Sandra Heiler	Xerox
Ken Jacobs	ORACLE
Michael Jende	UNISYS
William Kent	Hewlett Packard
Haim Kilov	Bellcore
Salvatore March	University of Minnesota
William McKenney	
Alvin McQuarters	N.E.C. America
Ken Moore	Digital Equipment Corporation
Patrick O'Brien	Digital Equipment Corporation
Allen Otis	Servio Corporation
Bhadra Patel	Hughes Aircraft
Girish Pathak	Xerox
Edward Perez	Texas Instruments
Paul Perkovic	Informix Software Inc.
Satya Prabhakar	Honeywell
Gary Rivord	Sandia National Labs
Katie Rotzell	Object Sciences
Andy Rudmik	Software Productivity Solutions
Shyam Sarkar	UNISYS
Mark Sastry	Honeywell, Inc.
Ron Schachel	UNISYS
Jay Smith	TVW Inc.

Alan Snyder	Hewlett-Packard
Tom Soon	Pacific Bell
Robert Spurgeon	Coopers and Lybrand
Ed Stull	Summa International
Mary Ellen Stull	Summa International
Satish Thatte	Texas Instruments
Craig Thompson	Texas Instruments
Andrew Wade	Objectivity Inc.
David Wells	Texas Instruments
Miya Yuen	Andersen Consulting

SECTION 1
INTRODUCTION

In January, 1989, the Database Systems Study Group (DBSSG), one of the advisory groups to the Accredited Standards Committee X3 (ASC/X3), Standards Planning and Requirements Committee (SPARC), operating under the procedures of the American National Standards Institute (ANSI), established a task group on Object-Oriented Database (OODBTG). This report describes the technical results and recommendations of OODBTG.

1.1 Goal

Interest in and use of object-oriented solutions for information technology problems has exploded in the last several years and has led to diverse definitions and terminology. The lack of a common definition of these terms is causing confusion among the vendors, users and standards developers in the database community. At inception, DBSSG requested OODBTG to investigate the following:

- Establish a working definition for the term "Object Database,"
- Establish the relationship between Object Database technology and "object-oriented" technology in related fields, including programming languages, user interface methodologies, and information modeling methodologies, and
- Establish a framework for future standards activities in the object information management area.

1.2 Purpose and Motivation

The purpose of this final report is to identify those areas of consensus in object information management where standardization activities should be pursued.

At present, there are many, increasingly mature efforts to develop object information technology. There are now several commercial object database management systems and applications based on these systems starting to emerge. Programming languages, repositories, extended relational database systems, communications, user interfaces, and methodologies are developing closely-related object information technology. In addition, there are increasing numbers of groups that are working on standards to cover various aspect of the object paradigm.

There continues, however, to be a need to ensure interoperability among systems. The OODBTG's objective has been to identify potential standards in the object database area, to determine their relationship to existing standards, and to recommend whether and how to proceed to realize new, needed standards for object information management.

1.3 Scope

Since the OODBTG is part of the Database Systems Study Group, the scope of the project has been to focus primarily on object database systems. However, evolving object technology is reshaping some traditional roles and boundaries. Standards for object technology can no longer be developed as isolated components of a software system. The group has realized that the semantics of object technology spans several other areas as well.

In OODBTG documents, the scope of the term *object data management* (ODM)[1] includes object models and database management systems. The term *object information management* (OIM) broadens this scope to additionally include the use of objects in programming languages, network management, repositories, operating system services, user interfaces, design methodologies, and related areas. This broadening of scope reflects our finding that common object concepts cross these domains. In this report, we use the term **Object** rather than **Object-Oriented**. For historical reasons, the name of the task group remains **OO**DBTG.

1.4 Approach

OODBTG began its work in January, 1989, and completed this Final Technical Report in August, 1991. Documents produced by the OODBTG represent the work of many individuals who attended quarterly OODBTG meetings held in conjunction with DBSSG meetings, as well as several ad-hoc subgroup meetings held in the east coast, west coast, and midwestern regions. The opinions and ideas expressed here are not necessarily endorsed by all members nor by the members' sponsoring organizations.

The methodology used to arrive at our recommendations is illustrated in Figure 1–1.

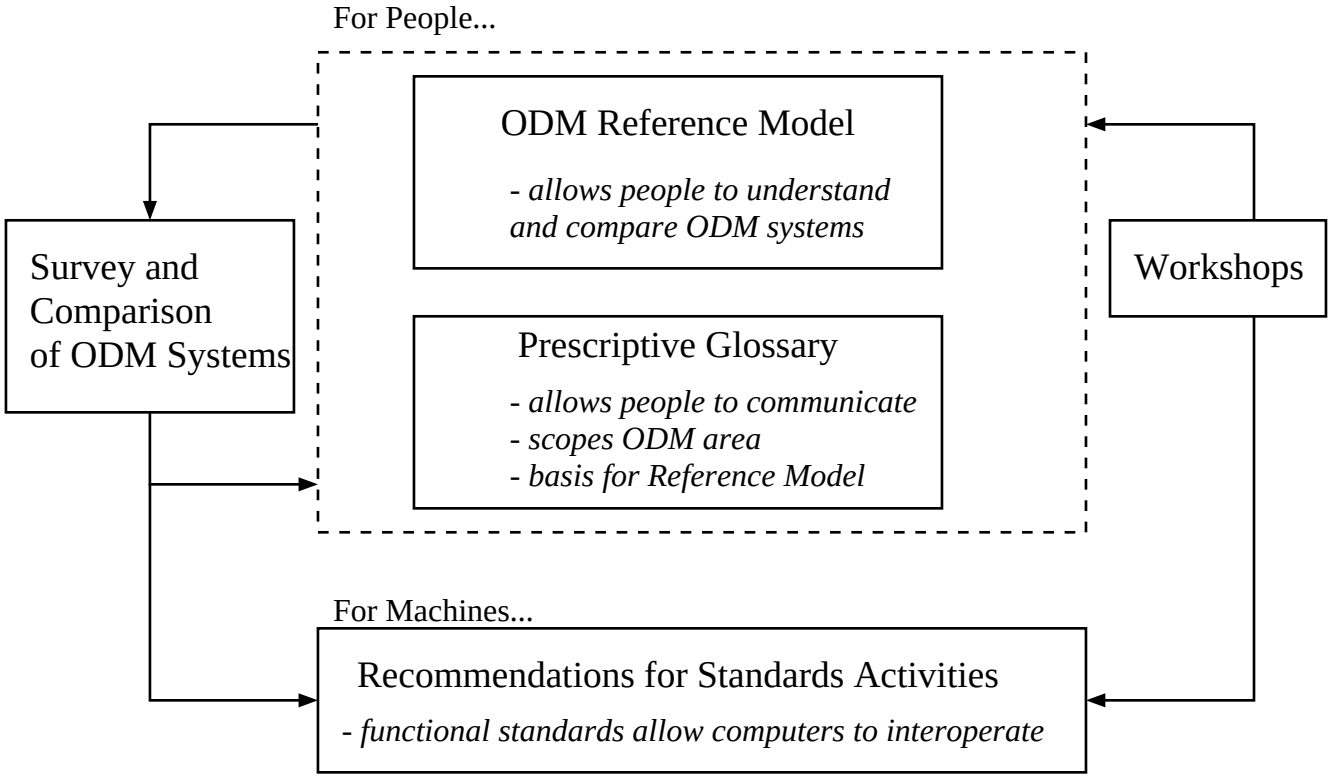
Key deliverables of OODBTG include the following reports:

- *Recommendations for Standards in Object Information Management*
- *Reference Model for Object Data Management*
- *Glossary for Object Data Management*
- *Report on a Survey of Object Data Management Systems*
- *Report on Workshops on Object Data Management Standardization*
- *Bibliography*

The documents *Reference Model for Object Data Management* and *Recommendations for Standards in Object Information Management* represent the careful distillation of direct contributions by the members of OODBTG. Each of these documents has been widely circulated in draft form and reviewed through several revisions at several public OODBTG meetings. The document *Glossary for Object Data Management* directly reflects how terms are used in the reference model and recommendations documents.

[1] ODMs are also known as **Object-Oriented Data Base management systems**, or **OODBs** .

Figure 1–1: How OODBTG Deliverables Interrelate



A survey of object data management systems and two workshops on standardization in the object data management area have provided data on the way practitioners in the field describe their systems.

- A survey form, based on early versions of the reference model document, was developed for the purpose of collecting data about currently available object data management systems. The actual survey was conducted by members of OODBTG affiliated with universities. Eleven very detailed responses were received. Survey results were analyzed for suggested changes to the reference model and recommendations documents.
- Two public workshops were held to assess the degree of consensus for object data management standardization. Workshop papers, discussions, and evaluation forms were used to refine both the reference model and recommendations documents. The workshops were held in conjunction with the International Conference on the Management of Data (SIGMOD) in May, 1990, and the Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA) in October, 1990. Together, they attracted 35 position papers and participation from 10 countries and around 50 organizations and 100 participants.

In both the survey and the workshops, public and vendor comment was explicitly solicited concerning where the object data management community recommends that standards are needed. We have used these inputs to provide a map identifying where concrete standards are desirable and to develop priorities for standards development in the ODM/OIM area. In addition, both forums were used to refine the reference model based on public comment.

1.5 Organization of the Final Report

The final report of the OODBTG is a compilation of a number of self-contained reports, which are identified as sections. The final report is structured as follows:

1. Section 2 contains recommendations for standards in the object information management area. This report is aimed at those who are managing the standards process and will influence the direction of further work in the area of object information management.
2. Section 3 is the reference model for object data management. The purpose of the reference model is to provide a framework within which object data management features can be distinguished from other data management features.
3. Section 4 provides the prescriptive glossary of object data management terms used in the object data management reference model.
4. Section 5 describes the survey of current object database management systems, both commercial and research prototypes, that was conducted. The survey was used to determine how terms are used, what design alternatives exist, how object database management systems vary, and where there may be some consensus on the development of standards.
5. Section 6 contains a summary of workshops on object database system standardization. In addition to the two workshops sponsored by the OODBTG, a third workshop was sponsored by the DBSSG. The purpose of the two OODBTG workshops was to identify areas where consensus of object standards may be possible and desirable, in a setting where authors could expect feedback and possible action on their ideas. The purpose of the third DBSSG workshop was to foster communication among standards groups whose work focuses on object-oriented solutions for information management problems.
6. Section 7 is a bibliography to provide further reading in object information management area.

SECTION 2

RECOMMENDATIONS FOR STANDARDS IN OBJECT INFORMATION MANAGEMENT

2.1 Purpose and Structure of this Section

Functional standards allow computer systems to communicate interoperably. The purpose of this document is to identify potential functional standards in the Object Information Management (OIM) area and recommend how to pursue these standards.

The intended audience for this document consists of:

- Those who are managing the standards process and will influence the direction of further work in the area of object information management systems.
- Those who are building OIM systems or considering using them and need a roadmap of what OIM standards may be available in what timeframe.
- Those who may develop more detailed, progressively refined reference models in the OIM area or in OIM subareas, and those who may develop interoperability standards based on those reference models.

Section 2.2 describes current trends in object information systems standardization. Section 2.3 makes clear the need for functional standards in the OIM area. Section 2.4 documents OODBTG's process for arriving at the list of candidate functional standards. Section 2.5 lists a "family" of potential OIM standards. Section 2.6 lists relevant existing standards bodies. Section 2.7 lists some issues blocking consensus in the OIM area. Section 2.8 concludes with OODBTG's recommendation to X3/SPARC/DBSSG for standardization activity in the OIM area.

2.2 Directions in Object Information Systems Standardization

Today's applications use database operations to manipulate data structures. There is a single interface between user programs and the system code managing the database (see Figure 2–1). The semantics of the operations at this interface are defined in terms of the system-supplied data structures (e.g., records, hierarchies, networks).

OIM introduces a new middle ground (Figure 2–2). Object-oriented applications no longer directly manipulate data structures. In keeping with the object-oriented principle of abstraction, applications apply operations to objects without knowing the implementation of such objects.

Figure 2–1: Traditional Application/Database Interface

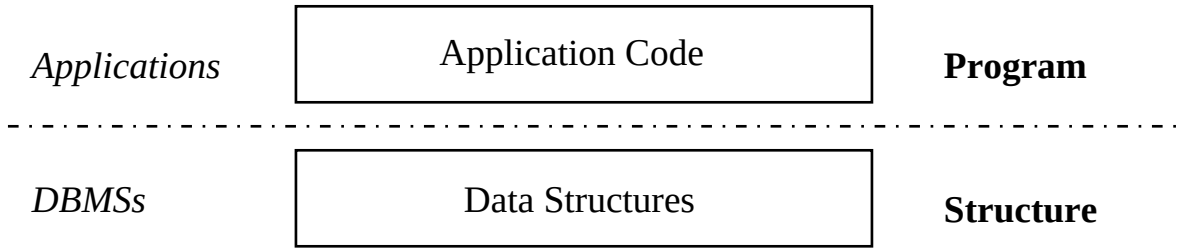
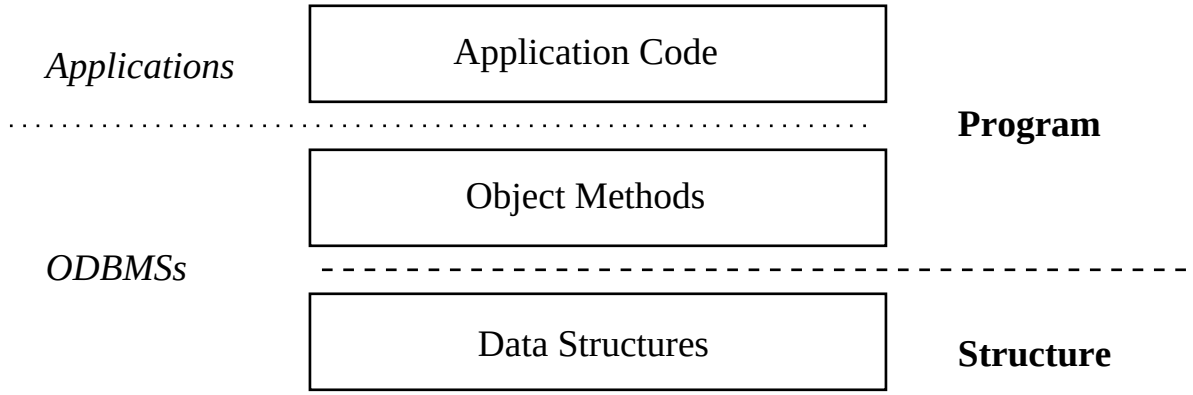


Figure 2–2: OIM Application/Database Interface



The boundary between application and object base no longer coincides with the boundary between user programs and data structure. User programs exist both as applications and as methods in the object base. The object base includes both method programs and data structures. Applications do not manipulate data structures, but invoke operations implemented by methods which manipulate data structures.

The ongoing development of standards should adapt to this evolution of boundaries. Existing standards groups organized to follow the traditional boundaries may have difficulty adapting to OIM boundaries. Gaps or redundancies among standards may result. Strong communication among standards groups (especially, ANSI and ISO, but also OMG, CFI, X3H6 (formerly CIS), and PDES) will be necessary to rearrange the boundaries and to produce effective, useful OIM standards.

Another dimension along which traditional boundaries are shifting involves choice of object model. OIM systems may provide support for object models that match the object model in programming languages or that match future object models that better support interoperation. It is clear that traditional "impedance mismatches" between database systems and programming languages will be reduced or eliminated.

2.3 The Need for Functional Standards

A key deliverable to X3/SPARC/DBSSG are recommendations for potential standards in the OIM area. A common vocabulary (Section 4, Object Data Management Glossary) and a common descriptive reference model (Section 3, Object Data Management Reference Model) are important deliverables since they allow *people* to communicate about OIM systems and compare them. They are necessary but not sufficient deliverables of OODBTG.

In addition, of key importance, functional/operational standards will be needed that will allow *computer systems* to communicate and provide common OIM services. This is needed to ensure interoperability among systems.

As with other areas, standardization of OIM systems will involve tradeoffs: standardization gives up flexibility but gains interoperability. While it may be premature to actually standardize on a specific OIM system interface at present, this report represents a careful assessment of the state of OIM systems vis-a-vis a standardization window and provides a plan for accelerating progress leading to OIM standards.

Standards in the OIM community are of critical importance since lack of standards represents a real risk to industry (design, manufacturing, DoD, etc.). Increasingly complex applications require capabilities beyond those provided by today's established database vendor community (hierarchical, network, relational). Requirements like

- better information modeling tools and methodologies
- more seamless support for programming language data models
- database support for lifecycle management
- queries over complex objects
- a higher level of abstraction and data independence to support application interoperability and code reuse
- better performance

are major needs driving the development of OIM systems.

At present, there are many, increasingly mature efforts to develop OIM capabilities. As described in Section 5, Survey of Object Data Management Systems, these effort range from extended relational systems to persistent language systems, with several variations. Some startup companies and established vendors are already delivering OIM products to their customers who need to adopt OIM technology.

In addition, there are increased numbers of standards groups (see Section 2.6) that are working independently on object models, which are ultimately incompatible.

Unfortunately for the application community, there is no interoperability between today's OIM systems. They do not use common interfaces or protocols or share common formats. While this diversity is valuable and allows the OIM community to experiment, it represents a real danger to the user community since they cannot insure that the products they depend on will survive. That is, there has been no second-sourcing nor any basis for sharing information stored in heterogeneous OIM systems since there has been no agreement that a common object model is possible.

Such an agreement can occur in several ways. A major vendor or alliance could create a de facto standard. Or the standards community can be proactive in working with OIM vendors, relevant consortia (Object Management Group, CAD Framework Initiative, PDES, etc.), and academia to identify and accelerate consensus leading to standards in the OIM community. Effectively, whether consensus comes from an industrial push or a proactive pull, the mandate to reach accord and to be compliant to evolving standards will help drive OIM efforts toward stabilization.

Our effort to identify potential functional standards for OIM systems is described in the following sections. OODBTG is not itself a Technical Committee convened with the objective of defining standards. Instead, OODBTG's objective has been to identify potential standards, their relationship to existing standards, and recommend whether and how to proceed to realize new, needed standards.

Whether or not such standards are initially formed from within the standards community, the basic identification of what are candidate standards still provides a roadmap of the potential for consensus in the OIM area.

2.4 Process for Identifying Potential OIM Standards

Potential OIM standards were identified from three sources:

- Workshops
- Survey of ODM Vendors
- ODM Reference Model

2.4.1 Workshops

Two public workshops were held in 1990. The focus of both workshops was to solicit ideas, positions, and proposals for potential, specific candidate OIM standards. More specifically, the objectives of the workshops were:

- to make the efforts of OODBTG recognized as a force for accelerating consensus towards OIM standardization,
- to gauge public interest in the potential for OIM standardization and to receive feedback on OODBTG technical work, principally, the ODM Reference Model (see Section 3, Object Data Management Reference Model), and
- to solicit from the public specific proposals for OIM standardization.

The workshops were timed to coincide with two key conferences, one concentrating on database technology and the other on object-oriented technology. The first was held May 22, 1990, in Atlantic City, NJ, just before the annual SIGMOD International Conference on the Management of Data (SIGMOD'90), a major annual database conference. The second was held in October 23, 1990, in Ottawa, Canada, in conjunction with the annual Conference on Object-Oriented Programming, Languages, and Applications (OOPSLA'90).

As part of the Workshops, there were discussions and inputs on potential OIM standards. The Proceedings of the two workshops have been published as National Institute of Standards and Technology (NIST) technical reports, NISTIR 4488 and 4503. Workshop reports appear in Section 6, Workshops on Object Data Management Standardization and will appear in a 1991 special issue of *International Journal of Standards and Interfaces*.

A third workshop, on Objects in Data Management, was held to foster communication among standards groups whose work focused on object-oriented solutions for information technology problems. It was sponsored by the Database Systems Study Group of X3/SPARC and held in Anaheim, CA on January 14-15, 1991. Proceedings were published by The MITRE Corporation (MITRE Report No. MP-91W00016).

Recommendations on potential areas for OIM standards were also solicited at the DARPA-sponsored Workshop on Open OODB Systems, held in Dallas, TX, on March 13-15, 1991. A Workshop Report is available from Texas Instruments.

2.4.2 Survey of ODM Systems

The Survey of ODM Systems (Section 5, Survey of Object Data Management Systems) explicitly requested inputs from the vendor community on what can be standardized in the OIM community. In addition, analysis of the Survey suggested other potential standards.

2.4.3 ODM Reference Model

The ODM Reference Model, itself arrived at through a consensus process, is a source of potential standards. The structure of the document reflects the structure of OIM systems, which in turn reflects potential standards. In this view, standards are identified from OODBTG's ODM Reference Model by considering that OIM standards can correspond to each high-level feature of an OIM system ("divide and conquer").

2.5 Potential Technical Areas for Standardization

This section reviews the technical landscape of OIM systems.

Evolving object technology is reshaping some traditional roles and boundaries. OIM is not just another data structure with yet another query and update syntax. OIM encapsulates data structures behind object interfaces. The object model for object users defines interfaces in terms of domain-specific operations, not structures accessed by system-defined operations. Methods play a new role, mediating between the operations defined for object interfaces and the data structures in which they are implemented. The very concept of OIM exists at two levels: the object interfaces defined for users to access information, and the data structures accessed by methods. One of the issues needing clarification is which aspects of information management, and corresponding standardization, are at which level.

Correspondingly, there are two distinct modes in which information might be manipulated. In the first, more traditional mode, data is retrieved from the database to be manipulated by program constructs in the program spaces. In the second, more object-oriented mode, operations are executed in the database space, without exposing data structure to the requester. Standardization of data structure is more important in the first mode than in the second.

In another dimension, OIM is no longer as distinguishable from other aspects of information processing as it used to be. The semantics of object technology cross traditional boundaries, being applicable to user interfaces, programming languages, network management, repositories, operating system services, storage management, and other areas. It should be possible, but not required, to have uniform semantics, syntax, or both for transient data (typically associated with a programming language) and persistent data (typically associated with a DBMS). Mechanisms developed for distributed object management should also be

useful for object data management. The semantics of object requests, as well as the syntaxes and protocols, should be as uniform as possible across all these disciplines.

Thus a *very strong* recommendation is to re-examine traditional boundaries, both in computer technology and in standards organizations (see Section 2.8.1, Recommendation 2). Database may no longer be an isolatable component of a software system. Relevant standards should be developed cooperatively by database committees, programming language committees, repository committees, network management committees, and so on. Object technology standards should be developed in a way that promotes harmony across these boundaries.

When viewed together, the broad range of OIM technologies may seem difficult to categorize. This apparent confusion is due to the multiple dimensions on which the technologies could be viewed. Following are three such views: a conceptually or functionally organized view, a user-oriented layered view, and a view of interoperation levels.

First, the conceptually organized view is shown in Figure 2–3. It provides a "OIM Concept Map" that together compose an OIM system. The various concepts are arranged for expository purposes with related technology areas tied closely together. The lines between the concepts should not necessarily be interpreted as inheritance, composition, or directed relationships. Figure 2–3 is closely related to the OODBTG ODM Reference Model (see Section 3, Object Data Management Reference Model, Figure 3–1, ODM Design Space). The principal conclusion of this section is that OIM standardization will likely be via a "family of standards," not a single monolithic standard, and that several of these conceptual areas may be candidates for standards.

Second, the user-oriented layered view is shown in Figure 2–4. It shows the relative positioning of the many standards related to OIM. From the user's standpoint, interoperation of OIM components based on these standards requires each standard be coordinated with a common model. There are four dimensions of such coordination:

- Within a single category, different standards should be consistent. For example, within programming languages it should be possible for C++ to invoke a method written in FORTRAN or object COBOL. Evolution from one version of a standard to the next should be possible.
- Across a layer (except across Integration Frameworks, which usually have domain-specific semantics), common semantics should apply to similar operations, but redundant operations should not be provided. For example, the repository may need to perform method dispatching, but should utilize the Object Request Broker rather than standardizing a private mechanism.
- An Integration Framework needs to be coordinated with the concepts of a specific application domain. An integration framework may include wrappers around standards, glue between standards, and domain-specific components.
- Object model concepts and terminology should be defined and implemented the same throughout all standards.

Figure 2–3: OIM Concept Map

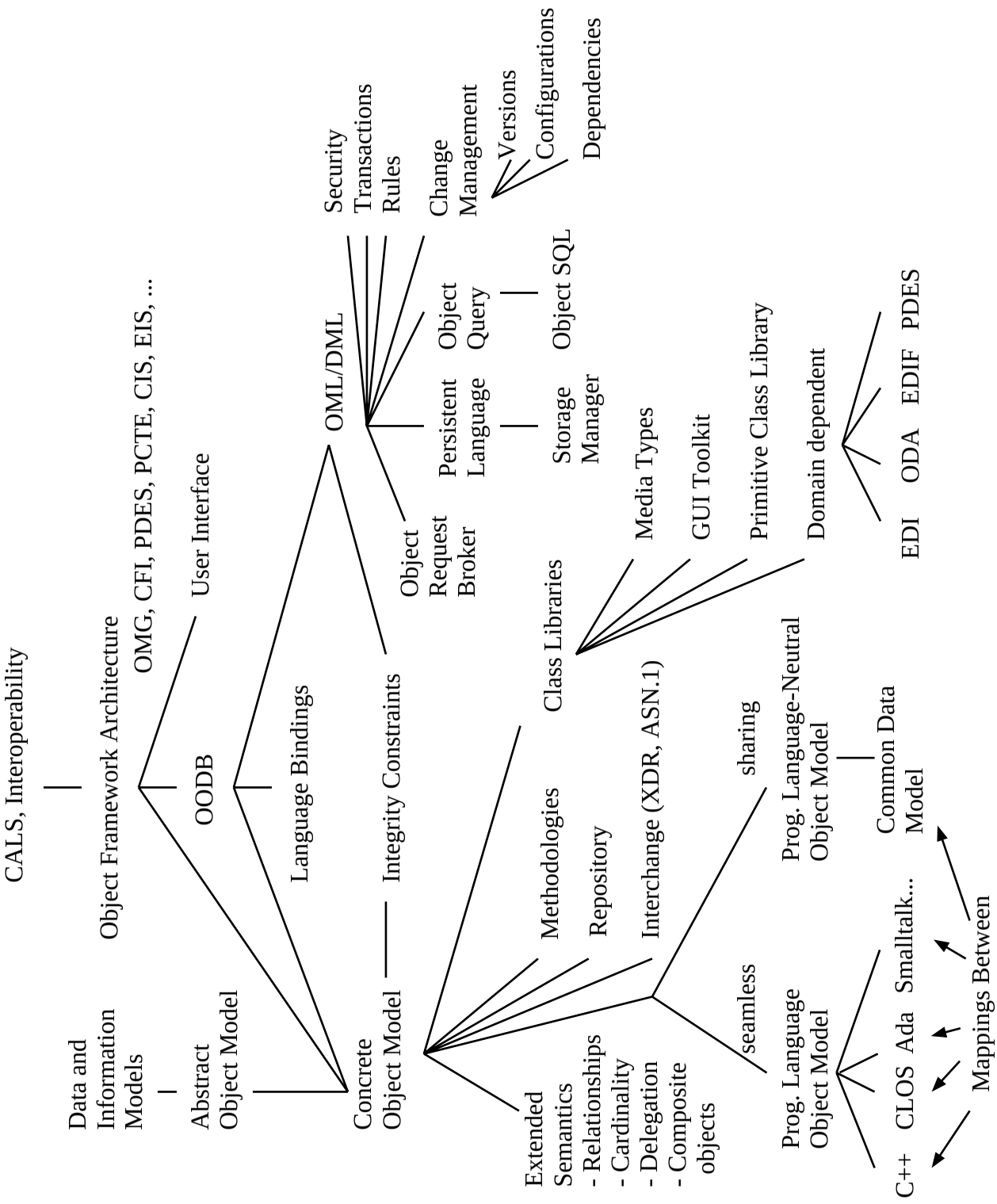
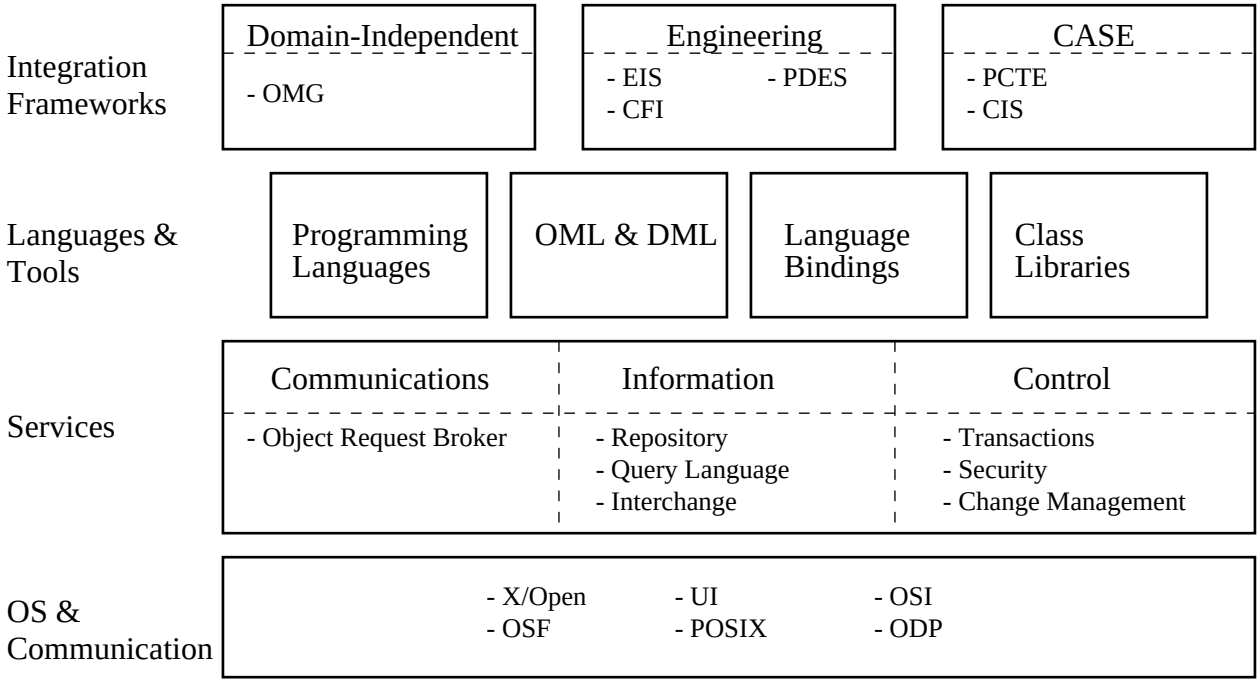


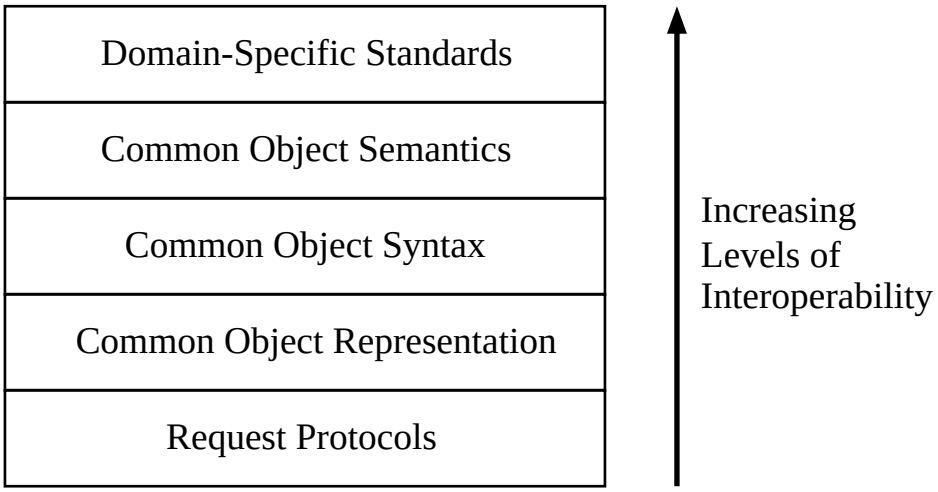
Figure 2–4: Layers of Object Standards



Third, the levels of interoperation provided by a family of standards is shown in Figure 2–5. Within the family of OIM standards, there are varying degrees of commonalty and abstraction. These degrees of commonalty provide five levels of interoperability:

1. At the lowest level, there are standards that define low-level syntax and protocols for making and managing requests.
2. Standards define a common representation for objects. With a single representation, object information interchange is possible.
3. Common object syntax provides a consistent application or user interface. Examples include SQL extensions and persistent C++.
4. Common object semantics are defined in a programming-language-neutral fashion. Standards provide a uniform understanding of identity, inheritance, polymorphism, containment, query, types/classes, etc.
5. At the highest level, domain-specific concepts tailor OIM concepts to particular consumers.

Figure 2–5: Levels of Interoperation



An OIM system should have most of the functions listed in the OODBTG ODM Reference Model (Section 3, Object Data Management Reference Model) and shown graphically in Figure 2–3. Thus, those functions are candidates for interface specifications that can lead to standards.

The remainder of Section 2.5 ignores how and where OIM standards should be pursued and concentrates only on what standards are needed. Based on inputs from the several sources listed in Section 2.4, for each area identified, a measure (HIGH (Section 2.5.1)/MEDIUM (Section 2.5.2)/LOW (Section 2.5.3)) is given for whether consensus can lead to standards in that area over the next three to five years, together with some supporting comments. The following sections describe the potentially standardizable areas.

2.5.1 High Priority Areas

- OIM Components (orthogonal constructs for a type system, persistence, queries, transactions, concurrency, recovery, distribution, and inter-language sharing) — There are good reasons to consider standardizing identified components of OIM systems independently of their use in OIM systems. For instance, OIM systems depend on some subsystems (like transactions and distribution) that may be needed by other subareas (like domain-dependent class libraries and data dictionaries) but there are also separate reasons for these other standards.

  OIM systems should support various kinds of concurrency control: optimistic, pessimistic, long, nested, hybrid, semantic, etc., transactions as well as synchronization. But there is no agreement on exactly which set. Standards for distributed and nested transactions may be driven by standards bodies such as X/Open or by products that achieve de facto standard status.

  Standards for OIM systems should contribute to or adopt object communication and distribution standards that may emerge from groups like OMG (Object Request Broker) or X3T3 Open Distributed Processing.

- Unified Object Model — The General (Abstract) Object Model in the OODBTG and OMG Reference Models provides a description of a very general Object Model. The object model includes the concepts of object identity, class-instance, class-subclass, behavior, encapsulation, delegation, and composite object.

  A unified object model is desirable and may be possible to achieve such that different specific models are subsets of it. It would have to be extensible since there are many (an open-ended number of) kinds of semantic data modeling extensions that such a model might have to account for.

  Without a common object model, standards groups will inevitably continue to develop divergent object models. Further standardization of specific languages and models at this time is counterproductive: freezing the differences will impede convergent evolution. A unified object model would be *very* useful in helping existing standards groups move together toward an interoperable object model.

- Specific Object Models — There are likely to be many specific object models, useful for different purposes. Several programming-language specific object models (e.g., C++, CLOS, ADA 9X, and Smalltalk) and domain-dependent object representations (e.g., PDES, EDI, ODA, and EDIF) already exist. Multiple so-called language-neutral object models (that have automated mappings into programming language object models) also already exist (e.g., SQL3, PDES/Express, IDL, and CASE tools that map to multiple programming languages).

  It is possible and likely that the same OIM system could support multiple specific object models.

- External representations of objects for specific object models — The purpose would be to support both bulk and fine-grained transfer of objects portably across machines, operating systems, and compilers. This support would be useful for distributed systems whether an ODM is present or not. Ada 9X will likely provide this support for Ada. Implementations exist for Common Lisp and are possible for C++. Domain-dependent representations like PDES, EDI, ODA, and EDIF already provide interchange formats.

- Object Repositories (object-oriented data dictionaries) — The scope of data dictionaries includes extensibility for new kinds of metadata; different support for languages like Common Lisp, which supports some data dictionary functionality, and C++, which at present does not; possible support for change management of metadata; and support for mappings between representations. An object-oriented approach seems right but specific standards seem premature. General-purpose OIM systems will need a data dictionary but variations in content, functionality, and interface will probably exist for some time to come.

- Class libraries for primitives (list, set, hash table)— There already exist several such language-specific libraries for Smalltalk, PDES/Express, EDIF, EDI, Eiffel, and IDL. In C++ there are several competing candidates though a de facto standard may emerge.

- Persistence — Persistent programming languages that seamlessly support the object model specific to host programming languages like C++, CLOS, and Smalltalk are a good idea. There is potential for a language-neutral model of persistence. Note that persistence does not imply the need for query support but probably requires the addition of transactions or concurrency control.

- Query Capability — An OIM should support a query language. A query language for an OIM needs to be capable of handling any legal classes (i.e. datatypes) that may be defined in the OIM system, provide a low impedance coupling to the language(s) used to write OIM methods and/or to a persistent programming language, and provide non-procedural constructs that operate on sets and other appropriate collection types.

  A query language bound to a particular language-neutral object model (SQL3 is moving in this direction) and a query language that is bound to different programming languages seamlessly are both desirable.

- Object Design Methodologies — It is desirable to have standards for object methodologies. There is active work in the area. At least for now, there are likely to be several approaches and a design theory for objects is needed. This area is becoming increasingly important, and still more work is needed.

2.5.2 Medium Priority Areas

- Extensibility of the Object Model — It should be possible to include semantics not currently in an object model but that some group wants to add (relationships, delegation, etc.).
- Graphical, media type, hypermedia, and document types (Motif, Interviews, Dexter, ODA) — Some standards already exist, like PHIGS. Other systems exist that are not standards and may be superseded.
- Computationally complete languages for writing methods — Operations in such a language must be able to operate on Object Model objects.

  There is a question whether a language must be new or can be the same as one or more host programming languages. If the former, do we really need YAPL (yet-another-programming-language)? If the latter, how are methods written in multiple host programming languages "bound" or linked to the database?

2.5.3 Low Priority Areas

- Internal architecture of an OIM system — It is likely that OIM systems will not be monolithic since there are independent reasons for the existence of most of their functions and some OIM systems are implemented as object libraries. Similarly, the architecture of an Object Query System might be open to support registration of user-defined indices and programmable to support new query algebras and optimizers.
- OIM Rules Systems — Very little work has been done in this area and there is a rich variety of rule systems. How do rules combine with the ODM Reference Model? Is a new language needed to support them?
- OIM Integrity Constraints — It currently isn't clear which kinds of integrity constraints to support, how to express them, and what their relationship is to a Query or Rules (sub)language, or to a design methodology.
- OIM Namespaces — Eventually some default namespaces may evolve that have hierarchical structure and perhaps some form of OSF/DCE, X.500, or NFS compatibility. In at least some ODM systems, the namespace is programmable.
- OIM Security — In OIM systems, there is a major need for more consensus on security, since little work has been done in this area.

- OIM Change Management — It seems that change management should not be included in the kernel of an OIM system. However, an OIM should somehow support change management of metadata.
- OIM Storage Servers — This is an area that is likely best left to implementers. There is no likely forthcoming consensus on whether to support sets at this level, how to cluster and prefetch, how to buffer, whether to move objects or pages, etc. Consensus may be possible for a minimal interface to allow different backend storage servers to be hooked together (wrapped) in one OIM system.
- User Interface — It seems clear that OIM systems will not drive object-oriented interface standards. X Consortia is looking at a C++ class library based on Stanford Interviews, but from a user interface point of view, OIM systems would just be another application. While built-in object browsers for OIM systems are likely in the near term, more general programmable object browsers that can interface to multiple OIM systems seem more likely to subsume them. Interface standards are not further considered in this document.
- Utilities and Tools — Traditionally, utility and tool components have not been standardized. 4GLs and report writers are examples of such components. Object versions of those components should also not be standardized.

2.6 Current Efforts Relevant to OIM Standardization

Using Figure 2–3, it is possible to identify several existing standards bodies that closely relate to (some of) the OIM family of standards. If OIM standards are viewed as a family or suite of standards, these existing standards bodies are candidates for potentially standardizing "parts of the elephant."

Based on the DBSSG Workshop on "Objects in Information Management Standards," January 14-15, 1991 in Anaheim, CA, several standards groups are moving toward adopting the object paradigm as central to their standard. This list includes several of the groups identified below.

One of the major challenges is that these groups are not coordinated to ensure that OIM standards developed by individual groups will be interoperable and useful when assembled into a whole OIM. Charters and goals may overlap or conflict among formal standards groups, such as ANSI and ISO, and consortia, such as X/Open and OMG.

The working relationship between ANSI and consortia should be further developed and refined. In particular, consortia should have information flow from them into ANSI as well as the other way around. ANSI should examine the way consortia are organized and operate to see what other kinds of committee interactions are required.

The remainder of the section briefly describes how existing standards groups might relate to OIM standards.

Standards Effort	Description
	Database Management
X3H2 - SQL3	A technical committee responsible for the standardization of database languages NDL and SQL. They have, in May 1991, completed specification of SQL2, and are currently working on SQL3, an extension to current SQL standard which will include object concepts.
X3H2.1 - RDA (Remote Data Access)	A task group under X3H2 on Remote Data Access (RDA). This group is responsible for the specification of a protocol concerned with providing access to data stored at remote sites using SQL.
JTC1 SC21/WG3 - Data Management	An international standards committee responsible for the specification of standards on data management. Projects include data management reference model, database languages SQL, IRDS and RDA.
SQL Access Group	A consortium of users and vendors working to advance the RDA protocol and planning to work on a call-level interface to SQL systems.
	Transaction Processing
X3T5 - TP (Transaction Processing)	A task group under X3T5 (OSI) is responsible for the specification of TP which is an application layer protocol used for exchange of information between two or more distributed systems.
JTC1/SC21/WG5	An international standards committee responsible for the specification of standards on transaction processing languages and bindings, including concurrency, commitment, and recovery (CCR).
POSIX 1003.1	A group working on a profile for transaction processing.
X/Open Transaction Processing	A working group developing the XTP model of transaction processing, which includes the XA transaction specification.
	Object Communications and Distribution
X3T5 - OSI (Open Systems Interconnection)	A technical committee responsible for the specification of protocol standards in accordance with the 7-layer Open System Reference Model. In particular, the X3T5.4 Network Management Task Group is responsible for the specification of managed objects using object-oriented technology.
X3T3 - ODP (Open Distributed Processing)	A U.S. technical committee contributing to the international effort JTC1/SC21/WG7. The ODP effort is working on the specification of a standard reference model which will make the complexities of distributed computer systems more transparent. The ODP-RM defines an ODP trader which is a computational object offering services to other objects at service ports.
OMG ORB — Object Management Group's Object Request Broker	A task force within OMG developing technology that performs application invocation and sharing of large granule objects.
JTC1 SC21/WG4 - Management Information Services	An international standards committee responsible for the definition of the information model of managed objects that corresponds to the information aspects of the systems management model. Although the documents refer to CCITT applications, they define general object management concepts.

Standards Effort	Description
	Object Communications and Distribution
X3T1M1.5	A technical committee responsible for common management information services for managed objects defined in accordance with JTC1 SC21/WG4 documents.
OSI/NM Forum	An international forum on OSI network management.
	Data Interchange
X3T2 - Conceptual Schema for Data Interchange	A project under X3T2 working on the standardization of conceptual schema mechanism for data interchange. Responsible for ASN.1, a language for data encoding and interchange.
	Domain-specific Data Representations
PDES/STEP (Product Data Exchange using STEP)	The PDES is the U.S. organizational activity that supports the development and implementation of STEP. STEP is the standard for the exchange of product model data. The level 3 product data sharing implementation specifies that multiple user applications access data to a common shared database.
EDI (Electronic Data Interchange)	EDI is an application layer protocol. It is a standard which describes formats for orders, payments, shipments, billings, and other business transactions.
EDIF (Electronic Data Interchange Format)	A format for exchanging CAD chip design data.
ODA (Office Document Architecture)	ODA is a standard for interchange of documents (including text, facsimile and graphics information) which are produced in an office environment. Interchange of ODA documents may be by means of data communications or exchange of storage media.
	Repositories
X3H4 - IRDS (Information Resource Dictionary Systems)	A technical committee responsible for the specification of IRDS1 family of standards. This IRDS1 family of standards includes a command language and panel interface specification, a soon to be approved Export/Import File Format standard, and a Service Interface specification. The next family of IRDS standards will utilize object technology.
X3H6 - CIS (CASE Integration Services)	A technical committee working on a family of standard interfaces between CASE environment framework components and tools. Standards are being developed for service and tool registration, change management (versions and configurations), and an object model.
EIA CDIF (Electronic Industry Association CASE Data Interchange Format)	An industry association established to permit interchange of CASE models and data among tools.
	Programming Languages
X3J4 - COBOL (OO COBOL)	A technical committee responsible for the standardization of the COBOL programming language is working on extensions to COBOL that will include object concepts.

Standards Effort	Description
	Programming Languages
X3J9 - Pascal	A technical committee working on the standardization of the Pascal programming language, which is working on a Technical Report for object-oriented extensions to Pascal.
X3J13 - Common LISP	A technical committee working on the standardization of Common LISP which includes the Common LISP Object System (CLOS).
X3J16 - C++	A technical committee responsible for the standardization of C++ programming language.
X11/SC1/TG11 - MUMPS	A task group working on object-oriented extensions to MUMPS
Smalltalk, Objective-C, Eiffel, ...	Object programming languages.
Ada Joint Program Office	The group coordinating the development of Ada 9X.
	Frameworks and Consortia
CFI - CAD Framework Initiative	CFI is a consortium chartered to define interface standards that facilitate integration of design automation tools for the benefit of end users and vendors worldwide.
OMG - Object Management Group	The OMG is a consortium to promote object-oriented concepts and methods. The OMG architecture defines an interface called Object Request Broker (ORB). The Object Model Task Force is developing a description of a concrete object model.
PCTE - Portable Common Tools Environment	PCTE is an emerging ECMA standard for specifying interfaces which are primarily designed to facilitate communications and interoperability among cooperating CASE tools and applications.
OSF - Open Systems Foundations	A consortium which, as part of its project, is defining a distributed management environment (DME) which utilizes object concepts.
ESPRIT	Esprit is a European funding agency working on the specification of information system architectures, including CIM-OSA, COMANDOS, CSA, AND DELTA-4.
EIS - Engineering Information Systems	An US Air Force sponsored framework effort.
X/Open	A consortium of users, hardware and software vendors, developing portability guides for languages, databases, and operating systems.
DARPA	The US DoD funding agency funding work on: knowledge representation Standards Initiative, Open OODB, and NIST Persistent Object Testbed.

2.7 Issues That Need Resolution

Existing OIM systems vary in several ways. These variations are either the source of a family of standards or they are roadblocks to standards. Following is a list of some important variations. Since these are sources of variations in OIM systems, they can be viewed as "issues needing resolution" when standards are considered.

- Choice of Object Model
  Several OIM systems are based on a persistent version of C++, Smalltalk, or some other existing language. The idea is to support strong typing "seamlessly" at the application-to-database boundary and to use the language itself as the object manipulation language. Object models in programming languages have advantages: they are widely used and accepted. A disadvantage is that a mapping is needed to cross language boundaries (though many applications do not need to do this -and also- such

mappings are needed anyway, independent of persistence or databases, to pass data in interlanguage subroutine calls).

Other OIM systems support a programming language-neutral object model. In the latter case, there may be an automated mapping from the OIM object model to language-specific object models. Or the mapping may be left to the application programmer. From any other language's point of view, an OIM object model based on any one other language can be viewed as a language-neutral object model.

A likely resolution is to pursue standards for persistent programming language bindings and also one or more language-neutral object models.

- Object Model Extensions
  Relationships, composite objects, integrity constraints, etc. What family of object model extensions will a standard OIM permit?
- OIM Reliability
  OIM systems built on persistent versions of existing programming languages may or may not be able to provide certain kinds of protection to user data (for example, integrity or security constraints).
- Object Model support for Sets and Queries
  Existing OIM languages and systems exhibit wide variation in the semantics of sets and queries in their object models. A class definition may or may not include a set of all instances of a class. Sets may be supported by the object language, by a class library, or not at all. In contrast, in the relational model, sets of tuples are the only first class constructor.

  An OIM system may or may not support a query facility. If queries are supported, the query portion of an OIM Object Manipulation Language may be language neutral, such as some form of Object SQL, or may be a language-specific facility built into the Object Manipulation Language. Sets and queries may or may not be orthogonal to persistence, that is, transient sets and queries may be supported.
- The Line dividing OIM and Applications
  There seems to be some consensus that Change Management (Versions and Configurations) of objects should *not* be built in to the OIM system but be added on top, possibly via class libraries. Change Management (support for design lifecycles) seems to be needed for design applications (with variations) but not so much for some other kinds of applications.

  This issue can be viewed as an instance of the problem of setting some dividing line between OIM systems and applications that use them. An issue of whether an OIM system should itself provide a rule system is similar.
- Form of Application Program Interface
  Should the standard interface to an OIM be functions, message passing, or language extensions? mixed? or something else?
- OIM Systems versus Frameworks
  Application Integration Framework architectures (OMG, EIS, CFI, PCTE, ...) overlap in functionality with OIM systems. Both depend on some form of data or model, support libraries and programming language interfaces, some form of distribution (object communications), and some form of data dictionary. Also, frameworks typically assume some kind of database support though how much varies widely (queries, change

management, persistence, ...?). At issue for standards groups is the dividing line between OIM systems and Frameworks.

2.8 Recommendations for Standards Activities

This section contains our final technical recommendations to X3/SPARC/DBSSG regarding standards activity in the OIM area. Specific organizational recommendations are not made here.

The recommendations are divided between areas for standardization and areas to defer or avoid standardization at this time.

2.8.1 Areas for Standardization

RECOMMENDATION 1

Standards should be developed for the following subareas of OIM systems: Object Model, Object Communications, Persistence, Transactions, Query Capability, and Object Repository. Proposed work items in developing standards in these areas are to define requirements; develop coordinated reference models for each area; develop programming language-neutral (functional) models; then to define interface specifications.

OODBTG has laid the groundwork for useful continued work that will be needed to assess, guide, and accelerate standards in the OIM area. Some areas have on-going standardization activities - some do not. The object model brings new issues and requirements to existing areas that should be taken into consideration by these standardization efforts. Standardization of these subareas, in the form of a family of standards, is desirable and likely in the three to five year time frame.

- Object Model
  OODBTG, OMG, PDES, X3J16, X3H2, and many other groups are working on object models. Continued work is needed on harmonizing different object models, on interoperability of object models, and on object model semantics and syntax. There is a driving need for sharing objects across language, operating system, and machine boundaries. Current systems are based on similar but different object models. Interoperation among systems can be achieved by:

  1. defining a common, unified object model
  2. defining semantics-preserving mappings between specific object models
  3. defining a minimal object model

  These approaches should all be pursued.
- Object Communications
  Clearly needed by OIM systems, this subsystem is at the heart of Framework architectures as well. X3T3 (Open Distributed Processing) already exists to provide standards in this area. The OMG Object Request Broker Task Force is working on a specification in this area.
- Persistence
  A base document, Language Persistence Reference Model, could be the first step, with sections on various languages (C++, Common Lisp, Ada, SQL, MUMPS, etc.). A language-neutral model of persistence may be possible.

- Transactions
  Due to new application domains in which OIM systems are used, new transaction models need to be defined. These additional models should be compatible to existing or in-progress transaction standards (e.g., X/Open, X3T5). As with Persistence, a base document for a Transactions Reference Model would be a useful first step.
- Query Capability
  The advantage of early standards for persistent languages and for object query languages, such as SQL3, is that these areas are needed by many next-generation database applications.
- Object Repository
  To provide for shared definitions to be used among OIM components, a standard object repository needs to be defined. X3H4 is currently working on such an object repository.
- Class Libraries
  Wherever multiple class libraries exist for the same domain, agreement needs to be reached on the common contents and organization of sharable and interoperable class libraries. This applies within and across languages.
- Object Design Methodologies
  There exists a need to add methodological considerations to current perspectives on using object technology. Different object methodologies may exist and will appear in the areas of system planning, analysis, and design.

The benefits for having a family of OIM standards are:

- Define basic concepts - avoid re-invention
  The lack of an accepted definition for fundamental OIM concepts has forced standards groups and vendors to start from scratch. Providing a family of standards that cover the basic concepts will raise the level of common agreement and allow future work to address more advanced OIM concepts.
- Eliminate redundant effort
  Multiple standards groups working in the same area generate similar but different definitions for the same concepts. Unfortunately, there are currently many examples of overlap in the OIM area. Also, areas not addressed by standards have to be developed by each vendor individually.
- Provide for convergence of existing systems
  The Survey of ODM Systems found that there are many OIM products and it is likely that there soon will be more. A family of OIM standards will provide the means to converge those existing systems with themselves and other non-OIM systems. Users of OIM products could then be able to use compatible components from multiple vendors.

RECOMMENDATION 2

An Interoperable Object Model (IOM) group should be formed.

Almost every standards group listed in Section 2.6 is considering some form of object-oriented extensions. In Section 2.5, we listed a family of potential OIM standards. The conclusion is that many of these proposed object standards are relevant to many of these existing standards groups.

Uncoordinated standards groups may not arrive at a family of standards from which a workable OIM can be constructed. Object-related standards should be developed in a coordinated fashion by database committees, programming language committees, repository committees, network management committees, and so on.

Specifically with respect to object models, which cross the boundaries of many groups, we recommend the formation of a new group to undertake the work item on Object Models in Recommendation 1.

Both OODBTG's ODM Reference Model (Section 3, Object Data Management Reference Model on General Object Model), Object Management Group's Abstract Object Model, and PDES/STEP Express object model may be considered as base documents for such a group. The proposed group will need *strong* liaisons to several X3 and other standards groups. These include most of the standards efforts listed in Section 2.6.

2.8.2 Areas to Defer or Avoid Standardization

Some specific areas in Figure 2–3 do not seem like good candidates for standards and may not need to be pursued as aggressively at this time. These include areas listed as having Medium or Low priority in Section 2.5.

For example, while it will eventually be useful to reach consensus on rule-based systems and some form of openness may be needed to allow rule-based systems from different OIM systems to be compatible, it does not seem that standards for OIM Rules Management are needed as quickly as some other OIM standards. This is because there is likely to be a variety of interesting rule-based systems required for different application domains. This is not to discourage consensus-building efforts in this area, just to recognize a lower priority for this area for the near term.

Since there are several areas with Medium or Low priority, more work will be needed before these areas can be standardized. There is particular need for research in OIM integrity and security.

Violations of the ODM Reference Model should not be subject to standardization. For example, implementation-specific behavior of methods or the notion of attributes associated with stored data, which violates the concept of encapsulation, should be avoided.

SECTION 3

OBJECT DATA MANAGEMENT REFERENCE MODEL

3.1 Introduction

This section summarizes the motivation behind the development of **Object Data Management** (**ODM**) technology. It then summarizes the purpose and intended audience of this document, and describes related OODBTG standardization activities. It concludes with an overview of Object Data Management systems, and a description of the organization of this document.

3.1.1 Motivation

Industry trends are driving the computer community toward developing Object Database Management systems. One trend is the emergence of new classes of applications in areas such as office automation, document processing, and various engineering and manufacturing disciplines. New kinds of information management systems, like hypermedia systems and application integration frameworks, need support for representing and modeling the behavior of large, complex, multimedia data types. These applications are dominated by a need for rich data modeling support and often depend as much on the ability to navigate through heterogeneous types of data as to process queries on sets of homogeneous types of data.

Another trend is the need for information systems to solve larger, more complex problems with requirements for planning, simulation, development, maintenance, enhancement, portability, information integrity, extensibility, and interoperability. Design applications in software, electrical or mechanical CAD, or documents need information management systems that have mechanisms for change management and for managing group work to better manage the life-cycle of large designs developed by many designers. The opportunity is to transfer some of the burden of this extra complexity from applications to underlying systems.

In programming, the use of objects has emerged as an important approach to making better data modeling support available to programs while making programs less aware of and less dependent on details of how data structures and operations are implemented.

As object-oriented applications have grown in size, there is an increased need to share and permanently store such objects so that they can be made available to other programs than the ones which created them. Thus there is a need for *persistent objects*. One approach to persistence is to store objects in files. Very often, however, the need arises for additional functionality, such as recovery, concurrency control, access control, and query – i.e., for database support.

There are now several Object Data Management System (ODM) products and substantial research prototypes. Several authors have stated requirements for ODMs and have offered definitions and initial specifications for consideration. ODMs have reached the point where it makes sense to consider their potential for formal standardization.

This document is a "reference model" which characterizes Object Data Management Systems. With the expanding use of objects in both programming and database work, there has emerged a set of general object principles that apply across a wide range of systems using objects. These principles are grouped together in Section 3.2 of this document on the "generalized object model." Readers whose technical background is primarily databases, as opposed to programming languages, may find that this document contains more discussion of object-oriented concepts, which are commonly associated with programming languages, than they might have expected. Object-oriented concepts are independent of particular programming languages. These concepts are central to any meaningful discussion of object data management. Though, historically, many of these concepts first appeared in object programming languages, this does not limit their applicability to object data management. Section 3.3 characterizes object data management, and what it adds to both the object and database paradigms. Section 3.4 characterizes aspects of Object Data Management systems.

Some Object Data Management systems exhibit an integration of programming and database features to the extent that an application may be written in a single language. Thus, the ODM Reference Model and future ODM standards need to accommodate this *persistent programming language* form of ODM system. The concepts presented in this reference model are intended to be applicable to both systems with a traditional application/database partitioning, and systems using the single persistent language approach. For instance, both kinds of systems can be augmented with query, change management, or other data management support.

3.1.2 Purpose of Document

This document defines a **reference model** for ODMs. The purposes of this reference model are:[1]

- To provide prescriptive definitions for ODM terms, and thus a common language for people to use in communicating about ODMs.
- To provide a framework within which ODM systems can be distinguished from other data management systems.
- To define common characteristics for ODM systems, based on object and database management concepts, allowing people to understand the key ideas and scope of ODM systems and providing a basis for comparing ODM systems.
- To enable OODBTG to make recommendations regarding future ASC/X3 standards activities in the areas of data management and languages, such as SQL, IRDS, ODP, C++, and related standards. [2]

[1] Good examples of successful reference models, not necessarily related to ODMs, are the OSI 7-layer reference model for data communications and the ANSI/SPARC 3-schema architecture.

[2] See Section 2.

The ODM Reference Model is neither a tutorial on object concepts, nor an exhaustive reference. Both tutorial and detailed reference material are beyond the scope of this document. The ODM Reference Model is not entirely prescriptive; it is occasionally descriptive, using phrases like "*some* systems ... *may or may not* ... (have some characteristic)." Furthermore, the ODM Reference Model does not explicitly provide ODM requirements, nor does it define an ODM architecture, nor does it provide concrete interfaces, nor is it a vehicle for conformance testing. Though important, these objectives are also left for future or other work.

Instead, the document is an abstract description that will support concrete specifications of multiple ODM systems. Future work will be need to progressively refine the ODM Reference Model to provide concrete standards that will allow systems and machines to interoperate.

3.1.3 Intended Audience

This document accommodates both the programming language and database perspectives of ODM systems and can be read by individuals in both communities.

The intended audience for this reference model consists of:

- Those seeking to understand what the object paradigm adds to the traditional database paradigm and to data management systems.
- Those seeking to understand requirements of next-generation database systems, since the ODM Reference Model *can* be viewed as an implicit Requirements Document in that it characterizes a next-generation database system meant to satisfy a wide variety of next-generation database requirements.
- Those who are building or using ODM products and needing a common way to compare or characterize ODM systems.
- Those who are managing the standards process and will influence the direction of further work in the area of object databases.
- Those who may develop more detailed, progressively refined reference models in the ODM area or in ODM subareas, and those who may develop interoperability standards based on those reference models.

3.1.4 Relationship to Standards Activities

It is necessary to put this Reference Model document into the context of the overall OODBTG charter and statement of work. The Final Technical Report of the OODBTG will be submitted to its parent organization (ASC X3/SPARC/DBSSG) in July 1991. It will include:

- **A Survey of ODM Systems**. The data collected by OODBTG in this survey is being analyzed to provide refinements to the Reference Model, including dimensions of comparison for how ODM systems differ. The survey also provides inputs to the **Recommendations for Standards in Object Information Management**.
- **Workshop Reports** covering two public workshops on the potential for ODM Standardization. OODBTG organized these workshops in May, 1990, and October, 1990. The workshops have provided inputs to the ODM Reference Model and to the **Recommendations for Standards in Object Information Management**.

The workshop proceedings are available as NIST technical reports[3] and will be reprinted in the International Journal on Computer Standards and Interfaces in mid-1991.

- **The ODM Reference Model**. This Reference Model document is an important instrument in gaining consensus for characterizing what an ODM is. It provides a prescriptive glossary of ODM terms that builds a common language for talking about ODMs. Further, it provides a set of characteristics that an ODM should have, which provides a framework for comparing different ODM systems.
- **Recommendations for Standards in Object Information Management**. Based on other OODBTG work, this report provides recommendations on what standards would be reasonable and useful in the ODM area.

  This report presents a map of areas in the ODM area that may be a basis for ODM standardization. It argues that ODMs will best be standardized by a "family of standards" approach. It relates potential ODM standards to existing standards groups and describes road blocks to standardization. It concludes with recommendations for next steps in ODM standardization.

The various documents in the **Final Technical Report** have been influenced by liaisons with standards groups and industrial consortia including Object Management Group (OMG), CAD Framework Initiative (CFI), PDES/STEP, X3H2 SQL, X3H4 Information Resource Dictionary Systems (IRDS), X3T3 Open Distributed Processing (ODP), X3J13 Common Lisp, X3J16 C++, and others, including the DBSSG-cosponsored Third Joint Standards Workshop focused on "Objects in Information Management" held at Anaheim, CA, January 14-15, 1991, and the DARPA-sponsored Workshop on Open OODB Architectures, held in Dallas, TX, March 13-15, 1991.

3.1.5 Object Data Management Overview

This section presents an overview of the ODM reference model. As described above, a "reference model" provides a framework that defines a system, process, or other artifact, providing criteria and features that cover important aspects of existing and future systems, thus permitting them to be compared. In addition, a reference model should be useful in providing a road map for identifying and developing concrete standards. The framework of the ODM Reference Model is presented as a design space of characteristics (features, capabilities, functions).

A design space is a methodological tool that provides an organized way to record the space of design characteristics and design decisions. It takes the form of an AND-OR graph. AND layers show how a module is composed into functions, parts, or characteristics. OR layers represent alternative ways to realize a function. AND layers show similarities shared by all decompositions. OR layers provide dimensions for comparison. The upper layers of a design space are more abstract. As the space is refined, it covers more system-specific and concrete design details. Because it encodes not only similarities but also variations, a design space can describe a **family** of systems.

Figure 3–1 shows the design space for ODM systems. Only the top AND-layer is shown in the diagram. Sections 3.2, 3.3, and 3.4 describe these characteristics in detail. As can be seen in the diagram, the characteristics are grouped by heritage, since ODM systems inherit object, database, and system characteristics.

[3] Contact Elizabeth Fong, fong@ecf.ncsl.nist.gov, 301-975-3250.

Figure 3–1: ODM Design Space

```
Object Data Management
    General Characteristics of Object Models
     |---Objects: Operations, Requests, Messages,
     |    Methods, and State
     |---Binding and Polymorphism
     |---Encapsulation
     |---Identity
     |---Types and Classes
     |---Inheritance and Delegation
     |---Noteworthy Objects:  Relationships, Attributes,
     |    Literals, Containment, and Aggregates
     |---Extensibility
     |---Integrity Constraints
     |---Object Language

    Data Management Characteristics
     |---Persistence
     |---Concurrency and Transactions
     |---Distribution
     |---ODM Object Languages and Queries
     |---Data Dictionary and Namespace
     |---Change Management: Versions, Configurations,
     |    Dependencies, Schema Evolution
     |---Reliability and Recovery
     |---Security

    ODM System Characteristics
     |---Class Libraries
     |---Program and User Interfaces
     |---User Roles
```

Based on consensus of the OODBTG, the characteristics shown in Figure 3–1 are the commonly shared characteristics of ODM systems. **The design space we describe is maximal in the sense that not all ODM systems have all features in this AND layer.** For instance, some ODM systems may not have user interfaces. In addition, some of the characteristics (e.g. transactions, recovery, distribution, security) appear to be largely data model independent; that is, they do not differ from similar notions for relational and other database systems, except as noted in the related sections. Finally, OR layers are not shown in Figure 3–1 though they are described in the respective sections. They provide differentia, or dimensions of comparison, where different ODM systems may take different approaches in implementing a characteristic. Some ODM systems may implement more than one OR-alternative (e.g. both pessimistic and optimistic transactions). AND-layers have implications for standards since they provide areas where individual standards may occur if there is consensus or where systems should be left open for implementors when there are many OR-layer alternatives. Since only the top levels of the design space is described in this document, it is most accurate to consider that the ODM Reference Model to date is an "abstract reference model" for ODM systems. That is, at this level of detail, it provides a way to compare ODM systems but does not provide an immediately implementable specification.

3.1.6 Organization of this Document

Following the Introduction section, this document consists of three major sections.

- Section 3.2 describes general characteristics of object models of ODM systems that are identifiable in any object-oriented system, independent of whether the system has database characteristics.

- Section 3.3 describes data management characteristics of an ODM System that are additions to the general characteristics of object models.
- Section 3.4 describes the ODM system characteristics, discussing other ODM system characteristics including class libraries, user roles, and ODM interfaces.

3.2 General Characteristics of Object Models

This section is a self-contained description of object models and is not specific to Object Data Management systems.[4] We present the following general characteristics of object models:

> Objects, Operations, Requests, Messages, Methods, and State, Binding, Polymorphism, Encapsulation, Identity, Types, Classes, Inheritance, Delegation, Relationships, Attributes, Literals, Containment, Aggregates, Extensibility, and Integrity

These terms are defined independently of any distinction between the programming and database domains. These terms are found in a broad range of systems using the object paradigm, including programming languages, user interface systems, distributed systems, repository systems, application integration frameworks, and database systems.

3.2.1 Objects: Operations, Requests, Messages, Methods, and State

3.2.1.1 Operations, Requests, and Messages

Various object models embody various concepts of "object". All models consider an **object** to be an identifiable thing which plays a role with respect to a **request** for an **operation**. The request invokes the operation which defines some service to be performed. From a user's viewpoint, an object may be a **recipient** of the request, a parameter in the request, or something returned as a result of the request.

There are two broad categories of object models, which we refer to as "generalized" and "classical object models." Classical models are, in many ways, subsets of generalized object models but they are founded on different metaphors.

Generalized object models do not distinguish a recipient from other request parameters. In generalized models, a request is defined as an event which identifies an operation and optional parameters. For example, an *AddToInventory(part1, lot2, bin3)* request does not give special significance to either the part, the lot, or the bin. These objects participate uniformly in the request.

Classical or **messaging object models** do distinguish a recipient. In classical models, a request is defined as an event which identifies an operation, a recipient, and optional parameters. Either the part, the lot, or the bin could be designed to be the recipient of the *AddToInventory* request. The request to a specific recipient is called a **message** . A common syntax places the recipient first: *part1.AddToInventory(lot2, bin3).*

[4] Since the Object Management Group's Abstract Object Model document and the ISO Open Distributed Processing group's object model document were developed over the same period of time as OODBTG's "General Characteristics of Object Models", a careful attempt to combine these reference models has been left to future work. There is a pressing need to develop one interoperable Object Model Reference Model shared among many standards groups. See Section 3.2.11.

The **protocol**, [5] or **interface**, of an object describes the operations in which the object can participate. In classical models, the interface of an object includes the operations for which it can be a recipient. In generalized models, the interface includes operations for which the object can play the role of any parameter. The protocol completely specifies the behavior of an object but does not provide visibility to the implementation of the object's operations. Only operations specified in the protocol are allowed.

Objects may be static (**passive**) recipients of requests or dynamic (**active**) agents capable of such activities as sending requests.

3.2.1.2 Methods

A **method** provides the implementation of an operation for certain objects. Different objects may have different methods for the same operation.

In the classical object model, a method is considered to implement the interfaces of operation recipients, but not the interfaces of other parameters. In generalized models, a method is considered to implement the interfaces of all parameters.

3.2.1.3 State

Some operations alter the behavior of future requests. For example, if the *ContainsPart* operation returns *True* or *False* depending on whether a certain bin contains a certain part, then the result of *ContainsPart(bin1,part2)* may be changed by the request *StoreInBin(bin1,part2).*

The general concept of **state** concerns the information that must be remembered when a request alters the future behavior of other requests. Some models equate state with structural **attributes**, i.e., state consists of the current values of attributes. In other models state may be defined as the results returned by certain operations, or state may not be precisely defined.

3.2.2 Binding and Polymorphism

Binding chooses a method or methods to be executed in response to a request, based on the operation and objects in the request.

Polymorphism means that the binding of a request involves a choice among alternative implementations of the operation. Polymorphism allows **overloading** of operations, which means that an operation may be implemented in several alternative methods, associated with different classes of parameters.

In classical models, the binding choice is based on the class of the recipient. In generalized models, the binding choice may depend on the classes of several parameters.

Different object languages (see Section 3.2.10) support different criteria and times for binding an operation to a method for execution. Examples are **early binding** at compile-time and **late binding** at run-time.

[5] Here, the term "protocol" is used as a synonym for "interface." This usage is different from the usage in communications.

3.2.3 Encapsulation

A key concept of the object paradigm is **encapsulation**, which is defined as a kind of abstraction that enforces a clean separation between the external interface of an object and its internal implementation.

Users of objects are people or programs which send requests to objects. Users of objects observe the behavior of objects in terms of the operations which may be applied to objects, and in terms of the result objects returned by such operations. An operation may be implemented (i.e., supported or realized) by a variety of different program code and data structures. Encapsulation means that these implementations are not visible to the user of the object, hiding details of whether and what data structures or code are used in an implementation. The importance of encapsulation is that it insulates applications from object implementations allowing them to be modified without requiring applications that use them to be modified.[6]

Some classical object models define a stronger form of encapsulation, in which the states of objects are isolated from each other; the implementation of objects is disjoint between objects; all methods and states belong exclusively to individual objects.

An assumption that attributes or other structural constructs are necessarily implemented as stored data would violate encapsulation. An attribute such as the age of a person, the weight of an assembled object, the circumference of a circle, or the font of a word within a document might be stored in some implementations and computed in others. Violations of encapsulation, if allowed at all, should be made explicit (e.g. *public* data in C++).

3.2.4 Identity

Each object has a unique, immutable **identity** which provides a means to denote or refer to the object independent of its state or behavior. The definition of identity includes the concepts of identity compare operation, logical identifiers, object identifiers, and object creation.

The identity concept can be described in terms of a primitive **identity compare operation** "==" such that, if *X* and *Y* are the same object, *X == Y*. Some consequences of identity compare are:

- *h(X)* and *h(Y)* have the same result for any operation *h()*, where *h()* has no **side effects** (i.e., *h()* is not a state-changing method), and
- the mathematical set *{X, Y}* has cardinality 1.

An abstract representation of object identity is by means of a **logical identifier** (**LID**), which is an implementation-free token that is uniquely associated with a specific object. The term **object identifier** (**OID**) refers to a specific implementation of a logical identifier. OIDs are normally system-generated, with a fixed format, and are not changeable by the user. In models which consider literals or extensional sets to be objects, such objects have LIDs but possibly not OIDs.

[6] The *three schema architecture* of conventional database systems, in which an application schema (view) maps to a shared conceptual schema which, in turn, maps to a physical schema, can be seen as providing the same kind of **implementation independence** (at two levels) as encapsulation.

An LID is meaningful within some limited scope which we shall call an **object space**. Object creation occurs within an object space. Each creation event results in a LID that identifies the created object uniquely within that space. The creation event creates a representation of the object in the space.

An LID is unique within a particular object space. If *X* and *Y* are objects within an object space, *X* = = *Y* if *X* and *Y* have the same LID. In many implementations, simple comparison of OID and literal representations is used as the basis for "= =". An LID from one object space is not a valid LID in another object space. Identity comparison between LIDs only makes sense when they are defined in the same object space. Object identifiers are handled differently in different systems when objects are exported, imported, replicated, versioned, or distributed.

3.2.5 Types and Classes

The object paradigm deals with both abstract, external behavior, and implementation. Objects may be grouped into types by commonality of behavior (interface), and into classes by commonality of implementations.[7]

A **type** defines a protocol shared by a group of objects, called **instances** of the type. The type itself may be an object.

A **class** defines an implementation (methods and data structures) shared by a group of objects. A class may be considered an implementation of a type. There may be several classes (implementations) per type. The class itself may be an object; methods may also be objects.

The **signature** of an operation identifies the number and types of the operation's arguments and results. In classical object models, an operation is associated with the type of its recipient, and a method is associated with the class of its recipient. In generalized models, an operation is associated with the types of its parameters, and a method is associated with the classes of its parameters. Thus, in generalized models, operations and methods may be "jointly owned" by multiple types and classes. In general, objects can have multiple types and classes, and these can change. Not all models permit this.

A **parameterized type** is the result of a compile-time operation that takes classes as parameters and returns one or more derived classes, e.g., *set<employee>* might return "class set-employee," the definition of a set class corresponding to a set of employee class instances.

3.2.6 Inheritance and Delegation

Inheritance and delegation both involve deriving new definitions from existing ones.

In **class-class inheritance**, classes are arranged as nodes in a graph, with unidirectional links between nodes defining inheritance paths. **Subclasses** (**derived classes**) are **specializations** of their more general parent classes. **Superclasses** (**base classes**) represent **generalization** of their child classes.

[7] Within this section, the terms "type" and "class" are used prescriptively. While it is important to define precisely the terms "type" and "class," almost everywhere in this document except this section, the terms are used interchangeably. The term "class" is used uniformly in the rest of the document because it is used in X3J13 Common Lisp (CLOS), X3J16 C++, and Smalltalk. Some programming languages like C++ define "type" to be a restricted form of class without encapsulation, inheritance, and behavior and "class" to include these object concepts.

Conflicting definitions of protocol, behavior, and state may be inherited along paths of the inheritance graph and are resolved in system-dependent ways. Different **conflict resolution** and **method combination** strategies are used in different systems to define the semantics of inheritance. When creating a subclass, attributes and operations may be added to those inherited from the superclasses, or they may selectively **override** (replace), or **block** (hide) those from the superclass.

The two most common forms of class-class inheritance are **single inheritance**, where the graph is a tree, and **multiple inheritance**, where the graph is a directed acyclic graph.

In **class-instance inheritance**, instances of a class inherit a common interface and an initial state (default or initial values of certain attributes) from the class.

In **delegation**, which can be characterized as **instance-instance inheritance**, an object assigns or transfers behavioral definitions or implementations or state to another object. Some delegation-based systems do not differentiate between class and instances. Object-to-object delegation generalizes both class-class inheritance and class-instance inheritance.

3.2.7 Noteworthy Objects: Relationships and Attributes, Literal Objects, Containment Constructs, and Aggregate Objects

Different object models may distinguish or provide special support for the following noteworthy objects. The list is not exhaustive nor are the objects with the characteristics identified here necessarily mutually exclusive. Some may not be treated as objects in all object models.

3.2.7.1 Relationships and Attributes

A **relationship** (**association**) is a logical relation between instances of one or more classes of objects. Relationships can be binary or n-ary. Binary relationships may be one-to-one, one-many, or many-many.

In generalized object models, a relationship may be realized as a separate class, as an operation along with its result, as a separate modeling construct, or not at all. In different object systems, relationships can be established, traversed, or removed, and may be system or user defined. In different systems, the implementation of relationships may be unidirectional or bidirectional.

In some models, object interfaces only contain explicitly defined operations. In other models, interfaces include operations which are implicitly defined in terms of logical **attributes** (**properties**, **instance variables**, **data members**) or other structural constructs (independent of implementation). Typically, defining an attribute *A* implicitly defines the inclusion of operations such as *Get A* and/or *Set A* in the interface of an object.

Attribute values are defined by a class which constrains possible values. Attribute values may be single- or multi-valued. Attributes are sometimes used to represent binary relationships.

3.2.7.2 Literals

Literals are objects such as numbers and character strings. Literal values are similar to created objects in that both may occur as operands of operations. Literal values are different from created objects in that they have immutable state and they do not have a create operation because their representations are explicitly recognized. For example, multiple occurrences of the same literal number are all references to the same point on the number line. Operations which return literal values are constructing references to objects, not creating objects. In most object systems, if *X* and *Y* are literals, *X* = = *Y* only if *X* and *Y* represent the same literal.

3.2.7.3 Containment Constructs (Composite or Complex Objects)

There are various sorts of **containment constructs** in object models. They can be characterized by how containment is established and what it signifies. Models which support an attribute concept sometimes consider all the values of an object's attributes to be contained in the object. Some models assume that objects have a natural intrinsic content, in the sense that a document consists of some text, or a file consists of a byte string. In other models, literal values are considered to be contained in the object, while non-literals are not. In some models attribute values may be either objects or pointers to objects; objects which are pointed to are not considered part of the object. There may be an explicit *contains* relationship among objects, so that an object contains those objects which are so related to it. Containment may be treated as a property of a relationship, allowing various relationships to be designated as implying containment.[8]

Abstractly, the semantics of containment can be described in terms of **propagation** of operations, resembling a form of inheritance. A request to print or move a document is expected to print or move the text of the document, along with any diagrams or other sub-objects contained in it. Thus, for certain operations *f*, if *x contains y* then *f(x)* is expected to cause *f(y)*. Details vary. There may be different decompositions of a given object into constituents: a book may be composed of chapters and paragraphs, and also of pages and lines. Different operations may propagate in different ways along different containment paths. Destroying an object might or might not destroy a contained object. This might or might not depend on whether the latter is also contained in some other object. The authors and font of a book may or may not be the authors and font of its chapters.

Object systems often support value-based **equality comparison operations** which allow objects to be *equal* even if they are not identical. Equality is defined differently in different systems, may be system- or user-defined, and may be class-dependent. Equality comparison operations may depend on the primitive identity comparison operation "= =", on value comparison operations, on containment comparison operations, or in terms of equivalent behavior.

[8] The notion of containment captures the implementation notion of *reference*.

3.2.7.4 Aggregates

Aggregate objects such as sets, bags, lists, tuples, arrays, etc.,[9] are containment constructs that group other objects organized in some manner. Aggregates typically support operations to access individual members, and to iterate over all members, as in queries. Aggregates may be **homogeneous**, containing only objects from the same class or from classes inheriting from the same class, or they may be **heterogeneous**, containing objects from any class.

With respect to identity, there are two kinds of aggregate objects. **Intensional** aggregates have an identity based upon their creation event. **Extensional** aggregates have an identity based upon their membership. If *X* and *Y* are the LIDs of extensional sets, then *X* == *Y* is equivalent to *X* and *Y* having the same members. For an intensional set, the membership at any time corresponds to an extensional set. Two intensional sets remain distinct objects if they have the same members. If the membership of an intensional set changes, the identity of the set does not change.

3.2.7.5 Other

In various object models, the following may not be treated as objects

- operations.
- classes/types.
- some of the arguments of an operation.
- the result of an operation.
- relationships and attributes.
- literals.
- aggregates.

3.2.8 Extensibility

Extensibility allows additions and modifications to existing classes. There are several kinds of extensibility:

- New classes may be defined, commonly based on existing classes.
- Existing classes may be modified to add new operations, attributes, constraints, or implementations. This extends the protocol of the class.
- Existing instances may acquire or lose a type (e.g. as when a *student* becomes an *employee*).

The term **schema evolution** is used for operations that modify existing class definitions or the inheritance graph; the term **instance evolution** is used for the process of making existing instances consistent with modified class definitions.

[9] These are logical constructs distinct from implementation-specific data structures.

3.2.9 Integrity

The encapsulation characteristic makes it possible to define consistency rules as part of the object definitions that are enforced by the system. These consistency rules are known as **integrity constraints**. Integrity constraints can be defined in several ways including:

- informally, in natural language or in an informal specification language. Such integrity constraints cannot be enforced.
- as procedures, in an object manipulation language. These may include **triggers** and **before- and after-methods**.
- as **assertions**, in a predicate-based language. Important kinds of assertional integrity constraints include: **preconditions** and **postconditions**, and **invariants**.

Preconditions and **postconditions** are Boolean expressions associated with an operation which specify the *pre*condition which should be TRUE immediately before execution of the operation and the *post*condition which should be TRUE as a result of its execution, i.e., immediately after execution of the operation. If the precondition or postcondition is not true, the operation results in an error. A precondition and a postcondition together constitute a **contract** that can be used to specify an operation.

Invariants are operation-independent conditions associated with a collection of classes that must be true at all times **not** within an operation on those classes. The concept of class-level invariants can be used to provide a means of defining concepts like "relationships" and "containment."

Examples of some important kinds of integrity constraints include:

- Only pre-specified operations should be allowed on objects; i.e., the protocol of objects should be enforced.
- Constraints may be expressed on the allowed types of operands and results of operations and on the allowed types of values for properties of objects.
- Uniqueness constraints may be expressed on values of a property, across the domain of instances of a class, or across members of some collection.
- Referential constraints, including rules governing identity and existence, are enforced. These include rules governing creation of object identifiers and rules that insure that referenced items exist.

Constraints may apply to all instances of a class, or only to specific instances. Depending on the database system, constraints may be automatically enforced at run-time, may be enforced at compile time, and/or may be performed only when a particular request is sent or when some system event occurs (e.g., transaction commit).

3.2.10 Object Language

Closely related to an object model is a language, referred to as an **Object Language** (**OL**), that is used to specify and manipulate class definitions and object instances.

An OL has the following uses and capabilities:

- **Class Definition and Manipulation** — An OL may be used to manipulate (i.e., create, access, modify and destroy) classes, including operation signatures, inheritance, and constraints.

- **Operation/Method Definition** — An OL may be used to specify the definition and implementation of operations on objects. Within the implementation of operations, OL commands may be used to manipulate the private data structures of objects. Encapsulation permits, and hides, that operations may be implemented in different programming languages.
- **Object Manipulation** — An OL may be used to manipulate (i.e., create, access, modify and destroy) instances of classes.

In some cases, an OL is **computationally complete**: it can be used to specify general programming operations. In other cases, it may be restricted, e.g. to a query language. An OL may or may not be an existing programming language (which may also support non-object-oriented programming).

3.2.11 Concrete Object Models

The Generalized Object Model presented in this section is described in terms of characteristics. It is missing a particular object language syntax and semantics and is not itself implementable. As such, it can conceptually account for similarities and differences among the family of realized, **concrete object models** that do provide these more detailed specifications. Operational object model standards will need to have concrete specifications.

A Generalized Object Model may help to reduce the growing number of concrete object models that are felicitously different. However, there appears to be no one universal object model but instead a need for various more or less rich object models: several different concrete object models already exist in languages such as Smalltalk, C++, Objective C, Common Lisp, and Eiffel; in knowledge representation systems; in data interchange languages like Express used by the PDES/STEP community[10]; and in specification languages like IDL .

There continues to be a need for an **interoperable object model** that allows object sharing across programming language boundaries. This sort of object model is sometimes called a "language neutral object model" though it typically has a data definition and manipulation language of some kind.

3.3 Data Management Characteristics

This section discusses those characteristics of an ODM system which are specific to database or data management. Some of these characteristics have first appeared in the data management field in association with the object paradigm. Others have always been part of data management systems but are modified by the object paradigm. A few are relatively unchanged. The characteristics discussed in the following subsections are orthogonal to each other.[11] This implies that ODM systems can exist that exhibit some but not all of these characteristics. Also, increasingly, the lines are being redrawn between operating systems, databases, programming languages, and applications, and a consequence is that some of the services described below may be offered by any of these systems, not exclusively ODM systems.

[10] Product Data Exchange using STEP (STandard for the Exchange of Product model data).

[11] The Data Management Characteristics are also orthogonal to choice of concrete object model and several can be applied to non-object systems, e.g. Persistent Algol.

This section does not provide an exhaustive treatment of each characteristic. Rather the goal is to provide a brief description of each characteristic, with emphasis on the aspects which are unique or changed because of the object paradigm.[12]

3.3.1 Persistence and Object Lifetimes

In most programming languages, objects are transient by default. That is they disappear when the process that created them terminates unless their state has been explicitly saved. In databases, data is usually persistent by default. Once created, database objects typically persist until they are explicitly destroyed.

In the ODM paradigm, new rules about persistence are needed to manage transient and persistent objects. The rules governing persistence and transience of objects are system dependent.

The **life** of an object is the period during which it exists. An object's life begins as the result of a **create** operation. An object's life ends as the result of a **destroy** operation. An object whose life begins and ends within a single process is called a **transient object**. An object which lives longer than the execution of the process that created it is called a **persistent object**. Some primitive objects such as literals, have eternal lives, are neither created nor destroyed, and represent themselves.

An object is **reachable** if and only if there is an operation that returns the object as a result. The **transitive closure** of an object is all objects contained directly or indirectly within the object, or more generally all objects reachable from the object directly or indirectly by repeated application of some filtering predicate. The term **navigation** is often used when moving from object to object, possibly through a sequence of move operations.

An object is perceived to exist if and only if it is reachable. If no operation exists that will return the object as a result, then the object is **unreachable** (though it may or may not be physically destroyed and system-internal operations may or may not still keep track of the object.) Destruction of an object makes it unreachable. Operations which render an object unreachable have the effect of destroying the object.

Rules for creating and destroying objects, both transient and persistent, are system dependent. In some ODM systems, the application explicitly invokes creation and destruction operations; in others, some sort of implicit scheme based on **referential integrity** insures that objects that other objects depend on cannot be explicitly destroyed as long as the other objects continue to exist. See Section 3.2.7 and Section 3.2.9.

At the implementation level, destruction or deletion of objects may involve reclaiming the storage space occupied by such objects. Some systems allow a destroyed object's object identifier to be reclaimed for later reuse; others do not.[13]

Possible rules for making an object persistent include:

- The system only operates on persistent objects.
- Persistence is a property of an object's type.

[12] Some terms, like the term **view**, are not defined in the object model. The term "view" could be defined as a stored query (see Section 3.3.4), as an application schema in the three schema database architecture (see Footnote 6), as a security notion, or, generalized, as a synonym for encapsulation.

[13] Schemes like **garbage collection** or **reference count** may be used to manage and reclaim space associated with destroyed objects.

- An object is designated persistent at object creation.
- An already created transient object is explicitly made persistent by means of some "make-persistent" operation, for instance, an explicit "save" operation.
- An object which is reachable from some persistent object is automatically persistent.
- Combinations of the above.

3.3.2 Concurrency Control and Transactions

Existing concurrency control, synchronization, and transaction concepts continue to apply in ODM systems. Concurrency control and transaction management can be assured for only those processing elements within the control of the ODM. **Concurrency control** consists of mechanisms provided to implement and control the sharing of objects in the presence of multiple processes. Such mechanisms include several varieties of locks, including **read, write**, and **notify locks**, and also **timestamps**. **Concurrency control protocols** are rules or conventions for controlling concurrent access to objects. **Synchronization protocols** are concurrency control protocols that each process must follow to concurrently access shared objects.

Some concurrency control protocols like **checkin** and **checkout**, in which objects are copied (checked out) of a database, manipulated, usually for a "long" time, then copied back (checked in) to a database, are simple to implement and useful for operating on independent objects, but do not provide a guarantee of operating on a consistent database state. Other concurrency control protocols do provide such a guarantee.

A **transaction** is a concurrency control protocol in which a sequence of operation requests are grouped into an **atomic** or indivisible unit of work. The transaction begins in a consistent database state and either **commits** or **aborts** all state changes, leaving the database in a consistent state. The transaction has exclusive or controlled access to shared objects during the duration of its execution. While orthogonal to the object paradigm, a rich variety of transactions is typically supported in ODM systems according to the differing kinds of applications ODMs support. Some important characteristics of transactions include (list is not exhaustive nor necessarily mutually exclusive):

- **Pessimistic or Optimistic** — Pessimistic transactions are the traditional kind of transaction, with two-phase locking, which waits when there is a conflict on shared data until the conflict is resolved. Optimistic transactions access objects without locking, assuming that conflict is unlikely, and check for conflict at the end of the transaction, sometimes checking timestamps.
- **Distributed** — A distributed transaction can access data from multiple sites.
- **Short or Long** — Short transactions last seconds and can be aborted; long transactions last hours to months and may need conversational or interactive protocols to merge branched database state.
- **Cooperative or Multi-Threaded** — Cooperative or multi-threaded transactions can have multiple threads of execution.
- **Semantic Transactions** — In semantic transactions, conflicts on shared objects are determined by the commutability of operations and states of the shared objects.
- **Nested** — A nested transaction is completely contained inside some other transaction.

- **Mixed-Mode and Hybrid** — In mixed-mode transactions, child transactions of a parent transaction (or a session) are of different kinds. For instance, optimistic and pessimistic concurrency control can coexist. In such an environment the concurrency control can be chosen on a transaction by transaction basis. This allows transactions that rarely conflict with other transactions to use optimistic concurrency control, while transactions expecting frequent conflicts can be pessimistic. In a hybrid transaction, attributes of various of the above kinds of transactions are combined (e.g., to form "long, optimistic, cooperative transactions," "distributed, pessimistic transactions," etc.).

3.3.3 Distribution

The **distribution** characteristic of an ODM is orthogonal to the object paradigm and also to other database characteristics.[14] A distributed system is one in which information or components of the system exist in one or more address space or computer system.

In an ODM system, things which may be distributed include objects, operations, classes (i.e., the schema), ODM system functions, ODM system processes, and user processes. The granularity of partitioning distributed things, the support for concurrency control and synchronization, and the degree of replication and reliability are system dependent.

Distribution of ODM objects may involve any of the following: making objects the unit of partitioning, such that an object or replica of an object exists on one computer system; spreading the implementation of a single object across multiple computer systems; or grouping objects together into units of transfer like pages or object transitive closures.

The extent to which ODM operations are affected by distribution is described by the system-dependent **transparency** attribute of distribution. For example, in non-transparent distribution, a query operation must include location information as one of its operand objects. In transparent distribution, a query operation does not include location information as an operand because the implementation of the query operation is capable of resolving location automatically.

Some distributed ODM systems will be implemented with each node having full ODM functionality; others will be implemented via client-server architectures with clients implementing some ODM functionality, such as methods, and servers providing other ODM functionality, such as storage.

3.3.4 ODM Object Languages and Queries

An Object Language (OL) of an ODM is analogous to the data definition language (DDL), data manipulation language (DML), and query language facilities of other database paradigms.[15]

An ODM OL may be special- or general-purpose. It is system-specific

- whether an OL is computationally complete and procedural, or is declarative,
- whether an ODM supports more than one OL,
- whether different languages are used for schema and method definition,

[14] Both X3T3 Open Distributed Processing's "trader" concept and Object Management System's Object Request Broker provide an object communications and distribution service that is database independent.

[15] The boundary between database systems and programming languages is blurring or shifting. The ODM OL may be distinct from or the same as the Object Model OL (Section 3.2.10) or the Application Program Interface language (Section 3.4.2).

- whether specialized languages may be available to different kinds of users, e.g., to a database administrator to tune or reconfigure the ODM,
- whether an ODM OL is a separate language or the same as some existing (object-oriented) programming language, and
- what kinds of integrity constraints the ODM supports and enforces.

A **query** operation may be one of the object manipulation operations provided by the ODM Object Language.[16] Queries are operations on sets or collections of instances that have a predicate-based, declarative specification, and that may result in sets or collections of objects. Queries may be specified in object derivatives of SQL, or in direct manipulation query languages. Queries may or may not be written in the form of methods, operations, or language constructs.

ODM query operations may be extended with respect to traditional query languages in such ways as:

- Querying and/or returning complex data structures.
- Operating on user defined classes of objects.
- Operating on other kinds of collections as well as sets.
- Allowing procedurally-specified operations in the query predicate.
- Making use of encapsulation, inheritance, or operation overloading.
- Opening the query optimizer to allow new kinds of indices or new query algebras.

3.3.5 Data Dictionary and Namespace

Meta data includes information about object class definitions, methods, inheritance, constraints, implementations, indices, location, and authorization. A **data dictionary** or **repository** is used to keep track of meta data. Some ODM systems treat metadata and data uniformly using the same language constructs to create, navigate, manipulate, query, and version both metadata and data.

In many ODM systems, a **namespace** is used to allow applications to associate some form of ODM-wide name with LIDs of ODM objects. Not all objects are named by namespace names. Namespace names may or may not be hierarchically structured, like file names in directory systems.

3.3.6 Change Management: Versions, Configurations, Dependencies, and Schema Evolution

The need for supporting **change management** in ODM systems reflects the influence of application domains such as CAD and document management where there is a need to record and manage the entire life-cycle of complex objects like software, chip designs, and documents. These domains generally require recording and controlling three aspects of change:

- **versions** — the history of an object
- **configurations** — the composition or grouping of objects into a higher level object

[16] For compatibility with the relational model, we stop short of generalizing the term “query” to be synonymous with “an operation that returns a value” and restrictively define it with respect to sets and collections.

- **dependencies** or **transformations** — how to propagate consistent effects of change to related objects.

Change management policies determine much of the behavior of the change management component of an ODM. These policies control:

- when new versions are created; how versions other than the current version are referenced; whether no versions, linear versions, or branching (non-linear) versions are supported, including how to "merge" branched versions; what the protocol is for referencing a versioned object; and whether each version has a separate identity;
- how deep to propagate the versioning operation into the components or parents of a newly versioned object; whether references to a versioned object refer to a specific version, or to some version in context; and
- whether dependencies will be propagated immediately or lazily.

ODM systems may provide operations, types, classes, and/or methods to manage versions, configurations, or dependencies or they may leave these operations to other system builders or to application developers.

Change management support applied to meta data is termed **meta data evolution**. **Schema evolution**, described in Section 3.2.8, is a special case of meta data evolution. ODMs offer varying levels of support for changes to meta data.

3.3.7 Security

An ODM system may provide support for **security**. Security is orthogonal to the object paradigm. Security has at least four essential aspects:

- The subjects, users, or agents of the system must be named or identified.
- The subjects must be **authenticated**, such as through a login procedure at the start of a session or transaction.
- **Authorization operations** are provided to specify which operations a subject is permitted to perform, and to specify which objects may be operated upon or used as operands of the permitted operations.
- An **audit operation** is needed to record such events as success or failure of authentication attempts and authorization operations granted or denied.

Both discretionary and mandatory **access control** schemes can be defined for ODMs. In **discretionary access control**, "grant" and "revoke" operations enable/disable agents to perform operations on objects. Thus, one could have different authorizations for different operations on the same object. In **mandatory access control**, users are constrained by a fixed security policy, e.g., as when objects are assigned to security categories like "secret" and access rules prevent users in lower categories from executing operations permitted only for users in higher categories.

In an ODM system, the security facilities should be integrated with the concepts of identity, class, encapsulation, inheritance, and integrity.

3.3.8 Reliability: Recovery, Fault Tolerance, and Error Handling

ODM systems must be **reliable**. It is possible that a process may fail at a time when the state of the ODM system is being changed. **Recovery** is the process of reproducing a consistent state of a database after failure of an application process (whether embedded or not), the ODM system, the operating system, or the underlying hardware. All processing elements of an application may or may not be within the control of the ODM, thus recovery from an error may not be fully possible.

ODM systems may offer other mechanisms to improve their reliability. **Fault tolerant** ODM systems offer replicated, semantically identical services in lieu of a failed service. **Error management** facilities can be defined that apply exceptions to the normal control flow possibly in combination with recovery facilities.

3.4 ODM System Characteristics

This section discusses ODM system characteristics, including class libraries, information modeling, user roles, application program interfaces, and user interfaces. The discussion concentrates on aspects that are unique to ODM systems.

3.4.1 Class Libraries

In object-oriented systems, classes are the central information modeling construct.[17] Classes are grouped into a **class library** when they are related to or depend on one another and together implement some family of abstractions. General principles for creating class libraries include:

- Class libraries should be specifiable at different levels of detail, from abstract to implementational.
- Class reuse can exist at many levels. Abstract class specifications may be reused even if their implementations cannot.
- The specification for the components of a class library should be precise.
- The specification for the components of a class library should be implementation-independent.
- Application-specific class libraries can be built on generic (primitive) ones with respect to an application.
- Operations, assertions, or invariants can be jointly shared by more than one class in a library. In this manner, associations between classes and their integrity rules may be specified.
- Class libraries exist at all levels of development, from system-level to application-level.
- Class libraries may deal with different levels of granularity.
- Class libraries can be a basis for standardization (e.g. Smalltalk and PDES/STEP)

[17] Use of the term “class library” blurs the distinction between “class” and “type” made in Section 3.2.5.

ODM class libraries share these general characteristics of class libraries. The principal implementation activity of an ODM application programmer may be to implement new ODM classes and methods, taking advantage of inheritance to **reuse** existing classes and methods. If the ODM requires use of special object definition and manipulation languages, classes will be specified and methods implemented in these languages.

As in programming languages like Smalltalk, where much of the semantics of the language is implemented not in the base interpreter but in class libraries, so also in some ODM systems, much of the ODM system functionality is available via a library of ODM **system** or **kernel classes**. As with ordinary classes, such kernel classes are extensible, permitting ODM system extensibility.

3.4.2 Application Program Interface and System Configurations

An **application program interface (API)** of a system, like an ODM, defines the syntax and semantics that allows an application written in a host programming language to control the system. Several variations in language characteristics can exist in ODM API:

- How many languages are used to write the application? Are the language(s) object based?
- Are ODM commands **embedded** in host language programs; or is the ODM API available through normal language capabilities (e.g. libraries of functions, class libraries, or macro libraries); or is the host language extended to make database operations appear as language extensions (e.g. through the addition of the keyword "persistent")?
- Is the API itself object-based, with requests like *begin-transaction*; is it function or macro-based (e.g. (*transaction ...body...*) in Common Lisp); or control-structure-based (e.g. an SQL-like "select-from-where" command for queries in C or C++).

In addition to these variations in the API, there are several possible architectures or system configurations for connecting a host programming language to an ODM system. These configurations may be characterized in terms of:

- Level of automation of persistence — What is the distinction, if any, between transient and persistent objects? Does the application programmer have to write explicit statements to save persistent objects?
- Number of object models — Is the data or object model in a host programming language different than or the same as (equivalent to) the ODM data model? Can the ODM support more than one data or object model? How is sharing across object models permitted? Are non-objects (e.g. arrays in C) first-class with respect to ODM operations like persistence or queries?[18]
- Number of object spaces — Is there more than one "heap" of temporary and/or persistent objects? How many different universes of object identifiers exist? Can objects reference other objects across object space boundaries?

[18] The term **seamless** is used when the data model of the host programming language and that supported by the database match, when automatic mappings from the host data model to ODM data model are supported, when the host language can be used as the ODM data manipulation language, and when the ODM API is a natural use of or extension to the host programming language.

- Number of operation execution spaces — How many execution spaces are there? Can operations execute in just the ODM space, just the application program space, or both?

Given these design choices, there are a number of different configurations of ODM system. For example, representative configurations include:

- Conventional language interfaces from C, C++, COBOL, Common Lisp, Smalltalk and Ada host languages via an embedding interface to an ODM; with programmer controlled persistence; with some programmer- controlled mapping between each host programming language data model and the ODM object model; with transient objects in the host environment and persistent objects in the ODM; with different data manipulation languages and execution spaces for operations written in the host language and in the ODM data manipulation language.
- ODM extensions are made for each supported host programming language, via libraries or language extensions; the ODM supports the object or data models of each host language directly and persistence is automatic, semi-automatic, and/or programmer-controlled; there are two object spaces (transient application space and persistent database space); the ODM uses the host language for its data manipulation language and can execute operations in either the host or ODM execution space; there is strong typing at the database to programming language interface; inter-language sharing is handled by the same mechanisms used to support cross-language remote procedure calls.

3.4.3 User Interfaces

A common characteristic of many ODM systems is that they include one or more user interfaces, in addition to application program interfaces. Detailed discussion of these interfaces is beyond the scope of this document. However, it is expected that a wide variety of interfaces will exist, each serving one or more user roles (see Section 3.4.5).

The user interfaces may be command-line-oriented, but often will be graphical and allow browsing and editing of object types, classes, and/or instances. A variety of generic user interfaces (e.g. forms or reports) and domain or application-specific interfaces will also be common. In some cases, ODMs may not make an application program interface available at all to some classes of users and may be manipulable only via user interfaces or turn-key applications.

3.4.4 Information Modeling

In the entire information modeling community, there is an increasing emphasis on "object-oriented" planning, requirements specification, analysis, design, and enterprise modeling.

In the context of an ODM system, object-oriented information modeling consists of analyzing an enterprise and its information management problems to produce a specification of classes at appropriate levels of detail. Thus the information modeler is identifying and designing classes. These classes are not usually specified in terms of detailed Object Language statements of the ODM; but a class defined at a conceptual level by an information modeler should have a direct mapping to one or more class definitions written by the application developer, class definer and/or database designer. In the course of defining classes, the information modeler and application developer take into account classes that already exist.

The use of classes at different stages in the development process of information-intensive systems is expected to lead to more information reuse, allow for more uniformity in development methodologies used at different stages of the development process, and make the boundary between information modeling and application development less distinct.

3.4.5 User Roles

It is useful to distinguish **user roles** within a system since different kinds of user may use or interact with a system in different ways. Of course, often the same person may take on different user roles at different times. Each user role represents use of the system at a different layer of abstraction. The roles of users of an ODM system may be characterized generally in terms of function, scope, and organization and specifically in terms of user interfaces employed, class libraries visible to the role, relationships to other roles, and placement of the role in an enterprise.

Under previous database paradigms, users of database systems might play one or more of the following roles. These same roles exist in the ODM paradigm:

- **End user** — accesses information directly through ad-hoc query interfaces, and indirectly through turn-key programs.
- **Application developer** — builds end-user applications.
- **Information or data modeler, system analyst** — identifies classes and analyzes operations of the application; performs tasks such as auditing, reporting, forecasting, and planning.
- **System installer or database administrator** — performs system administration functions such as installation, backup, tuning, system maintenance, etc.
- **System developer or maintainer** — implements system internals.

In addition, the ODM paradigm brings with it some new roles:

- **Class designer** [19] — specifies public class and method interfaces.
- **Method programmer** — implements methods.
- **Class library developer or vendor** — develops and/or maintains class libraries.

Since in ODM systems, both system and application objects may be defined as classes, these latter user roles cut across the traditional roles yielding roles like:

- **ODM systems programmer** — implements kernel objects; writes primitive methods such as access methods.
- **Application method programmer** — implements methods defining the operations of an application.

The taxonomy above is only representative since other user roles can be distinguished. For instance:

- **Domain class designer** — works with application communities like Mechanical CAD and Electrical CAD to define common object descriptions (e.g. PDES/STEP).
- **Information modeler** — works within an enterprise to develop conceptual classes that can provide a common model of the organization.

[19] See Footnote 17.

- **Class librarian** — maintains a data dictionary of class libraries.

A loose association between user roles and a layered organization of class libraries may be made. System designers implement ODM system kernel classes; domain or enterprise class developers develop widely useful classes; application programmers build on these to implement derived classes; and end users create or modify instances of classes through some end-user interface. This leads to a view of an ODM system as a **layered system**, where each layer corresponds to one or more class libraries built by one or more kinds of user. In such a system, system administration functions, for example, are not viewed as a fundamentally different set of functions and interfaces, but rather as just one of many class libraries containing specializations of functionality from a more general purpose class library.

3.4.6 Other System Characteristics

Other system characteristics include:

- **heterogeneity** — how portable and interoperable the ODM system is within or between different environments (machine, operating system, language, compiler, ODM system release, etc.).
- **openness** — how easy it is to reconfigure or add or replace parts of the ODM system.
- **performance, scalability** — how well the system scales to support large objects, large numbers of objects, large numbers of classes, distribution, and high-speed requirements.
- **industrial strength** — such factors as robustness, maintenance, availability, cost, licensing restrictions, reporting and database administration utilities, support for migration to/from legacy information system, support for data import/export.

SECTION 4

OBJECT DATA MANAGEMENT GLOSSARY

4.1 Introduction

This section summarizes the purpose of this report, lists the intended audience, and describes the organization of this document.

4.1.1 Purpose of Document

This document provides a set of *prescriptive definitions* of ODM terms, as used in the Reference Model for Object Data Management. These terms provide a common language for people to use when communicating about ODMs.

The value of a *prescriptive glossary* in standardization is to provide a common vocabulary so we all understand common terms the same way. In addition, glossary terms are important in the development of a reference model, where they have precise meaning. Finally, glossary terms provide a simple approximate way to scope a domain.

The ODM Glossary is not an exhaustive *descriptive* glossary of ODM terms, as used in specific systems. The value of a *descriptive glossary* in standardization is to list the range of variations of term definitions. A descriptive glossary is beyond the scope of this document.

4.1.2 Intended Audience

This document accommodates both the programming language and database perspectives of ODM systems and can be read by individuals in both communities. The intended audience for this glossary consists of:

- Those seeking to understand some of the main terms in the object paradigm.
- Those building or using ODM products and needing a common way to communicate about ODM systems.
- Those managing the standards process and influencing the direction of further work on standards in the area of ODMs.
- Those developing interoperability standards based on reference models for ODMs, and those developing more detailed, progressively refined glossaries in the ODM area or in ODM sub-areas, including X3/SPARC/DBSSG Glossary Task Force.

4.1.3 Relationship to Standards Activities

It is necessary to put this Glossary document into the context of the overall OODBTG charter and statement of work. This Glossary is a section of the the *Final Technical Report of OODBTG*. The *Final Technical Report* includes the following documents. See the Reference Model document for a description of the other documents.

- *Report on Recommendations for Standards in the Object Information Management Area*.
- *Reference Model for Object Data Management*.
- *Glossary for Object Data Management*.
- *A Survey of ODM Systems*.
- *Workshop Reports* covering two public workshops on the potential for ODM Standardization.

The various documents in the *Final Technical Report* have been influenced by liaisons with standards groups and industrial consortia (including Object Management Group (OMG), X3H2 SQL, X3H4 Information Resource Dictionary Systems (IRDS), X3T3 Open Distributed Processing (ODP), X3J16 C++) and various ODM-related workshops (including the DBSSG-co-sponsored Third Joint Standards Workshop, "Objects in Information Management", held at Anaheim, CA, January 14-15, 1991, and the DARPA-sponsored Workshop on Open OODB Architectures, held in Dallas, TX, March 13-15, 1991.)

Additional sources consulted in editing this glossary include:

- Thompson, C. (ed) *DARPA Open OODB Glossary*, Texas Instruments, Dallas TX, 1991 (DARPA contract DAAB07-90-C-B920).

4.2 Object Data Management Terms

—A—

abort — The transaction operation that a program uses to indicate that the transaction it is executing should be terminated abnormally and its effects should be obliterated. See **COMMIT.**

access control — A security procedure that validates a subject's right to perform an operation. See **AUTHORIZATION, DISCRETIONARY ACCESS CONTROL, MANDATORY ACCESS CONTROL, SECURITY.**

active object — An object that is capable of receiving and sending messages. For antonym, see **PASSIVE OBJECT.**

after-method — An operation (or side-effect) that occurs after an operation.

aggregate object — An object that is a collection of objects.

application program interface (API) — A definition of the syntax and semantics that allows an application written in a host programming language to access or control an ODM system.

assertion — A boolean expression serving as an integrity constraint that should be true when it is checked; otherwise, an error occurs. See **PRECONDITION**, **POSTCONDITION** and **INVARIANT.**

atomic operation — An operation that terminates consistently, meaning that it either performs all of its work or performs none of its work. An indivisible unit of work.

attribute — Object information manipulated by get and set operations, defined as a visible part of the state of an object. Also called **PROPERTY** or **INSTANCE VARIABLE.**

audit operation — In the context of **SECURITY**, an operation that records such events as success or failure of authentication attempts and authorization operations granted or denied.

authentication — The process of proving the identity of a subject (an object or person).

authorization — The process of determining whether a subject (an object or person) is permitted to perform certain operations or to access certain objects.

—B—

base class — A class from which one or more other classes are inherited. Also called a **SUPERCLASS.** See **GENERALIZATION.**

before-method — An operation (or side-effect) that occurs before an operation.

behavior — Perceived realization of actions.

binding — The selection of a method to be executed in response to an operation request. Binding occurs either at compile time (**early binding**), or at run time (**late binding**).

—C—

change management — A consistent set of techniques that control life cycle evolution, composition and policy management of the design and implementation of an object or system.

checkin/checkout — A concurrency control mechanism in which objects are copied from a database or reserved for use in a database (checked out), manipulated, then copied back to the database or released (checked in).

class — A definition of an implementation (methods and data structures) shared by a group of objects.

class library — A grouping of classes based on some relationship to or dependence on one another. Together, the classes usually implement some family of abstractions.

commit — The transaction operation that a program uses to indicate that the transaction it is executing should terminate normally and its effects should be made permanent. See **ABORT**.

computationally complete — A property of a programming language that includes sequence, selection (i.e. if-then-else), and iteration constructs, as well as other constructs, like assignment, function definition, etc.

concrete object model — An object model with a specified semantics and syntax. See **OBJECT MODEL**.

concurrency control — Mechanisms that control simultaneous sharing of objects among processes.

concurrency control protocol — A set of rules or conventions for controlling simultaneous access to objects.

configuration — In the context of change management, the composition or grouping of objects into a higher level object.

contract — A set of pre- and post-conditions that specify an operation.

create — An operation that establishes the existence of a new object.

—D—

data definition language (DDL) — A traditional term for the subset of the constructs of a database language in which the data model is declaratively expressed. See **OBJECT LANGUAGE**.

data dictionary — A collection of class/type definitions. Also called, **REPOSITORY**. See **SCHEMA**.

data manipulation language (DML) — A traditional term for the subset of the constructs of a database language in which the behavior is specified. A DML may or may not be computationally complete. See **OBJECT LANGUAGE**.

delegation — The concept of an object assigning or transferring to another object the responsibility for defining an object, performing an operation, or implementing state. Also called **INSTANCE-INSTANCE INHERITANCE**.

dependency — The reliance of one object upon another object for some behavior.

derived class — A class that inherits from one or more other classes. Also called a **SUBCLASS**. See **SPECIALIZATION, BASE CLASS.**

design space — A methodological tool that provides an organized way to record the characteristics and decisions of a design.

destroy — The operation of terminating the existence of an object.

discretionary access control — Access control that is granted or modified based on some subject's decision. See **ACCESS CONTROL** and **MANDATORY ACCESS CONTROL.**

distribution — (1) The action or process of placing or positioning an object in two or more places. (2) The action or process of placing information or components of a computer-based system in separate address spaces or computer systems.

—E—

embedded language — An language whose commands are placed directly in another (host) programming language.

encapsulation — Hiding representation and implementation in order to enforce a clean separation between the external interface of an object and its internal implementation.

error management — Techniques for handling errors that may occur during the execution of a computer program.

extensibility — (1) The ability to add new classes and subclasses to an existing schema. (2) The ability to add new attributes, methods, superclasses, etc. to a class. (3) The ability for existing instances to acquire or lose types.

extensional aggregate — An **AGGREGATE** whose identity is based upon membership. See **INTENSIONAL AGGREGATE.**

equality — A comparison operation whose definition may be class-dependent or system dependent.

—G—

garbage collection — A scheme used to manage and reclaim space associated with destroyed or unreachable objects.

generalization — The action or process of deriving from many objects a concept or principle that is applicable to all the objects. A base class(superclass) is a generalization of its derived classes (subclasses). For antonym, see **SPECIALIZATION.**

—H—

heterogeneous — (1) An ODM system supports heterogeneous environments if it supports simultaneous execution in different hardware and software environments, including different machines, operating systems, etc. . (2) An aggregate is heterogeneous if the objects it contains are instances of different classes/types.

homogeneous — An aggregate is homogeneous if the objects it contains are instances of the same class/type.

—I—

identity — A characteristic of an object that provides a means to uniquely denote or refer to the object independent of its state or behavior.

identity compare operation — An operation that determines whether two references refer to the same object.

impedance mismatch — The presence of incompatibilities between the data model of a programming language and that of a database. See **SEAMLESSNESS**.

implementation independence — A characteristic of an object that allows for its interface to be independent of its underlying implementation. See **ENCAPSULATION**.

inheritance — Deriving new definitions from existing ones.

instance — An occurrence of a class or type.

instance evolution — The process of making existing instances consistent with modified class definitions.

integrity constraint — Typically, a predicate that states a condition that must hold for a system or object to be in a legal state or that defines legal state transitions.

intensional aggregate — An **AGGREGATE** whose identity is based upon its creation event (and whose membership may change without affecting its identity). See antonym **EXTENSIONAL AGGREGATE**.

interface — The operations in which an object can participate. Also called **PROTOCOL**.

invariant — An operation-independent condition associated with a class or collection of classes that must always be true outside of an operation on this class or collection of classes. See **ASSERTION**.

—K—

kernel class — A class that is a central or essential part of a system. Also called **SYSTEM CLASS**.

—L—

layered system — A system where each layer corresponds to one or more class libraries built by one or more kinds of user.

LID — See **LOGICAL IDENTIFIER**.

life — The period of time during which an object exists. See **CREATE** and **DESTROY**.

literal — An object that has an eternal life, is neither created, destroyed, nor modified (i.e., it is constant) and is identified by itself.

lock — A mechanism often used to control the access to and sharing of an object.

logical identifier — An implementation-free conceptual token that is uniquely associated with a specific object. See **IDENTITY**, acronym **LID**, and **OBJECT IDENTIFIER**.

—M—

mandatory access control — Access control that is granted or modified based on some fixed policy. See **DISCRETIONARY ACCESS CONTROL**.

message — A **REQUEST** to a specific **RECIPIENT**.

metadata — See **SCHEMA**.

metadata evolution — See **SCHEMA EVOLUTION**.

method — The code specifying an implementation of an operation.

—N—

namespace — An environment where mapping between symbolic names and LIDs of objects is supported.

navigation — The act of getting from one object to another by applying a sequence of operations.

—O—

object — An instance of a class or type.

object identifier — A specific implementation of a logical identifier using a fixed format. Also called **OID**.

object language (OL) — A language used to specify and manipulate class definitions and object instances.

object model — A model that supports encapsulation, object identity, types, classes, behavior, inheritance, and instances.

object space — A scope or environment within which object creation and destruction occur. LIDs and OIDs are only meaningful within the object space in which they were created.

OID — See **OBJECT IDENTIFIER**.

operation — Something that can be applied to one or more objects in a **REQUEST**. See **METHOD**.

overloading — The process of implementing an operation in alternative methods. See also **POLYMORPHISM**.

override — Replacement of an inherited property or method with an alternative property or method.

—P—

parameterized type — A class constructor that takes parameters and returns one or more related classes (e.g., set<employee> might return “class set-employee,” a type corresponding to a set of employee class instances).

passive object — A kind of object that is not capable of sending messages. For antonym, see **ACTIVE OBJECT**.

persistent object — An object that exists longer than the process that created it. For antonym, see **TRANSIENT OBJECT.**

persistent language — A programming language in which the values of some or all data elements (data structures or class instances) survive an execution of a program written in the language.

polymorphism — A form of **BINDING** in which the same operation can bind to different implementations when sent to different objects.

postcondition — An assertion that must be true after an operation completes. If the assertion is not true, the operation fails.

precondition — An assertion that must be true before invoking an operation. If the assertion is not true, the operation cannot be performed.

propagation — The action or process of causing an operation applied to one object to be applied to other related objects.

property — See **ATTRIBUTE.**

protocol — See **INTERFACE.**

—Q—

query — A statement or operation written in a language, often predicate-based or declarative, specifying what data to retrieve from or update in a database. A query does not specify how to manipulate the data. Queries usually operate on sets or collections of instances and may return or affect a single object or a set or collection of objects.

—R—

reachable — An object is reachable if and only if there is an operation that returns the object as a result.

recipient — An distinguished argument of an operation that is the receiver of the operation.

recovery — The process of reproducing a consistent state of a system after a failure. In the context of an ODM system, the failure may be in an application process (whether embedded or not), the ODM system, the operating system, or the underlying hardware.

reference count — An implementation mechanism used to determine the number of currently existing references to an object.

reference model — A model that provides a framework to understand the key ideas and scope of a paradigm, to define the common characteristics of a system, and to provide a basis for comparing similar systems.

referential integrity — A kind of integrity constraint that guarantees that all referenced objects exist.

relationship — An association among two or more objects. If the relationship is between exactly two objects, it is termed **BINARY**. If the relationship is among three or more objects, it is termed **N-ARY**. Relationships may define direction and may be **UNIDIRECTIONAL** or **BIDIRECTIONAL**. A **cardinality** (allowed number of objects) may be specified for a relationship.

replication — The process of making copies of an object in more than one location or space, each of which are kept consistent (i.e., all copies will respond identically to identical requests).

repository — See **DATA DICTIONARY**.

request — The application of an operation to one or more objects. See **MESSAGE**.

—S—

schema — The collection of definitions of types, classes, and operations.

schema evolution — The process of altering the schema. Also called **METADATA EVOLUTION**.

seamlessness — Characteristics of ODM systems: (1) The data models of the database and the host programming language are the same syntactically and/or semantically; (2) The host DML can be used as the ODM DML; (3) The ODM API is a natural extension to the host programming language. See **IMPEDANCE MISMATCH**.

security — The measures taken to protect or guard an operation or an object from unauthorized actions.

side effect — A state change (or update).

signature — The specification of the number and types of an operation's arguments and results.

specialization — The action or process of adding a concept, principle, or operation to a class or type that is more specific or particular than other similar classes or types. For antonym, see **GENERALIZATION**.

state — The information that must be remembered when a request alters the future behavior of other requests.

subclass — A class that inherits one or more other classes. Also called a **DERIVED CLASS**. See **SPECIALIZATION**.

subtype — See **SUBCLASS** and **TYPE**.

superclass — A class inherited by one or more other classes. Also called a **BASE CLASS**. See **GENERALIZATION**.

supertype — See **SUPERCLASS** and **TYPE**.

synchronization protocol — The protocol that processes must follow to access an object concurrently. See **CONCURRENCY CONTROL PROTOCOL**.

system class — See **KERNEL CLASS**.

—T—

transaction — A control protocol in which a sequence of operation requests are grouped into an **ATOMIC** or indivisible unit of work. The transaction begins in a consistent database state and either **COMMITS** or **ABORTS** all state changes, leaving the database in a consistent state.

transformation — An aspect of change management that records or controls the propagating of consistent effects to related objects. See **DEPENDENCY**.

transient object — An object that lives no longer than the execution of the process that created it. For antonym, see **PERSISTENT OBJECT**.

transitive closure — (1) An operation that returns all objects reachable from an object. (2) An operation that returns all objects reachable from an object subject to a boundary condition.

transparency — The concept of hiding some or all details of an operation or behavior due to the operation being able to perform the operation without requiring the explicit details. Several examples are persistence transparency, syntactic transparency, fragmentation transparency, location transparency, replication transparency, execution transparency. While details may be hidden, there is often the need to control the orthogonal behavior (turn it off or on, specify some policy like eager or lazy, etc.).

trigger — An operation that should be performed when a specific method is performed or a specific event occurs.

type — The specification of a protocol (interface) shared by a group of objects, called instances of the type. See **CLASS** and **INSTANCE**.

—V—

version — A time-dependent variant of the state of an object.

4.3 Object Data Management Acronyms

ANSI	American National Standards Committee
API	Application Program Interface
ASC/X3	Accredited Standards Committee X3
CDIF	CASE Data Interchange Format
CFI	CAD Framework Initiative
CIS	CASE Integration Services
DBSSG	X3/SPARC Database Systems Study Group
DDL	Data Definition Language
DML	Data Manipulation Language
EDI	Electronic Data Interchange
EDIF	Electronic Data Interchange Format
EIS	Engineering Information System
Express	PDES/STEP object model
GUI	Graphical User Interface
IDL	Interface Description Language
IRDS	Information Resource Dictionary System
LID	Logical Identifier
MUMPS	Massachusetts General Hospital Utility Multi-Programming System
NIST	National Institute of Standards and Technology
ODA	Office Document Architecture

ODM	Object Data Management
ODP	Open Distributed Processing
OMG	Object Management Group
OID	Object Identifier
OIM	Object Information Management
OL	Object Language
OODB	Object-Oriented Database
OODBTG	X3/SPARC/DBSSG Object-Oriented Database Task Group
ORB	Object Request Broker
PDES	Product Data Exchange Standard
RDA	Remote Data Access
SPARC	X3 Standards Planning and Requirements Committee
SQL	Structured Query Language

SECTION 5

SURVEY OF OBJECT DATA MANAGEMENT SYSTEMS

As part of its workplan, the OODBTG undertook a survey of Object-Oriented DBMSs (ODMs). The purpose of the survey was to gather information on ODM systems from developers, vendors, and experts — to determine how terms are used, what design alternatives exist, how ODMs vary, and where there may be some consensus on the development of standards.

Funding was solicited from several selected organizations based upon a proposal document prepared by the University of Minnesota. The following organizations provided grants totalling $10,000 to support the survey effort. Their contributions are gratefully acknowledged.

- Texas Instruments
- Digital Equipment Corporation
- Ontologic
- U. S. National Institute of Standards and Technology (NIST)

Survey Instrument OODBTG developed an initial draft, edited by Craig Thompson (Texas Instruments), before the project was taken over by the University of Minnesota. At the University, the form was substantially revised with the help of members of the OODBTG. A copy of the final survey instrument is in Section 5.1. Recipients of the survey were 84 presumed vendors/developers of ODM systems. The survey form was sent out at the end of November 1990. A list of the survey recipients is in Section 5.2. This included about 20 vendors of conventional DBMS for whom it was not known whether or not they were developing an ODM system. Every effort was made to obtain the name of a responsible person for each recipient. No mailed surveys were returned due to an invalid address. Six recipients were non-responsive indicating they were not filling in the survey for some reason.

Respondents After some selective telephone follow-up, eight recipients responded in time to be included in the initial release of the raw survey data. The Raw Survey Results simply assembled the responses from each vendor under each question. They were distributed to the members of OODBTG at the 1991 April meeting. A ninth was received shortly after. Based on the raw survey results from these nine respondents, a list of suggested changes or additions to the draft *ODM Reference Model* and *Recommendations for Standards in OIM* documents was prepared. One of those respondents answered most survey questions with a reference to their own published technical summary. Hence, their response had less influence on the list of suggested changes. OODBTG processed the suggestions and revised these documents at the 1991 July meeting. To raise the response rate and provide additional input to the survey, OODBTG suggested a follow-up mailing to the non-respondents. This was done in 1991 June with the list expanded to include some experts in ODM technology. It resulted in three additional responses. Thus, a total of 12 recipients responded by filling in the survey form by the time this *Final Technical Report of the OODBTG* was prepared. We do

note that later respondents had at least the potential opportunity of seeing the raw survey results from the initial 8 respondents before preparing their own responses. Respondent efforts are gratefully acknowledged. Some responses were very substantive.

System Name	Vendor Name
KBMS	AICorp †
OBJECTIVITY/DB	Digital Equipment Corporation/Objectivity, Inc. †
IRIS	Hewlett-Packard †
CLORIS	IBM Thomas J. Watson Research Center ‡
ITASCA	Itasca Systems, Inc. ‡
O2	O2 Technology (Altair) †
OBJECT STORE	Object Design, Inc. ‡
GEMSTONE	Servio Corporation †
STATICE	Symbolics, Inc. †
ZEITGEIST	Texas Instruments †
SIM	Unisys ‡
VERSANT	Versant Object Technology †

†Early respondent serving as the basis for suggested changes and additions to the *ODM Reference Model* and *Recommendations for Standards in OIM* in the *Final Technical Report of the OODBTG*.

‡Provided a later response, thereby potentially having benefitted by seeing the Raw Survey Results from the eight initial respondents.

We make no endorsement of any of these systems. Some of them may not qualify as an Object-Oriented DBMS. They are all included simply because they responded to the survey. Readers are left to make their own judgment.

A Survey Analysis Report will be prepared based upon the expanded set of responses. This will include a paragraph for each question on the survey which summarizes the responses, and some overall observations, analysis, synthesis, summary, and conclusions. The results of this survey analysis will be made available as a technical report from the University of Minnesota about the end of 1991.

5.1 Survey Form

SURVEY OF
OBJECT-ORIENTED DATABASE MANAGEMENT SYSTEMS

INSTRUCTIONS FOR SURVEY RESPONDENTS:

1. Enclosed, for your convenience, please find a copy of the survey form on a 5 1/4 inch DSDD (360Kb) floppy disk. The SURVFORM file is saved in the following formats: WordPerfect 5.1 (W51) and WordPerfect 5.0 (W50) both assuming an EPSON printer, and ASCII/DOS (ASC). You may also request a copy of the survey to be sent by electronic mail. Your response can be sent on a floppy disk, on paper, or through electronic mail. Please send a hard copy printout of your electronic submission.
2. Please reply with the completed survey by December 14, 1990, if possible. Contact us with any questions or if you need more time to complete the Survey.
3. Some questions give you a set of **prescribed choices** for your response. The set of choices are always mutually exclusive (and often ranked in a natural progression), unless stated otherwise. Respond with a

```
                [ ]
check:          [X]     or circle:    RESPONSE1    ( RESPONSE2 )    RESPONSE3
                [ ]
```

   on your single choice. If the choices are not mutually exclusive, you are told to CHECK/CIRCLE ALL THAT APPLY.
4. Feel free to **add information** to any question with prescribed choices to further explain, clarify, or qualify your response.
5. Section 1 can be filled in directly on the survey form. Responses to the questions in sections 2 - 6 must be on separate sheets or in some electronic form of this survey.
6. Some questions are marked **OPTIONAL**. Answer them if you wish and have the time, but particularly if your response will contribute to a greater understanding of ODBMS concepts, architecture, or potential for standards.

ORGANIZATION OF THE SURVEY:

The survey is divided into the following parts:

1. General and Demographic information.
2. Opportunities for Consensus and Eventual Standards.
3. System Overview and Architecture:
4. Object Model — Definitional Concepts.
5. User Interface, Manipulation Languages, and System Operation.
6. Implementation Aspects. (OPTIONAL)

4 - 6 request detailed information about your system with an overview given in section 3. Section 2 is the most important for our purposes in conducting this survey. Please give it your most thoughtful response.

Thanks.

1. GENERAL INFORMATION

1.1 SYSTEM IDENTIFICATION

System Name: _____

1.2 **Current Status**: (OPTIONAL)

DESIGN UNDER DEVELOPMENT RELEASED to USERS Other:_____

1.3 **Date of First Release** to Users

_____

1.4 SYSTEM DEVELOPER

Organization Name: _____

Address: _____

_____

_____

Phone: (_____) _____-_____

1.5 **SCOPE** of Activity of the Above Organization

[ ] ODBMS only
[ ] Conventional DBMS Software primarily (including ODBMS initiatives)
[ ] Computer Hardware and/or Software Manufacturing and/or Sales
[ ] University or Research Lab

[ ] Other _____

1.6 **RESPONDENT** (person responsible for completing the survey form and contact for follow-up information or technical questions)

Name: _____

Title: _____

Address: _____

_____

_____

Phone: (_____) _____-_____

Fax: (_____) _____-_____

E-mail: _____

1.7 **PURPOSE OF IMPLEMENTATION** Identify PRIMary and SECondary purposes for this system. CIRCLE ALL THAT APPLY.

PRIM	SEC	for experimental or research purposes (to test ideas, feasibility, etc.)
PRIM	SEC	for use in an academic environment (to provide an educational learning platform)
PRIM	SEC	for in-house use
PRIM	SEC	for commercial distribution

1.8 APPLICATION AREAS expected to use your system, whether developed by you, third parties, or end users. For each application area below, indicate the status:

```
NOT EXPECTED  - not aware of nor expect any implementations in this
                area.
EXPECTED      - but no definite development plans yet.
IN DEVELOPMENT- resources committed to design, construction, test.
RELEASED      - for general use outside the development group.
```

Modeling (Generic)	NOT EXPECTED	EXPECTED	IN DEVELOPMENT	RELEASED
Design (Generic)	NOT EXPECTED	EXPECTED	IN DEVELOPMENT	RELEASED
CAD/CAM/CAE	NOT EXPECTED	EXPECTED	IN DEVELOPMENT	RELEASED
CIM (Manufacturing)	NOT EXPECTED	EXPECTED	IN DEVELOPMENT	RELEASED
GIS (Geog/Mapping)	NOT EXPECTED	EXPECTED	IN DEVELOPMENT	RELEASED
Imaging	NOT EXPECTED	EXPECTED	IN DEVELOPMENT	RELEASED
C.A.S.E.	NOT EXPECTED	EXPECTED	IN DEVELOPMENT	RELEASED
CAP (Planning)	NOT EXPECTED	EXPECTED	IN DEVELOPMENT	RELEASED
CAP (Publishing)	NOT EXPECTED	EXPECTED	IN DEVELOPMENT	RELEASED
Hypertext/Media	NOT EXPECTED	EXPECTED	IN DEVELOPMENT	RELEASED
Heterogeneous Database	NOT EXPECTED	EXPECTED	IN DEVELOPMENT	RELEASED
Multimedia/Video	NOT EXPECTED	EXPECTED	IN DEVELOPMENT	RELEASED
Simulation	NOT EXPECTED	EXPECTED	IN DEVELOPMENT	RELEASED
Finance/Business DP	NOT EXPECTED	EXPECTED	IN DEVELOPMENT	RELEASED
Office Automation	NOT EXPECTED	EXPECTED	IN DEVELOPMENT	RELEASED
Natural Language Processing	NOT EXPECTED	EXPECTED	IN DEVELOPMENT	RELEASED
Knowledge-Based /Expert Systems	NOT EXPECTED	EXPECTED	IN DEVELOPMENT	RELEASED
Other:_____	NOT EXPECTED	EXPECTED	IN DEVELOPMENT	RELEASED
_____	NOT EXPECTED	EXPECTED	IN DEVELOPMENT	RELEASED
_____	NOT EXPECTED	EXPECTED	IN DEVELOPMENT	RELEASED

1.9 OPERATING ENVIRONMENT: (OPTIONAL)

Hardware/Operating System **Platform(s)**:

```
Circle all that apply; add others below.

                                          DEC VAX___/ VMS
                                          IBM 309x  / MVS
_______________ / ___________             Sun______ / UNIX
                                          HP/Apollo / UNIX
_______________ / ___________             Apple Mac / Mac
                                          IBM-PC/AT / DOS
_______________ / ___________             IBM-PS/2  / OS/2
```

2. OPPORTUNITIES FOR CONSENSUS AND EVENTUAL STANDARDS. For purposes of our survey this section is the most important. Please give thoughtful responses to these questions. Return to this section after you have completed the rest of the survey.

2.1 What terms, capabilities, programmatic interfaces, languages, class libraries, interchange formats, benchmarks, etc. are common across ODBMSs and might be **candidates for industry consensus and eventual standards**?

2.2 Is your organization directly involved in or is your development effort aimed at **setting ODBMS standards** or guidelines for any community of users, developers, etc.? If so, describe.

2.3 Describe any **other standards activities** or efforts known to you which are directed at developing common, industry-wide specifications for ODBMS.

3. ARCHITECTURAL OVERVIEW. This section is designed to provide a brief overview of your system. Do not dwell too long on preparing your responses. You may refer to specific section or page numbers in previously prepared material where possible.

3.1 Please provide **TECHNICAL REFERENCES** to published papers, manuals, etc. and attach copies of relevant documents, if available.

3.2 List and then rank what you (and your user community) believe to be the **major distinguishing characteristics of an ODBMS**, i.e., what makes an object-oriented database management system different from an object-oriented programming language and a conventional DBMS? Rank the most important with a '1.' Do not limit your response to the characteristics we have included in sections 4 and 5 but develop your own list of the most important characteristics of an ODBMS.

RANK
[ ] _____
[ ] _____
[ ] _____
[ ] _____
[ ] _____
[ ] _____

3.3 Briefly describe your **OBJECT MODEL — Definitional Aspects**: the set of constructs and rules for creating/building a user's object model. You will provide greater detail in section 4. Your description here should use the same terms you define in answering the questions in section 4.

3.4 Briefly describe the **OPERATIONAL ASPECTS of your SYSTEM**; how it is used to create, maintain, and manipulate a user's object model. Describe the use of your Object/Data Definition Language and your Object/Data Manipulation Language(s). (Again, use the same terminology as in section 5, where you provide more detail.)

3.5 Provide a picture of the **SYSTEM ARCHITECTURE**; showing the major parts or modules and their interactions. Include only those features which are currently supported or the design of which has been incorporated into the overall architecture and integrated with other parts of the system.

3.6 Was your system architecture influenced by **Application Integration Frameworks** such as: CFI, OMG, EIS, or CIS? If so, describe how?

3.7 How did any of the **application environments** identified in question 1.8 influence the design and architecture of your system?

3.8 Do you recommend any **design methodology for data/object modelers** using your system? If so, describe the methodology or attach further information. Describe any database design aids you provide?

INSTRUCTIONS for SECTIONS 4. and 5.

The next two sections list concepts generally related to object-oriented database systems. Different systems/vendors may use different terms to describe these concepts. The purpose here is to discover *your* particular use of terminology relating to these concepts.

Relating to each concept below, tell us the terms *you* use, their definitions, and how they relate to each other. Try to arrange your responses around the list of concepts presented below. Focus on the concepts or categories rather than on *our* use of terminology. Consider the "Other and related terms" to be suggestive of terms you might define under that category. Tell us what *you* mean by these terms rather than trying to figure out what we mean. You need not have a definition for all the terms listed. Introduce and define additional terms as needed. Identify terms you consider to be synonymous with concepts and terms we have presented.

Define and use each term within the context of discussing each concept. Do not simply provide a glossary of terms (though such an alphabetical listing of terms and definitions can supplement your discussion of concepts).

4. OBJECT MODEL The following concepts deal with the set of constructs and rules which are used to define user object models — the *definitional* aspects of your object model. Section 5 deals with the user interfaces and languages provided by your system to *create*and *manipulate* user object models.

4.1 OBJECT: The primitive construct within the system that is the focus of storage and retrieval of information and manipulation actions (see OBJECT TYPE in Section 4.9). Other and related terms may be ENTITY, ABSTRACT DATA TYPE, DOMAIN.

4.2 OBJECT IDENTITY and OBJECT NAMING: Constructs used to identify and reference objects. How are object identifiers constructed? Other and related terms may be OBJECT ID (OID), UNIVERSAL ID, UNIQUE ID, PROXY, SURROGATE, INSTANCE, HANDLE, POINTERS, CONTEXTUAL, MUTABLE, LOCATION-INDEPENDENT, DATA-INDEPENDENT, OBJECT EQUALITY, SHALLOW EQUALITY, DEEP EQUALITY, OBJECT BOUNDARY, UNIT OF SHARING, TRANSITIVE CLOSURE, REFERENTIAL INTEGRITY. Can object ID's be reused after the associated object has been deleted?

4.3 OBJECT COMPOSITION: Constructs or mechanisms used to form other or more complex objects from primitive objects. Other and related terms may be COMPLEX OBJECTS, COMPOSITE OBJECT.

4.4 OBJECT CHARACTERISTICS: Constructs used to describe or characterize objects within the system. Other and related terms may be PROPERTY, ATTRIBUTE, RELATIONSHIP (discussed below, section 4.5), FUNCTIONS, OPERATIONS (discussed below, section 4.7). Describe any constraints or additional semantics which can be defined for these object characteristics, such as domain, multiplicity, cardinality, exclusivity, dependency, or exhaustivity.

4.5 RELATIONSHIPS: Objects may also be characterized by (or through) relationships with other objects. If your system supports the definition of relationships, indicate if they are n-ary or strictly binary, bidirectional or unidirectional, multi-valued or single-valued. Can a relationship have different names for each direction? Can relationships themselves have properties as objects do? Describe any constraints or additional semantics which can be defined for relationships.

4.6 INTEGRITY CONSTRAINTS: Additional semantics which can be defined in an object model pertaining to object characteristics and relationships, both static and dynamic (transitional) constraints. Other and related terms may be CONSISTENCY, RULES, TRIGGERS, ACTIVE DATA VALUES, DAEMONS, CONSTRAINT SYSTEM.

4.7 BEHAVIOR: Constructs used to prescribe object behavior. Other and related terms may be METHOD, MESSAGE, FUNCTION, OPERATION, OVERLOADING, GENERIC FUNCTION, VIRTUAL FUNCTION.

4.8 ENCAPSULATION: Combining the behavior of objects with the structural or state representation of objects. Other and related terms may be DATA ABSTRACTION, ABSTRACT DATA TYPES, user/program access to object behaviors or object structure/state, HIDING of object state or structure and behavior.

4.9 OBJECT TYPING: Constructs used to categorize or group objects. Other and related term may be CLASS, TYPE, GENERIC/PARAMETERIZED TYPES, PROTOTYPE, METACLASS, DATA TYPES, SCHEMA, TEMPLATE, SET, METADATA (may also be closely related to OBJECT COMPOSITION, see Section 4.3).

4.10 TYPE or CLASS GRAPHS: To represent inter-relationships among types. Other and related terms may be TYPE/CLASS LATTICE or HIERARCHY, BUILT-IN or USER-DEFINED SUBCLASSES, SUBTYPES and SUPERTYPES, EXTENSIBILITY of user-defined subclasses/types and methods.

4.11 INHERITANCE: Inheritance of characteristics, structure, constraints, and behavior. Other and related terms may be DELEGATION. Describe such issues as type-to-type, type-to-object, single vs multiple inheritance, conflict resolution, code-sharing vs set/subset.

4.12 Describe any **additional concepts or terms** which you use when describing the object model in your system.

5. USER INTERFACE, MANIPULATION LANGUAGES, and SYSTEM OPERATION

The following concepts deal with the operational aspects of your system — the object/data manipulation language(s), including the rules, mechanisms, services, and facilities for user/application interaction with the system to create, retrieve, manipulate, and maintain data/objects and metadata. The language to create the metadata (the schema) is often called the *Object Definition Language* to set it apart from the *Object Manipulation Language(s)* or User Interfaces used to create, access, and manipulate user application object databases. First, a little tutorial introduction to this section. For purposes of this survey, we categorize three major types of user interfaces to an ODBMS:

- **ODL**. *OBJECT DEFINITION LANGUAGE* — Object/Data Definition and Revision or Evolution; Metadata/Schema Creation and Manipulation Language. There may or may not be a clear distinction between the definition part of the language, which may be strictly declarative, and the part or facilities used to manipulate (access/query and modify) the object definitions or schema information. The ODL establishes the structure and vocabulary to be used in the two types of manipulation languages below. The key distinction here is that the ODL targets the creation (and possibly manipulation) of the schema, whereas the manipulation facilities below target the user object databases. Object (schema) definition (and manipulation) language facilities are the subject of questions 5.1 to 5.4. Use of the ODL may be restricted to an Object/Database Administrator.
- **POML**. *PROGRAMMATIC OBJECT MANIPULATION LANGUAGE* — to access and manipulate object/data (instances) from a user-written program; sometimes called Application Program Interface (API), and often characterized as procedural, assertional, or consisting of command strings. Used by application system developers to write programs for deferred execution. The POML is a *language* with which the users/system developers write *procedures* which tell the system what to do (later, when the procedure is executed). Hence, the qualification of *Programmatic* OML. The language generally consists of a set of verbs, commands, statements, messages, or procedure calls. The language may be "computationally complete" and, therefore, constitute a stand-alone programming language, or the verbs, etc. may have to be embedded in some host programming language (the host language providing the additional language constructs for execution control, etc. to make the combined language computationally complete.) The distinction between the POML (or other IOMI) and the ODL (Schema Manipulation Language) may be quite seamless operationally. The POML may also be augmented with facilities for defining and handling data display and data capture screens, forms, output reports, menus, help screens, etc.
- **IOMI**. *INTERACTIVE OBJECT MANIPULATION INTERFACES* — other user interfaces, generally for ad hoc, immediate, interactive access to a user's object/database. The facilities implied by both categories of POML and IOMI are for object manipulation. While most systems provide some form of programming language interface (POML), IOMI facilities are more optional. Furthermore, IOMI facilities come in much greater variety including menu (system-driven) interfaces and graphical, icon-driven interfaces (relaxing the procedural language characteristic of the POML). The languages in POML and IOMI may be very similar, the user being able to write commands into a program for deferred execution, or issue (many of) the same commands interactively for immediate execution. There may be no new functionality in the IOMI facilities, simply some new interfaces or "envelopes" within which the POML capabilities are made available. On

the other hand, the POML may be lower- level, less powerful with fewer capabilities, in which case the system may provide additional specialized or augmented facilities and languages for end- users to accomplish such functions as:

- set oriented operations on multiple instances,
- Screen, report, and menu definition and handling facilities,
- more general and flexible schema manipulation facilities, or
- high-level, ad hoc Object Query Language, recognizing that all of these could be provided by a comprehensive, higher- level POML.

5.1 METADATA as DATA: To what extent is the language/interface for creating and manipulating objects similar to or different from the language/interface for creating and manipulating metadata (schema information)? Are types, classes, relationships, functions, etc. first-class objects, that is, treated as objects in their own right through the object manipulation language?

5.2 OBJECT DEFINITION LANGUAGE: Describe the nature, use, and operation of the ODL, the interface by which the "user" (or object definer/administrator) creates object definitions (schema information). In a narrow sense, the ODL is a declarative language for telling the system to create object definitions. In a broader sense, the language may also be used to query and manipulate those definitions, hence the notion of a general Schema Manipulation Language (see the next question). Are methods defined as part of the schema/metadata? Can you add/delete new class/instance variables or methods dynamically?

5.3 Describe the **SCHEMA MANIPULATION FACILITIES** of your system. These are used to query, report on, modify, and control the evolution of the definitions in the object/database schema. They may be part of or extension to the ODL or they may be provided as separate facilities. How does your system support versioning of schema definitions? Other and related terms may be INSTANCE or CLASS INSPECTOR, INSTANCE MIGRATION, REDEFINITION, REVISION. (See Section 5.23 for versioning or change management of object instances in an application object/database).

5.4 RESTRUCTURING: When object definitions are changed, existing stored objects may not conform to the revised definition. How does the system effect and manage the changes required in the stored objects? To what extent is it a user responsibility and what facilities are provided by the system to assist the user? (closely related to VERSIONING and CHANGE MANAGEMENT, see Section 5.23).

5.5 How "**SEAMLESS**" is the boundary between the ODL (and schema manipulation language) and the manipulation facilities of the POML and the IOMI? Describe your understanding of seamlessness. Does it arise in other interfaces and operational aspects of your system?

INSTRUCTIONS

The following questions relate to the object manipulation facilities (referred to as OML below), primarily the Programmatic Object Manipulation Language (POML) or the Application Programming Interface (API) to the ODBMS. Some of them may also relate to end-user Interactive Object Manipulation Interfaces (IOMI). Give examples of syntax and associated semantics when appropriate.

5.6 Does your system have **MULTIPLE, DISTINCT OMLs** or APIs? If so, describe and differentiate each. Answer the following questions with respect to the richest facilities in the collection of language interfaces.

5.7 Is the POML a complete, **STAND-ALONE LANGUAGE or EMBEDDED** in a host programming language? Is it computationally complete? Explain. Describe any restrictions in the language used to access and manipulate objects. Is there a closure property? Is recursion allowed?

5.8 What **HOST PROGRAMMING LANGUAGES or APPLICATION PROGRAM INTERFACES** does your system support? How does a programmer call or invoke your ODBMS facilities? Give examples of the syntax.

5.9 An OML can be distinguished by the **LEVEL of OPERATIONS**. The user may operate at the level of single instances, requesting and manipulating one object/record at a time, or on multiple instances, referencing and manipulating sets of objects with single commands. Furthermore, a set of multiple object instances may be from more than one object type. Describe the single instance level operations in your OML. If the user can operate with sets of objects, describe the operations in the OML for retrieving and manipulating individual sets, and for manipulating (merging, differencing, etc.) multiple sets of objects. Explain if the facilities for manipulating sets of object instances are provided apart from the programmatic OML.

5.10 OBJECTS in SETS: Are OML operations or user queries always applied to sets/groups/collections/types of objects? Must every object (instance) be a member of some set (and thereby be queriable)? Can a user form a set by simply enumerating objects, as well as from the result of a retrieval operation? Are sets and classes/types strongly coupled? Can a set contain objects of different types or from different classes? Does a query about a class propagate to its subclasses? (See also question 5.17.)

5.11 Can a user **create and manipulate MULTIPLE SETS** or universes of objects using the OML or query language?

5.12 OML OPERATIONS: How does your OML support the operations of:

- selecting objects based upon object characteristics (given a predicate or Boolean selection expression in a "where" clause).
- projecting attributes/state of objects.
- ordering of object instances within a set or selected subset.
- joining object instances within/across sets.
- calculating statistics across a set of objects.

Give examples of the OML language syntax for these operations.

5.13 Does your system **support SQL**? How does your OML or end-user query/manipulation language/facility relate or compare to SQL?

5.14 How does your OML or end user query language support **NAVIGATION** through the object database? Give examples of path expressions, if supported. Describe any notions of inheritance along navigational paths? Other and related terms may be ASSOCIATIVE ACCESS, DIRECT REFERENCE, PATH, POINTER.

5.15 USER-DEFINED METHODS: Are methods part of the metadata/schema definition? In what language(s) can users write their methods (which define the behaviors of objects)? What commands and constructs of the Programmatic Object Manipulation Language, if any, can be used to write methods? Are user-defined methods allowed in the OML? Can the same class of user who invokes primitive operations such as create/read/update/delete also specify tailored operations on user-defined objects (such as Hire-an-Employee)?
5.16 Can methods used in the language have **side-effects**? If so, how do you guarantee database integrity?

5.17 Is the **RESULT OF A RETRIEVAL OPERATION** a first-class object that can be further referenced, manipulated, or queried? What is its class/type? How are its definitional (schema) characteristics determined? Is the result of a retrieval operation actual copies of the retrieved instances, or *references* (pointers) to the stored instances?

5.18 OBJECT VIEWS: How does your system support user views or subschemas? Must a user *always* reference/manipulate/query an object database through a view, or is it optional? How is an object view defined to the system? Can an object view contain multiple object types/classes? Does your system support incremental view update (dynamic views)?
5.19 OBJECT PERSISTENCE: How does a transient object become persistent? Is persistence at the type/class or instance level? Can an object have both persistent and transient methods? How can an application tell if an object is persistent? Other and related terms may be TRANSIENCE, REACHABLE OBJECTS, OBJECT CREATION AND DESTRUCTION, OBJECT DEPENDENCY.
5.20 Can all users **query/manipulate transient objects** or sets of objects? To what extent must the user of the OML or query language be cognizant of the transient or persistent state of objects? What happens if a persistent set contains transient elements? What happens if a persistent object references a transient object — are dangling references possible?

5.21 BINDING: How does your system associate or couple various components within the system, such as a message with an object or an operation on an object, a name with an object ID, a name with a memory location, or a message's argument with an operation's parameter. Other and related terms may be POLYMORPHISM, STATIC BINDING, DYNAMIC BINDING.
5.22 NAME SERVICE: Describe the name services and resolution provided by your system. Are object ID's accessible to application programs? Other and related terms may be local vs. remote names, aliases, name resolution from short to complete names, location transparency, site autonomy.
5.23 Are **VERSIONING or CONFIGURATION MANAGEMENT** facilities explicitly provided by your system? These facilities are for managing change in the user/application object/databases. (See Section 5.3 for schema change management facilities). Other and related terms may be CHANGE MANAGEMENT, VERSIONING, CONFIGURATION CONTROL, TRANSFORMATION, MULTIPLE REPRESENTATION.

5.24 CLASS LIBRARIES: Does your system provide class libraries or proforma schemas? How can they be modified and extended by the user?

5.25 Describe any **OTHER END-USER INTERFACE(s)** provided by your system, such as menu (system-driven) prompting interface, graphical user (query) interface, query-by-forms, report writer, hypermedia, other support for data entry and browsing, application generators, etc?

INSTRUCTIONS

The following set of concepts and questions are not unique to ODBMS but are nevertheless important to a fully functional object management system. Many of these facilities are generally provided by conventional DBMS. Consider these questions to be OPTIONAL, EXCEPT as the underlying facilities may be different in an ODBMS. Discuss how your system, being object-oriented, supports these concepts differently from a conventional DBMS. Also discuss any impact which the object-oriented paradigm has on the design or support of each concept. We want to discover any differences caused by being object-oriented.

5.26 TRANSACTIONS, CONCURRENCY CONTROL — optimistic, pessimistic, nested, long, other? locking, timestamps, read locks, write locks, exclusive locks, promotable locks, other? lock granularity? How are deadlocks handled? undo/rollback/redo? Do you permit transient transactions? Handling long (days, weeks) transactions.

5.27 ACCESS CONTROL, SECURITY — user identification, authentication, passwords, authorization, monitoring, etc.

5.28 BACKUP and RECOVERY, RESTART, RELIABILITY — logs? checkpoints?

5.29 PARALLELISM: Describe any explicit constructs provided by your system to allow the user to specify parallel or asynchronous processes.

5.30 DISTRIBUTION MODEL: How are sites introduced to each other? Are data dictionary entries distributed? cached? refreshed? or stored at a central site? Can distributed applications be built through function shipping? Do you support distributed transaction processing and consistent distributed commit?

5.31 IMPORT/EXPORT UTILITIES — bulk load, interfaces to data interchange formats like XDR or domain-specific data interchange formats like ODA or PDES?

5.32 Describe any **additional concepts or terms** which you use when describing the operational aspects of your system, user interfaces, and languages.

6. ODBMS IMPLEMENTATION STRATEGIES. (OPTIONAL)

The questions in this section are optional, except perhaps for the first one. They are intended to provide us with a better understanding of possible ODBMS architectures. Potentially standardizable interfaces may be derived from such architectures.

6.1 What are some *key ODBMS implementation issues* which you consider to be most significant and which may be relevant to the development of standards?

6.2 IMPLEMENTATION LANGUAGE(s) (languages in which your system is written):

```
C++      CLOS      SMALLTALK      OBJECT PASCAL      HOME-GROWN OOPL
Other OOPL: _____________
C      ASSEMBLY      HOME-GROWN non-OOPL      Other non-OOPL:_____________
FORTRAN       PL/I       PASCAL       Ada       SIMULA
Other Higher-Level PL:_________________
```

How did this choice influence the design of your system?

6.3 For implementing your **STORAGE MANAGER**, which of the following approaches has been taken?

```
[ ]    on top of an existing conventional DBMS
[ ]    built own conventional storage manager
[ ]    built an object-oriented storage manager
[ ]    Other, explain.
```

Describe the interface to your storage manager. What functions does your storage manager perform which directly relate to object management, beyond traditional data storage management?

6.4 Do you feel it is possible to produce a viable ODBMS by extending the modeling and manipulation facilities of a conventional relational DBMS (except perhaps from a performance perspective)? If not, what are the limitations?

6.5 How **MODULAR or LAYERED** is the architecture of your system? Can the system be configured with selected modules or layers? Can some functionality be added, replaced, or modified? Are module interfaces well-defined? Is the architecture like a database toolkit?

6.6 How **OPEN** is the system architecture? What is the extent of interoperability with other products and system components?

6.7 Do you support a **CLIENT/SERVER ARCHITECTURE**? Are servers multi-threaded (able to service multiple requests at the same time)?

6.8 In what ways is your system implementation **PORTABLE** to different hardware platforms, operating systems, programming language compilers, etc.?

6.9 How does your architecture relate to the **ANSI/SPARC three-schema architecture**?

6.10 OBJECT FAULTING: Relating to caching and sharing of objects. Is object faulting implicit or explicit? How does the system know which objects were modified during a transaction? How/when do you translate objects between their actual storage representation and their virtual memory representation? Is this translation compiler or machine dependent or is it portable? Can application programmers provide their own translation mapping?

6.11 What **ACCESS METHODS** are supported (B-trees, etc)? Are all objects stored in one big index? Can indices be created and destroyed? If an application can hold a pointer to an element of a set and modify the element, how are indexes on the set updated? Does your ODBMS use any novel features relating to disk storage and access methods?

6.12 Does **METHOD EXECUTION** take place in the user address space or the database address space (if they are different)?

6.13 QUERY OPTIMIZATION: Do you compile queries? When do you recompile? What statistics and parameters of the cost function are supported? How do you compile in the presence of methods?

6.14 Do you **CACHE RESULTS** of queries/methods and can you invalidate/recompute when necessary? Is recomputation lazy or greedy or user controlled?

6.15 Is there a **GARBAGE COLLECTOR**? What type? Are objects destroyed?

6.16 PERFORMANCE MONITORING AND ENHANCEMENT TOOLS: Do you support clustering, prefetching, training, decaching, statistics gathering, replication, other performance tuning techniques? How much do they improve performance? Can users tune these? Can objects be pinned in memory? Can the system fragment large objects? What kinds of hints are available to control the implementation or tune the system? What benchmarks or performance metrics do you use?

6.17 Describe **other significant ODBMS implementation strategies** or optimization used in your system.

5.2 List of ODM System Survey Recipients

Organizations believed to have some form of an Object-Oriented Database Management system (ODM) implemented or in development. Some are conventional DBMS vendors for whom it is unknown.

AICorp
Aion
Altair/O2 Technology
Ashton-Tate
AT&T Bell Laboratories
Associative Design Technology (MA)
Associative Design Technology (MI)
Athena Systems
BKS Software Entwicklungs GmbH
Borland
BULL H.N.
Cincom Systems
Computer Associates
Concurrent Computer Corp. (UK)
Concurrent Computer Corp. (NJ)
Data Access Corporation
Data General
DataEase International
Digital Analysis Corporation
Digital Equipment Corporation
Dome Software
Distributed Software Engrg Tools Corp.
GTE Laboratories, Inc.
Gupta Technologies, Inc.
Hewlett-Packard
Human Computers, Inc.
Infodata Systems, Inc.
Information Builders, Inc.
Information Dimensions, Inc.
Innovative Systems Technology, Inc.
Info. Impact International, Inc.
Informix Software Inc.
Instantiations, Inc.
Intel Corporation
IBM Corp, Santa Teresa Labs
IBM Corp, Almaden Research Center
IBM Corp, Research Lab
IBM Japan Ltd.
Itasca Systems, Inc.
Korea Adv. Inst. of Science & Tech
Mark V Systems Limited
The McKenna Consulting Group
Mentor Graphics
Microelectronics & Computer Tech. Corp.
MicroRIM, Inc.
Microsoft
Mind’s Eye
MITRE Corp
National Institutes of Health
NCR Corporation
Neuron Data
Object Databases, Inc.
Object Design, Inc.
Object Management Group, Inc.
Object Technology International, Inc.
Objective Systems
Objectivity, Inc.
On-line Software International, Inc.

Ontologic, Inc.
ORACLE Corp.
Progress Software
PTT Research
Relational Technology, Inc.
Revelation Technologies, Inc.
Servio Corporation
Shape Data Ltd.
Software ag International Inc.
Software Publishing
SYBASE, Inc.
Symbolics, Inc.
TeamOne Systems, Inc.
Teknekron Software Systems, Inc.
Texas Instruments
The RAND Corporation
Trinity College
Unify Corporation
UniSQL, Inc.
Unisys
University of Colorado
V 4th Company
Versant Object Technology
Vienna Software Publishing
XDB Systems, Inc.
Xerox Advanced Information Tech.(CCA)

5.3 Survey Data

The raw *Survey Results* and the *ODM Survey Analysis* (end of 1991) are available from:

Dr. Gordon C. Everest
MIS Research Center
Carlson School of Management
University of Minnesota
271 - 19 Avenue South, 355 HHH
Minneapolis, MN 55455 USA
VOICE: 612-624-0854
FAX: 612-626-1316
E-Mail: EVEREST@UMNSOM.BITNET

SECTION 6

WORKSHOPS ON OBJECT DATA MANAGEMENT STANDARDIZATION

6.1 Summary Report

6.1.1 Background

OODBs have reached the point where it makes sense to consider their potential for formal standardization. There are now several OODB products and substantial research prototypes. Several authors have stated requirements for OODBs and have offered definitions and initial specifications for consideration.

In January 1989, the Database System Study Group (DBSSG), one of the advisory groups to the Accredited Standards Committee X3 (ASC/X3), Standards Planning and Requirements Committee (SPARC), operating under the procedures of the American National Standards Institute (ANSI), established a task group on Object-Oriented Databases (OODBTG). To facilitate further development and use of OODB technology, OODBTG was commissioned:

- To define a common reference model for an Object-Oriented Database, based on object-oriented programming and database management system models.
- To assess whether and where standardization on OODBs is possible and useful. Some areas of possible standardization include glossary, reference model, operational model, interfaces, and data exchange.
- To complete a *Final Technical Report* in late 1991 containing recommendations regarding future ASC/X3 standards activities in the object-oriented database area, including how these standards would relate to existing standards.

6.1.2 Purpose

This report provides a summary of workshops on object database system standardization. The OODBTG organized two workshops in May and October 1990; a third workshop was sponsored by the DBSSG in January 1991; and a fourth workshop was sponsored by DARPA in March 1991. The purpose of the two OODBTG workshops was to identify areas where consensus of object standards may be possible and desirable in order to provide inputs to the two OODBTG reports:

- Reference Model for Object Data Management
- Recommendations for Standards in Object Information Management

The purpose of the third DBSSG workshop was to foster communication among standards groups whose work focuses on object-oriented solutions for information management problems. The purpose of the fourth DARPA workshop was to consider the requirements

and architecture for a modular OODB system; this workshop also provided inputs to the OODBTG reference model and recommendations documents.

6.1.3 Workshops Organized by X3/SPARC/DBSSG OODB Task Group

OODBTG organized two public workshops on OODB standardization in 1990.

- **First Workshop on OODB Standardization**, Atlantic City, NJ, May 22, 1990. This workshop, held one day before the annual International Conference on the Management of Data (SIGMOD'90), solicited public feedback from the database community. The call for participation attracted 22 papers. Fifty-five people participated in the workshop.

Program Chair:	Craig Thompson, Texas Instruments
Workshop Program Committee:	Gordon Everest, University of Minnesota Elizabeth Fong, NIST Bill Kent, Hewlett-Packard Laboratories Haim Kilov, Bellcore Ken Moore, Digital Equipment Corporation Allen Otis, Servio Corporation Mark Sastry, Honeywell Craig Thompson, Texas Instruments

- **Second Workshop on OODB Standardization**, Ottawa, Canada, October 23, 1990. This workshop, held on the third day of the Conference on Object Oriented Programming, Systems, Languages, and Applications (OOPSLA'90), canvassed the programming language community. The call for participation attracted 13 position papers covering various aspects where standardization on object database systems may be possible.

Program Chair:	Allen Otis, Servio Corporation
Workshop Program Committee:	Tim Andrews, Ontologic Haim Kilov, Bellcore Allen Otis, Servio Corporation Craig Thompson, Texas Instruments

The purpose of these workshops was to identify areas where consensus on OODBs is possible and desirable in a setting where authors could expect feedback and possible action on their ideas. These workshops focused on concrete proposals for language or module interfaces, exchange mechanisms, abstract specifications, common libraries, or benchmarks. The workshop announcements also solicited papers on the relationship of object database system capabilities to existing standards, including papers that questioned the wisdom of standardization.

As evidence of the wide-spread interest in OODB standardization, the two workshops together received papers and participation from:

- Australia, Canada, England, France, Italy, Japan, Republic of Korea, Netherlands, USA, West Germany.
- Altair O2, Ashton-Tate, AT&T, Bell Communications Research, Boeing, CAD Framework Initiative, Concurrent Computer Corporation, DARPA, Digital Equipment Corporation, Engineering Information Systems, Grumman, Hewlett Packard, Honeywell, IBM, MCC, McDonnell Douglas, Mitre, Mentor, NEC, NIST, Nixdorf, NOSC, Object Design Inc., Objectivity Inc., Object Sciences, Ontologic, Oracle Corporation, OSF, Servio Corporation, Summa International, Tandem Computers, Texas Instruments, USAF, US Army CECOM, Versant Object Technology, Xerox.

- Australian National University at Canberra, Columbia, Kyoto University, Stanford, Rensselaer University, University of Akron, University of Alberta, University of Maryland, University of Michigan, University of S.W. Louisiana, University of Texas at Austin, University of Toronto, University of Wisconsin, University of California at Berkeley, Monmouth College.

The workshops began with background material on the OODBTG, then shifted to selected presentations. A paper "Plan and Organization of the OODBTG", presented by Tim Andrews (Ontologic and, at that time, chairperson of X3/SPARC/DBSSG/OODBTG) reviewed the statement of work of OODBTG. A paper by Ed Stull (Summa International and Chair of X3/SPARC/DBSSG) provided "the big picture" of how OODBTG fits into the ISO and X3 standardization picture. The OODB area is related to a number of other standards groups, including X3H2 SQL, X3H4 Information Resource Dictionary Systems, X3T3 Open Distributed Processing, X3J16 C++ and X3J13 Common Lisp, as well as several industrial consortia working on application integration frameworks, including Object Management Group, CAD Framework Initiative, and USAF Engineering Information Systems.

An early draft of the *Reference Model for Object Data Management* was distributed to workshop attendees and reviewed at the workshop. The remainder of each workshop reviewed position papers on various aspects of OODB standardization. The papers in the workshops represent the opinions of the individual authors. These papers are neither approved standards nor recommendations of OODBTG. Neither OODBTG nor the Program Committee have made any judgments as to whether any topic of any paper complies with the ODM Reference Model currently under development by OODBTG.

At each workshop, participants were asked to provide feedback to identify and rank areas where they felt consensus may be possible and areas where roadblocks might block consensus. Results from their inputs are reflected in the document *Recommendations for Standards in Object Data Management*.

Proceedings of the two workshops have been published as follows:

- Craig Thompson and Elizabeth Fong (eds), *Proceedings of the First OODBTG Workshop on OODB Standardization*, Atlantic City New Jersey, May 22, 1990, NISTIR 4503, 299 p., (available from NTIS, order number PB91-159723, price $39, phone 800-553-NTIS).
- Allen Otis and Elizabeth Fong (eds), *Proceedings of the Second OODBTG Workshop on OODB Standardization*, Ottawa, Canada, October 23, 1990, NISTIR 4488, 150 p., (available from NTIS, order number PB91-157198, price $23, phone 800-553-NTIS).
- Special Issues on OODB Standardization, *International Journal of Computer Standards and Interfaces*, Elsevier, publication expected September, 1991. (Many of the papers from the original workshop proceedings were revised for publication in this special issue.)

6.1.4 Workshop Organized by X3/SPARC Database Systems Study Group

X3/SPARC Database Systems Study Group (DBSSG) sponsored a workshop in January 1991, for the purpose of fostering communication among standards groups whose work focuses on object-oriented solutions for information management problems.

- **Workshop on Objects in Data Management**, Anaheim, California, January 14-15, 1991, sponsored by X3/SPARC/DBSSG. Proceedings published by MITRE Corporation

(MITRE Report No. MP-91W00016), Center for Intelligent and Special Programs, 7525 Colshire Drive, McLean, Virginia 22102-3481 (contact Victoria Ashby, 703-883-6368).

Conference Chair:	Victoria Ashby, MITRE Corporation
Editors:	Elizabeth Fong, NIST Mary-Ellen Stull, Summa International, Inc.

Background

In January 1989, DBSSG sponsored the first joint meeting with X3T2 and X3H4 to promote interactions and identify potential overlap areas among developing database standards. A second joint meeting was held in January 1990 to include a look at potential overlap in areas which included OSI and data management standards developers.

Meeting Format

The two-day meeting was conducted in a workshop style consisting of an Opening Plenary, working group break-outs, and closing plenary.

The selection of working group areas of interest was not pre-determined, but generated by the participants through submission of issues prior to the meeting and submission of issues at the opening plenary. Seven areas of common interest that addressed these issues were identified:

- Inter-relationships of OODB, ODP, OMG, IRDS, DM, and EDI Reference Models
- Object Concept and Design (Naming and Data Element Design)
- Harmonization of Existing Standards within X3, Consortia Standards, and Information Standards
- Interoperability and Object Distribution
- Query Language: Object-Oriented QL, and Extensions to SQL
- Object Services (Class Library, Applications Domains, PDES, CAD, CFI, and Interfaces)
- Repository Requirements for Object-Oriented Systems

Who Attended

The 92 attendees consisted primarily of members of national and international standards technical committees. The national standards bodies participating were DBSSG, X3H2, X3H4, X3J16, X3L8, X3T3, X3T5, and SPC. The international standards bodies participating were SC 21 Working Groups from the United Kingdom, Japanese Standards Association from Japan, Canadian representative of WG3, Holland and Spain. There were also representatives from consortia interested in the object paradigm.

Brief Summaries of What Happened

Vicky Ashby, the Chair of this workshop, opened the meeting by welcoming the various X3 standards groups and other organizations who were participating. She noted that the expertise of these participants was imperative to provide fruitful discussions of objects in data management.

Ed Stull, Chair of DBSSG, reviewed the outcome of the previous Joint Meeting Two in Orlando in 1990. The proceedings for the second joint meeting are available from X3 Secretariat, CBEMA, 311 First St. N.W., Suite 500, Washington, D.C. 2001.

Opening Plenary presentations were as follows:

Dick Gibson, Chair X3	Challenges in Information Technology
William Kent, Vice-Chair OODBTG	Reference Model for Object Data Management
Ken Jacobs, X3H2	Object Databases and SQL
Steve Oksala, Chair X3/SPC	Strategic Planning
Jack Veenstra, Chair X3T3	Open Distributed Processing - A Distributed Platform

Closing Plenary

A final discussion was led by the following panel of experts, Dmitri Linkov, Jerry Winkler (X3H4), Ed Stull (DBSSG), Tim Andrews (OODBTG), Bill Kent (OODBTG), and Jack Veenstra (X3T3).

It was agreed that multiple object models exist, but the question remains, what is the common glue that holds them together. It was envisioned that this glue is a superstructure or super-set of these object models. It was agreed that meetings such as these, which bring together diverse communities who have a common concern, are important and useful.

6.1.5 DARPA Open OODB Workshop

A fourth workshop on object-oriented database systems, sponsored by DARPA, also played an important role in providing inputs to OODBTG documents, including the reference model, glossary, and recommendations for standards.

- Craig Thompson and David Wells, *Report on DARPA Open OODB Workshop I: Preliminary Architecture Workshop*, Dallas, TX, March 23-25, 1991. (Open OODB documents, including the Workshop I report, are available from Rita Doelling, doelling@csc.ti.com, 214-995-0308. Postscript and ASCII documents are available via anonymous ftp from "csc.ti.com [192.94.94.1]" directory "oodb".)

The workshop gathered forty attendees including researchers, system and module developers including OODB and language vendors, representatives from relevant standards groups and industrial consortia, application developers, end users, and DARPA/DoD representatives. Participants had expertise in OODB and conventional database design, software development environments, computer-aided design, computer-integrated manufacturing, operating systems, programming languages, networking, hypermedia, and knowledge engineering.

Along with documents on Open OODB requirements and architecture, the OODBTG reference model and recommendations documents were distributed to DARPA workshop attendees. A workshop evaluation form solicited feedback on both documents.

A second DARPA Open OODB workshop will be held on September 25-27, 1991, in Dallas, TX, to review preliminary reference models and interface specifications for each Open OODB module including: kernal, address space manager, object communications, object translation, transactions, change management, queries, and data dictionary. These documents are being developed under contract but are each being reviewed by working groups from the academic, DoD research and development, OODB vendor, standards, consortia, and user communities. Take together, these documents are complementary to the directions recommended by OODBTG in its recommendations document.

6.1.6 Conclusion

Based on the widespread participation in the four workshops described above, several communities are willing to participate in a consensus process that would lead to standards in the object data management area.

Existing information management standards organizations are moving to adopt improved standards that support object concepts. Continued and close coordination will be needed to insure that interoperability between related standards will be supported.

The OODBTG report "Reference Model for Object Data Management" reflects inputs from all four workshops and was developed for the purpose of characterizing object data management systems. The OODBTG report "Recommendations for Standards in Object Information Management" also reflects inputs from the workshops and the ideas and priorities of many individuals.

6.2 Calls for Participation

6.2.1 First OODBTG Workshop

- CALL FOR PARTICIPATION -
X3/SPARC/DBSSG OODB TASK GROUP WORKSHOP ON
STANDARDIZATION OF OBJECT-ORIENTED DATABASE SYSTEM
Trump Regency Hotel, Atlantic City, N.J.
May 22, 1990

Object-oriented database systems (OODBs) have reached the point where it makes sense to consider their potential for formal standardization. There are now several OODB products and substantial research prototypes. Several authors have stated requirements for OODBs and have offered definitions and initial specifications for consideration. The purpose of this workshop is to identify areas where consensus on OODBs may be possible and desirable in a setting where authors can expect feedback and possible action on their ideas.

OODB Task Group

In January 1989, the Database Systems Study Group (DBSSG), one of the advisory groups to the Accredited Standards Committee X3 (ASC/X3), Standards Planning and Requirements Committee (SPARC), operating under the procedures of the American National Standards Institute (ANSI), established a task group on Object-Oriented Databases (OODBTG).

To facilitate further development and use of OODB technology, OODBTG seeks:

- To define a common reference model for an Object-Oriented Database, based on object-oriented programming and database management system models.
- To assess whether and where standardization on OODBs is possible and useful. Some areas of possible standardization include glossary, reference model, operational model, interfaces, and data exchange.
- To complete a Final Report in 1991 containing recommendations regarding future ASC/X3 standards activities in the object-oriented database area, including how these standards would relate to existing standards.

OODBTG meets quarterly. Persons interested in OODBTG should contact Elizabeth Fong, National Institute of Standards and Technology (NIST), Tech. Bldg. A266, Gaithersburg, MD 20899 (301-975-3250).

WORKSHOP GOALS

The goal of this one-day workshop is solicit public input from the database community and begin to identify what aspects of OODBs may be candidates for consensus that can lead to standards. A companion workshop planned for OOPSLA'90 will canvas the programming language community.

We are seeking detailed position papers on OODB topics including the following:

- Object Data Model
- OODB System Architecture and interfaces
- Persistent Language
- Distributed Object Transport Protocol
- Data Manipulation Language

- Change Management
- Query Language
- Transactions

Papers should be specific, focusing on concrete proposals for language or module interfaces, exchange mechanisms, abstract specifications, common libraries, or benchmarks. They should identify capabilities that it may be possible to reach community consensus on. Papers on relationships of these OODB capabilities to existing standards are also welcome as are papers that question the wisdom of OODB standardization. OODB system designers and implementers are especially encouraged to share their practical experience where it can lead to consensus.

The workshop itself will begin with some background on OODBTG, then shift to presentations and/or panels with speakers invited based on paper submissions.

Workshop attendees will receive OODBTG draft documents (OODB Reference Model, etc) before the workshop. All contributed papers will be available to workshop participants and will be identified in the OODBTG document register. Following the workshop, selected papers will be bound into a workshop proceedings to be available as a National Institute of Standards and Technology (NIST) technical report. A report on the workshop will be published in ACM SIGMOD RECORD.

LOGISTICS

All parties interested in OODB standardization are invited to submit three copies of a position paper (5-10 pages long) by March 30, 1990, to:

```
OODBTG Standardization Workshop
ATTN:  Craig Thompson
Texas Instruments, Incorporated
PO Box 655474, MS 238
Dallas, TX  75265
telephone:  215-995-0347
email:  thompson@csc.ti.com
```

The position paper is a requirement for participation. Authors should include the Registration Card below attached to their submission. Authors of papers selected for presentation will be notified by May 4. Attendees who do not submit papers can register at the above address on a first-come, first-served basis before April 20 and will be notified by May 4 if their registration is accepted. Participation and attendance is limited by room size. All participants will be asked to pay a modest fee (under $50) at the door to cover workshop expenses.

The OODB Workshop takes place one day before the 1990 ACM Sigmod International Conference on Management of Data, May 23-25. A block of rooms is reserved for the night of May 21 for $90 (single/double)–please make reservations before April 25 and mention "X3/SPARC" to take advantage of the room rate. The Trump Regency hotel is adjacent to the Trump Plaza Hotel where the Sigmod Conference will meet.

```
---------------------------------------------------------------------------
REGISTRATION INFORMATION:  X3/SPARC/DBSSG OODBTG Standardization Workshop

  Name: _____________________________  Affiliation:_____________________
  Address:__________________________  Business Phone:__________________
        __________________________  Email:_____________________________
  City:_____________________________  Fax:_____________________________
  State:___________Zip:_____________
  Country:__________________________
```

Send to the above address.

6.2.2 Second OODBTG Workshop

CALL FOR PARTICIPATION
X3/SPARC/DBSSG OODB Task Group Workshop on
Standardization of Object Database Systems
Chateau Laurier Hotel, Ottawa, Canada
Tuesday, October 23, 1990

Object-oriented database (OODB) systems have reached the point where it makes sense to consider their potential for formal standardization. There are now several OODB products and many research prototypes. Several authors have stated requirements for such systems and have offered definitions and initial specifications for consideration. The purpose of this workshop is to identify areas where consensus on OODB systems may be possible.

The OODB Task Group

In January 1989, the Database Systems Study Group (DBSSG), one of the advisory groups to the Accredited Standards Committee X3 (ASC/X3), Standards Planning and Requirements Committee (SPARC), operating under the procedures of the American National Standards Institute (ANSI), established a task group on Object-Oriented Databases (OODBTG). To facilitate further development and use of object database technology, OODBTG seeks:

- To define a common reference model for an OODB, based on object-oriented programming and database management system models.
- To assess whether and where standardization of OODB systems is possible and useful. Some areas of possible standardization include glossary, reference model, operational model, interfaces, and data exchange.
- To complete a Final Report in 1991 containing recommendations regarding future ASC/X3 standards activities in the object database area, including how these standards would relate to existing standards.

OODBTG meets quarterly. Persons interested in OODBTG should contact Elizabeth Fong, National Institute of Standards and Technology (NIST), Tech. Bldg. A266, Gaithersburg, MD 20899 (301-975-3250).

Workshop Goals

The goal of this one-day workshop is solicit public input from the object database and programming language communities to help identify what aspects of object database systems may be candidates for consensus that can lead to standards. A companion workshop was held on May 22, 1990, coincident with the SIGMOD conference. A proceedings from that workshop will be available in mid-summer 1990, from NIST at the above address.

Position Papers

We are seeking detailed position papers on OODB topics including the following:

- Object Data Model
- OODB System Architecture and Interfaces
- Persistent Language
- Distributed Object Transport Protocol

- Data Manipulation Language
- Change Management
- Query Language
- Transactions

Papers should be specific, focusing on concrete proposals for language or module interfaces, exchange mechanisms, abstract specifications, common libraries, or benchmarks. They should identify capabilities that may be candidates for standardization. Papers on relationships of these OODB capabilities to existing standards are also welcome as are papers that question the wisdom of standardization. OODB system designers and implementers are especially encouraged to share their practical experience where it can lead to consensus.

The workshop itself will begin with a brief presentation on the current work of OODBTG, including the reference model. The balance of the day will consist of presentations and/or panels with speakers selected based on position paper submissions.

Workshop attendees will receive OODBTG draft documents (Object Data Management Reference Model, etc) before the workshop. All contributed papers will be available to workshop participants and will be identified in the OODBTG document register. Following the workshop, selected papers will be bound into a workshop proceedings to be available as a National Institute of Standards and Technology (NIST) technical report. A report on the workshop will be published in ACM SIGMOD RECORD.

All parties interested in object database standardization are invited to submit three copies of a 5 to 10 page position paper by August 10, 1990, to:

```
OODBTG Standardization Workshop
ATTN:  Allen Otis                    telephone:  503-690-3603
Servio Corporation                   email:    uunet!servio!otisa
15220 NW Greenbrier Pkwy., Suite 340         otisa@servio.slc.com
Beaverton, Oregon  97006
```

The position paper is a requirement for participation. Authors should include the Registration Card below attached to their submission. Authors of papers selected for presentation will be notified by September 15. Attendees who do not submit papers can register at the above address on a first-come, first-served basis before August 20 and will be notified by September 15 if their registration is accepted. Participation and attendance is limited by room size. All participants will be asked to pay a modest fee (approximately $30) at the door to cover workshop expenses.

This OODBTG Workshop takes place on the third day of the OOPSLA/ECOOP Conference, October 21-25, 1990. Participants are responsible for arranging their own hotel accommodations.

The Chateau Laurier (613 232 6411), the Westin (613 560 7000) and the Novatelle (613 230 3033) hotels are offering special rates for OOPSLA attendees.

6.3 Workshop Feedback Forms

This section is only available in hardcopy. Contact:

Elizabeth Fong
National Institute of Science and Technology
Building 225, Room A266
Gaithersburg, MD 20899
Tel: 301-975-3250
Fax: 301-590-0932
Email: fong@ise.ncsl.nist.gov

SECTION 7

OBJECT DATA MANAGEMENT BIBLIOGRAPHY

7.1 Goal

The goal of this document is to assist the reader in understanding the Object Data Management Reference Model and to provide a guide to further reading, without going into application-specific and implementation-specific details.

7.2 Roadmap to the Literature

Object design and development — in various forms — has been used since time immemorial. The concept of a "number", instead of "number of wolves" or even "number of animals", or the concept of a "subroutine" which dates back probably to Babbage's engine, are good examples. Therefore what follows will mostly deal with abstract object concepts and only then with their particular applications in design, programming languages, databases, networking, AI, etc. It is important to understand that abstract object concepts are application- and technology-neutral. Most computer users are already acquainted with and have used — perhaps inconsistently — these concepts within particular application areas, be it programming languages, databases, information modeling, networking, design, etc. The names as well as the presentation of these concepts may have been area-specific, but the use of abstraction helps to understand that these named concepts are essentially the same.

The main underlying concept in the design, development, and use of objects is the one of abstraction. For a person with a programming background, the probably best references are [Dijkstra 76], [Dijkstra 76-1], [Dijkstra 82], [Meyer 88], [Meyer 89], [Knudsen et al. 88], [Booch 91], [Hehner 84]. For a person with a programming language background, getting out of the traditional mindset and using a radically different language notation may be important. In this case, use of Smalltalk [LaLonde et al. 90] may certainly help. ([Stroustrup 86] may serve as an example of an evolutionary mindset.) For a person with a DBMS background, [Meyer 89], [Nierstrasz 88], [Tsichritzis 88], [Dittrich 90], [Kent 89], [Maier, Zdonik 90], [Hughes 91] may be of use, as well as the abovementioned programming-oriented references. And for a person with a "data modeling" background, [Booch 91], [Meyer 89], [Kilov 91], [Wand 89], [Wand et al. 89] provide a good start, again, together with programming-oriented and DBMS-oriented references and the understanding that "any discussion of data modeling is really a discussion of integrity" [Thompson 89]. A "system designer", especially, a person interested in promoting open systems [Overcoming Barriers 91], will find [Booch 91], [Wand et al. 89] and [Dijkstra 76-1] helpful. Both a system designer and a data (in fact, information) modeler should understand that precise and explicit contracts [Meyer 88, Morgan 90, OSI/NM Forum 91, Ince 88] are essential on all stages of system design and development. And, finally, general problems in management and economics of software development from a technical viewpoint are summarized in [Tsichritzis 90]. The other references in the following list deal with object concepts applied in some particular areas. Although they may not necessarily use the terminology, the

appropriate concepts are there. For additional reviews and reference lists, see [Kilov 89-1], [Kilov 89-2], [Mrdalj 90], [Vossen 91].

This is not an exhaustive reference list.

7.3 Bibliography

[**Atkinson et al. 89**] M.P.Atkinson, F.Bancillon, D.DeWitt, K.Dittrich, D.Maier, S.Zdonik. The object-oriented database system manifesto. *Proceedings of the First Deductive and Object-oriented Database Conference*, Kyoto, 1989, pp. 40-57.

[**Bertino 91**] E.Bertino, L.Martino. Object-oriented database management systems: concepts and issues. *IEEE Computer*, April 1991, pp. 33-47.

[**Beeri 91**] C.Beeri. Theoretical foundations for OODBs — a personal perspective. *IEEE Data Engineering*, Vol. 14, No. 2 (1991), pp. 8-12.

[**Booch 91**] G.Booch. *Object-oriented design with applications*. Benjamin/Cummings, 1991.

[**Codd 79**] E.F.Codd. Extending the database relational model to capture more meaning. *Transactions on Database Systems*, Vol. 4 (1979), No.4, pp. 397-434.

[**Cattell 91**] R.G.G.Cattell. *Object data management: object-oriented and extended relational database systems*. Addison-Wesley, 1991.

[**Committee 90**] The Committee For Advanced DBMS Function. Third-Generation Database System Manifesto. *SIGMOD Record*, Vol. 19, No. 3 (1990), pp. 31-44.

[**Dijkstra 76-1**] E.W.Dijkstra. The teaching of programming, i.e., the teaching of thinking. In: *Lecture Notes in Computer Science*, Vol.46. Springer Verlag, 1976.

[**Dijkstra 76**] E.W.Dijkstra. *A discipline of programming*. Prentice-Hall, 1976.

[**Dijkstra 82**] E.W.Dijkstra. *Selected writings on computing: a personal perspective*. Springer Verlag, 1982.

[**Dittrich 90**] K.R.Dittrich. Object-oriented database systems: the next miles of the marathon. *Information Systems*, 1990.

[**Hehner 84**] E.C.R.Hehner. *The logic of programming*. Prentice-Hall, 1984.

[**Hughes 91**] John Hughes. *Object-oriented databases*. Prentice-Hall, 1991.

[**Ince 88**] D.C.Ince. *An introduction to discrete mathematics and formal system specification*. Oxford University Press, 1988.

[**Joint Meeting 3-91**] Workshop on objects in data management. *Proceedings of the Third Joint Meeting sponsored by the Database Systems Study Group of ASC/X3/SPARC (January 1991)*. MP-91W00016. The MITRE Corporation, 1991.

[**Kent 79**] W.Kent. *Data and reality*. North-Holland, 1979.

[**Kent 89**] W.Kent. The leading edge in database technology. In: *Information system concepts: An in-depth analysis*. (Ed. by E.D.Falkenberg and P.Lindgreen). North-Holland, 1989, pp. 353-356.

[**Kent 91**] W.Kent. A rigorous model of object reference, identity, and existence. *Journal of object-oriented programming*, Vol. 4, No. 3 (June 1991), pp. 28-36.

[**Kilov 89-1**] H.Kilov. Reviews of object-oriented papers. *SIGMOD Record*, Vol. 18 (1989), No. 1, pp. 12-15.

[**Kilov 89-2**] H.Kilov. Reviews of object-oriented papers, 2. *SIGMOD Record*, Vol. 18 (1989), No. 4, pp. 50-55.

[**Kilov 90**] H.Kilov. From semantic to object-oriented data modeling. In: *Proceedings of the First International Conference on Systems Integration*, Morristown, NJ, April 1990. IEEE Computer Society Press, 1990, pp. 385-393.

[**Kilov 91**] H.Kilov. Generic information modeling concepts: a reusable component library. In: *TOOLS '91 (Proceedings of the Fourth International Conference on Technology of Object-Oriented Languages and Systems), Paris, 1991, pp. 187-201*. Prentice-Hall, 1991.

[**Knudsen et al. 88**]. J.L.Knudsen, O.L.Madsen. Teaching object-oriented programming is more than teaching object-oriented programming languages. In: *Lecture Notes in Computer Science*, Vol.322. Springer Verlag, 1988, pp. 21-40.

[**LaLonde et al. 90**] W.LaLonde, J.Pugh. *Inside Smalltalk*. Volume 1. Prentice-Hall, 1990.

[**Maier, Zdonik 90**] D.Maier, S.Zdonik. Fundamentals of object-oriented databases. In: *Readings in Object-oriented Database Systems*. Morgan Kaufmann, 1990, pp. 1-32.

[**Maier 91**] D.Maier. Comments on the "Third-Generation Database System Manifesto". *Oregon Graduate Institute working paper*. April 18, 1991.

[**Meyer 88**] B.Meyer. *Object-oriented software construction*. Prentice-Hall, 1988.

[**Meyer 89**] B.Meyer. From structured programming to object-oriented design: the road to Eiffel. *Structured Programming*, Vol. 1 (1989), No. 1, pp.19-39.

[**Morgan 90**] C.Morgan. *Programming from specifications*. Prentice-Hall, 1990.

[**Mrdalj 90**] S.Mrdalj. Bibliography of object-oriented system development. *ACM Software Engineering Notes*, Vol. 15 (1990), No. 5, pp. 60-63.

[**Nierstrasz 88**] O.M.Nierstrasz. A survey of object-oriented concepts. In: *Active Object Environments* (Ed. by D.Tsichritzis). University of Geneva, 1988, pp. 1-17.

[**OMG 91**] *Object management architecture guide*. 1.0 (November 1, 1990). OMG TC document 90.9.1. Object Management Group, Inc., 1990.

[**OSI — Data Management 90**] The convergence of Open Systems Interconnection and Data Management Standards (Ed. by Elizabeth Fong). *Proceedings of the Second Joint Meeting in Orlando, Florida, January 22-23, 1990*. March 1990, CBEMA (Distributed by ANSI X3 Secretariat), pp. 34-35.

[**OSI/NM Forum 91**] *Modeling Principles for Managed Objects*. Technical Report Forum TR102, Issue 1.0, January 1991. OSI/Network Management Forum, Bernardsville, NJ.

[**Overcoming Barriers — 91**] *Overcoming barriers to open systems information technology: First official report*. January 27, 1991. Distributed by User Alliance for Open Systems, McLean, VA.

[**Parent et al. 86**] C.Parent, S.Spaccapietra. Enhancing the operational semantics of the entity-relationship model. In: *Data and Knowledge* (Ed. by T.Steel, Jr., and R.Meersman). Elsevier, 1986, pp. 159-173.

[**Stroustrup 86**] B.Stroustrup. *The C++ programming language.* Addison-Wesley, 1986.

[**Thompson 89**] J.P.Thompson. *Data with semantics. Data models and data management*. Van Nostrand Reinhold, 1989.

[**Thompson 91**] C.Thompson. *Open OODB requirements.* Technical Report, Texas Instruments, 1991 (DARPA contract DAAB07-90-C-B920).

[**Tsichritzis 90**] D.Tsichritzis, S.Gibbs. From custom-made to pret-a-porter software. In: *Object management* (Ed. by D.Tsichritzis), University of Geneva, 1990, pp. 367-376.

[**Tsichritzis et al. 88**] D.C.Tsichritzis and O.M.Nierstrasz. Application development using objects. In: *Active Object Environments* (Ed. by D.Tsichritzis). University of Geneva, 1988, pp.18-31.

[**Vossen 91**] G.Vossen. Bibliography on object-oriented database management. *SIGMOD Record*, Vol. 20 (1991), No. 1, pp. 24-46.

[**Wand 89**] Y.Wand. A proposal for a formal model of objects. In: *Object-oriented concepts, databases, and applications* (Ed. by Won Kim and Frederick H. Lochovsky). Addison-Wesley, 1989. ISBN 0-201-14410-7, pp. 537-559.

[**Wand et al. 89**] Y.Wand, R.Weber. An ontological evaluation of systems analysis and design methods. In: *Information system concepts: An in-depth analysis*. (Ed. by E.D.Falkenberg and P.Lindgreen). North-Holland, 1989, pp. 79-107.

[**Wegner 90**] P.Wegner. Concepts and paradigms of object-oriented programming. *OOPS Messenger*, Vol. 1 (1990), No. 1, pp. 7-87.

APPENDIX A

PLAN AND ORGANIZATION FOR OODBTG

PLAN AND ORGANIZATION
FOR
OBJECT-ORIENTED DATABASE TASK GROUP (OODBTG)
Updated May 1991

OBJECTIVE

In recognition that the rapid emergence of a new information management technology, called Object Database Systems, and the lack of a common definition of these terms is causing confusion among the vendors, users and standards developers in the database community, the Object-Oriented Database Task Group seeks:

- To establish a working definition for the term "Object Database,"
- To establish the relationship between Object Database technology and "object-oriented" methods and technology in other fields, including programming languages, user interface methodologies and information modeling methodologies, and
- To establish a framework for future data management standards activities, both extensions to ongoing SQL and IRDS development, and related future standards.

These results are to be presented in the form of a Technical Report on Object Database Technology and a Recommendation For Standards Planning.

SCOPE

The Task Group will review existing and under development products claiming to be object databases or object-oriented products with database capabilities. The Task Group will also review published literature and research activities concerning object database technology world wide, and will seek to generate discussion among practitioners in the field.

OPERATIONAL APPROACH

The Task Group divided the work into five related areas:

- Glossary definition of object-oriented terms.
- Compilation of object-oriented bibliographic references.
- Identification of characteristics and features for object database management systems.
- Survey of commercial and research prototype object database management systems using the features identified.
- Reference model for object database management system.
- Issues relating to recommendations to data management standards planning.

The Task Group plans to meet its objectives in about two years. The final product will be the publication of the Technical Report and recommendations for standards planning.

The Task Group plans to hold quarterly meetings, in conjunction with DBSSG quarterly meetings when possible. Additional geographic subgroup meetings will be held to permit more technical work progression. Subgroups may be created on an ad-hoc basis by the Task Group Chairperson. The Task Group Chairperson will report quarterly, either directly or through a representative, to the DBSSG, and will include in the report a list of active subgroups and the issues being addressed, and a statement of progress made during the previous quarter.

Workshops and panels are also planned to generate discussion among practitioners in the database and programming language fields

PLANNED MILESTONES AND SCHEDULES

Jan 1989	Formation of the Task Group by DBSSG.
Jul 1989	Statement of issues to be addressed.
	Listing and classification of concepts going by the name Object Database.
Oct 1989	Outline for Reference Model and Technical Report.
Oct 1990	Initial draft tutorial collected.
	Initial features explained for the survey form.
Jan 1991	Initial draft Reference Model collected.
	Preliminary data collected for 4 OODBMSs.
Apr 1991	Draft Reference Model completed.
	Preliminary recommendations formulated.
	More survey data collected and analyzed.
Jul 1991	Draft Final Report ready for submission to DBSSG.
Oct 1991	Revise final report.
	Initiate standards projects in subareas.
Jan 1992	OODBTG disbanded after responding to questions and comments from SPARC and X3 Technical Committees.

OBJECT DATABASE TASK GROUP ORGANIZATION

Role	Individual(s)
Acting Chairperson[1]	William Kent, Hewlett-Packard Laboratories
Acting Vice Chairperson	Craig Thompson, Texas Instruments
Secretary	(designated at each meeting)
Correspondence Secretary	Elizabeth Fong, NIST
Reference Model Editor	Allen Otis, Servio Corporation
	Craig Thompson, Texas Instruments
Recommendations for Standards Editor	William Kent, Hewlett-Packard Laboratories
	Ken Moore, Digital Equipment Corporation

[1]Tim Andrews (Ontologic) served as chairman of OODBTG with William Kent (Hewlett-Packard) serving as vice-chairman throughout 1989 and 1990.

Role	Individual(s)
	Craig Thompson, Texas Instruments
OODBMS Survey Subgroup Liaison	Gordon Everest, U. of Minnesota
	Magdy Hanna, U. of St Thomas
Bibliography Subgroup Liaison	Mark Sastry, Honeywell Inc.
	Haim Kilov, Bellcore
Glossary Subgroup Liaison	Roy Gates, Rand Corporation
	William Harvey, Robert Morris College
	Mark Sastry, Honeywell Inc.

MEMBERSHIP

As specified in X3 Procedures, members of OODBTG serve as individual experts. The rules for OODBTG membership is the same as membership for DBSSG.

The OODBTG has divided its membership into "active" and "correspondence" members. The active members need to attend at least two consecutive meetings in order to be qualified as "active." "Correspondence" members are those who are interested in the activities of OODBTG, but may not wish to attend and participate in meetings.

APPENDIX B

STRUCTURE OF DOCUMENTS

```
Final Report (FR)..............................  Fong, Thompson
       TABLE OF CONTENTS.....................
       PREFACE...............................
  FR.  Final Technical Report (TR)...........  Moore, Thompson, Kent, Fong
            Cover Letter.....................  Kent, Thompson
       TR1. Introduction.....................  Fong, Thompson,
       TR2. Recommendations for Standards....
            Activity in Object Information...
            Management.......................  Moore, Thompson, Kent
       TR3. Reference Model for
            Object Data Management ..........  Thompson, Otis, Kent
       TR4. Glossary for.....................
            Object Data Management..........   Perez, Sastry, Otis
       TR5. Survey Report....................  Hanna, Everest
            1. Survey Form...................  Everest, Hanna, Thompson
            2. List of ODM Systems Surveyed..  Everest, Hanna
            3. Survey Data...................  Everest, Hanna
                                               (ref UMinn. MISRC Tech Rpt)
       TR6. Workshop Report..................  Thompson, Otis
            1. Calls for Participation.......  Thompson, Otis
            2. Workshop Proceedings..........  (ref NIST tech reports &
                                               Intl J of Standards and I/Fs)
            3. Workshop Evaluation Forms.....  Thompson
       TR7. Bibliography.....................  Kilov
  AP1. Plan and Organization for OODBTG......  Fong
  AP2. Structure of Documents................  Thompson, Moore
  AP3. Schedule..............................  Thompson
  AP4. Document Log..........................  Fong
  AP5. Minutes of Meetings ..................  Fong
  AP6. Membership Roster.....................  Fong
       INDEX................................
```

APPENDIX C

SCHEDULE

```
                      90       91                                      92
                      10 11 12 01 02 03 04 05 06 07 08 09 10 11 12 01
Meeting               31.......17.......21.......31.......21.......29

Final Report          ..........................v..f1....F2.......r1
 v  vote on document completion
 f1 report transmitted to DBSSG;
    public release and request for "public comment"
 F2 approve by DBSSG
 r1 review of "public comment" on completed report

Technical Report      ..............................t1.....T2
 t1 report to DBSSG
 T2 publish as NIST Tech Report

Recommendations       ...r...........r......r....RF
for Standards

Reference Model       ........r1.....r2.....r3...RF
 r1 revise for Anaheim
    distribution
 r2 revise for San Antonio
 r3 final inputs by 5/15
 RF final revision

Glossary              ............................GF

Survey                ...s1....s2.....s3.........SF
 s1 mail out
 s2 prelim results
 s3 draft report 4/5
 SF final report 6/30

Workshop Reports      ............WP.............W1
 WP publish proceedings as NIST Tech Report
 w1 draft summary report on both Workshops

Bibliography          .........b........b........BF
```

APPENDIX D

DOCUMENT LOG FOR OBJECT-ORIENTED DATABASE TASK GROUP

Number	Document
OODB91-01	Presentation viewgraphs by David Beech from ORACLE.
OODB91-02	Presentation viewgraphs by José Blakeley from Texas Instruments on Object Query Language (OQL[C++]).
OODB91-03	Thompson, C., ''Open OODB Requirements,'' Technical Report, Texas Instruments, Dallas, TX March 1991 (DARPA Contract DAAB07-90-C-B920).
OODB91-04	NISTIR 4503, *Proceedings of the OODBTG Workshop*, Atlantic City New Jersey, May 22, 1990.
OODB91-05	NISTIR 4488, *Proceedings of the OODBTG Workshop*, Ottawa, Canada, October 23, 1990.
OODB91-06	Bellcore, Information Modeling Concepts and Guidelines, *Bellcore Special Report SR-OPT-001826*, Issue 1, January 1991.
OODB91-07	Thompson, C. ''Open OODB Glossary,'' Technical Report, Texas Instruments, Dallas, TX May 1991 (DARPA Contract DAAB07-90-C-B920).
OODB91-08	Fong, E. (ed.) *Proceedings of the Workshop on Objects in Data Management*, 3rd Joint Meeting sponsored by Database Systems Study Group, in Anaheim, CA January 1991. (Available from MITRE Crop. 7525 Colshire Drive, McLean, VA 22102-3481).
OODB90-01	Thompson, Craig, et. al., ''Open Architecture for Object-Oriented Database Systems,'' Technical Report 89-12-01, Information Technology Laboratory, Texas Instruments Incorporated, December 6, 1989.
OODB90-02	Kent, William, ''The Evolving Role of Database in Object Systems,'' Draft Paper, Hewlett-Packard Laboratory, Feb. 9, 1990.
OODB90-03	Kilov, Haim, ''Reviews of Object-Oriented Papers, 1,'' *SIGMOD Record*, March 1989.
OODB90-04	Kilov, Haim, ''Reviews of Object-Oriented Papers, 2,'' *SIGMOD Record*, December 1989.
OODB90-05	Kilov, Haim, ''From Semantic to Object-Oriented Data Modelling,'' in *Proceedings of the First International Conference on Systems Integration*, New Jersey, April 1990.
OODB90-06	Olle, William T. ''Programming Language, Database and Object-Oriented Paradigms,'' Comments on the OODBTG Reference Model, Feb. 1990.
OODB90-07	Everest, Gordon, ''Requirements for the Next Generation of DBMS,'' *Proceedings of NCGA-C4 Conference*, Santa Clara, CA September 1989.
OODB90-08	Stonebraker, Michael, et al., ''Third Generation Database System Manifesto,'' paper submitted to OODBTG Workshop, Atlantic City, NJ, May 1990.
OODB90-09	Dittrich, K.R., ''Object-Oriented Database Systems: The Next Miles of the Marathon,'' draft paper to be published, 1990.
OODB90-10	Wand, Yair, ''A Proposal for a Formal Model of Object,'' in Kim, W. and Lockovsky, F. (eds.) *Object-Oriented Concepts, Databases, and Applications*, ACM Press, Addison-Wesley Publishing Company, 1989.
OODB90-11	Kilov, Haim, ''Some Comments on the O-O Reference Model,'' Notes for discussion at the OODBTG, April 1990.
OODB90-12	Kent, William, ''A Framework for Object Concepts,'' draft paper to be published, 1990.

Number	Document
OODB90-13	Everest Gordon, ''The Object-Oriented Paradigm: Driving the Future of 'Database' Management Systems,'' draft paper to be published, 1990.
OODB90-14	Blakeley et al. ''Strawman Reference Model for Object Query Language,'' *OODBTG Workshop on OODB Standardization*, Atlantic City, NJ, May 22, 1990.
OODB90-15	Joseph et al. ''Strawman Reference Model for Change Management of Objects,'' *OODBTG Workshop on OODB Standardization*, Atlantic City, NJ, May 22, 1990.
OODB90-16	Perez. ''A Strawman Reference Model for an Application Program Interface to an Object-Oriented Database,'' *OODBTG Workshop on OODB Standardization*, Ottawa, Canada, Oct. 23, 1990.
OODB90-17	Wang. ''A Strawman Reference Model in Transaction Processing for an Object-Oriented Database,'' *OODBTG Workshop on OODB Standardization*, Ottowa, Canada, Oct. 23, 1990.
OODB90-18	Fong, E. ''Proposal for the Establishment of a Composite Technical Committee for the Development of Object-Oriented Paradigm Standards,'' October 5, 1990.
OODB89-1R	Otis, Allen (ed.), A Reference Model for an Object Database, Working Document available from Elizabeth Fong, NIST, Technology Bldg, A 266, Gaithersburg, MD 20899, March 18, 1989. (Version R8)
OODB89-02	Thompson, Craig, *Survey of OODB Systems*, August 4, 1989.
OODB89-03	Snyder, Alan, *The Essence of Objects*, STL Report 89-25, HP Labs, September 22, 1989.
OODB89-04	Snyder, Alan, Walt Hill, and Walter Olthoff, *A Glossary of Common Object-Oriented Terminology*, STL Report 89-26, HP Labs, September 22, 1989.
OODB89-05	Tsichritzis, D. and A. Klug (eds.) *The ANSI/X3/SPARC DBMS Framework*, Report of the Study Group on Database Management Systems, Pergamon Press, 1978.
OODB89-06	Joseph, J., S. Thatte, C. Thompson, and D. Wells, ''Report on the Object-Oriented Database Workshop,'' To appear in the *ACM SIGMOD Record*, 1989.
OODB89-07	Joseph, J., S. Thatte, C. Thompson, and D. Wells, ''Object-Oriented Databases: Design and Implementation,'' Texas Instruments report, 1989.
OODB89-08	Kim, Won, Object-Oriented Database Systems: Mandatory Rules and Prospects, to appear in *Datamation*, January 15, 1990.

APPENDIX E

MINUTES OF MEETINGS

This section is only available in hardcopy. Contact:

Elizabeth Fong
National Institute of Science and Technology
Building 225, Room A266
Gaithersburg, MD 20899
Tel: 301-975-3250
Fax: 301-590-0932
Email: fong@ise.ncsl.nist.gov

APPENDIX F
MEMBERSHIP ROSTER

This section is only available in hardcopy. Contact:

Elizabeth Fong
National Institute of Science and Technology
Building 225, Room A266
Gaithersburg, MD 20899
Tel: 301-975-3250
Fax: 301-590-0932
Email: fong@ise.ncsl.nist.gov

INDEX

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X