NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

OBJECT-ORIENTED DATABASE MANAGER
FOR THE
LOW COST COMBAT DIRECTION SYSTEM

by

Debra L. Ross

December 1989

Thesis Advisor Michael L. Nelson

REPORT DOCUMENTATION PAGE

1a Report Security Classification: Unclassified		1b Restrictive Markings	
2a Security Classification Authority		3 Distribution Availability of Report: Approved for public release; distribution is unlimited.	
2b Declassification Downgrading Schedule			
4 Performing Organization Report Number(s)		5 Monitoring Organization Report Number(s)	
6a Name of Performing Organization: Naval Postgraduate School	6b Office Symbol (*if applicable*): 37	7a Name of Monitoring Organization: Naval Postgraduate School	
6c Address (*city, state, and ZIP code*): Monterey, CA 93943-5000		7b Address (*city, state, and ZIP code*): Monterey, CA 93943-5000	
8a Name of Funding Sponsoring Organization	8b Office Symbol (*if applicable*)	9 Procurement Instrument Identification Number	
8c Address (*city, state, and ZIP code*)		10 Source of Funding Numbers	
		Program Element No	Project No / Task No / Work Unit Accession No

11 Title (*Include security classification*): OBJECT-ORIENTED DATABASE MANAGER FOR THE LOW COST COMBAT DIRECTION SYSTEM

12 Personal Author(s): Debra L. Ross

13a Type of Report	13b Time Covered	14 Date of Report (*year, month, day*)	15 Page Count
Master's Thesis	From To	December 1989	64

16 Supplementary Notation: The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17 Cosati Codes			18 Subject Terms (*continue on reverse if necessary and identify by block number*)
Field	Group	Subgroup	object-oriented database manager, combat direction system

19 Abstract (*continue on reverse if necessary and identify by block number*)

A modern Combat Direction System (CDS) must have the capacity to perform real-time processing of tactical data from more than twenty interfaces. These include multiple tracking sensors, multiple weapons interfaces, electronic warfare, and multiple tactical data link systems. Efficient processing and display requirements in such sophisticated systems have greatly increased the development cost of fielding new systems. The Navy wishes to develop a Low Cost Combat Direction System (LCCDS) which could be installed on non-combatant ships or could augment the processing capability on ships currently equipped with CDS systems. During the initial portion of Increment One of the LCCDS project, a suitable object-oriented database management system (OODBMS) must be chosen to form the basis for the remainder of the project. The goal of this thesis is to determine the feasibility of using a commercially available OODBMS. Evaluation of three systems shows that GemStone, a product of Servio Logic Corporation, Alameda, CA, best fits the requirements of this project.

Keywords: theses; NAVAL TACTICAL DATA SYSTEMS; Management Information systems; Computer Systems. (CP)

20 Distribution Availability of Abstract	21 Abstract Security Classification	
☒ unclassified unlimited ☐ same as report ☐ DTIC users	Unclassified	
22a Name of Responsible Individual: Michael L. Nelson	22b Telephone (*include Area code*): (408) 646-2026	22c Office Symbol: 52NE

DD FORM 1473,84 MAR — 83 APR edition may be used until exhausted; All other editions are obsolete — security classification of this page

Unclassified

Approved for public release; distribution is unlimited.

Object-Oriented Database Manager
for the
Low Cost Combat Direction System

by

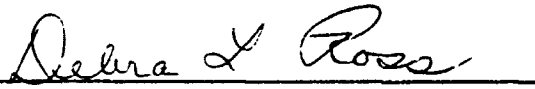
Debra L. Ross
Lieutenant, United States Navy
B.S., Florida International University, 1980

Submitted in partial fulfillment of the
requirements for the degree of

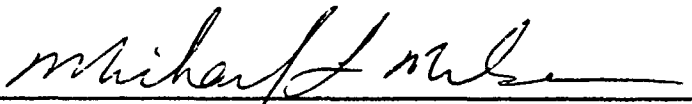
MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1989

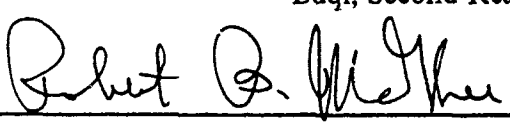
Author: 

Debra L. Ross

Approved by: 

Michael L. Nelson, Thesis Advisor

Luqi, Second Reader



Robert B. McGhee, Chairman,
Department of Computer Science

ABSTRACT

A modern Combat Direction System (CDS) must have the capacity to perform real-time processing of tactical data from more than twenty interfaces. These include multiple tracking sensors, multiple weapons interfaces, electronic warfare, and multiple tactical data link systems. Efficient processing and display requirements in such sophisticated systems have greatly increased the development cost of fielding new systems. The Navy wishes to develop a Low Cost Combat Direction System (LCCDS) which could be installed on non-combatant ships or could augment the processing capability on ships currently equipped with CDS systems. During the initial portion of Increment One of the LCCDS project, a suitable object-oriented database management system (OODBMS) must be chosen to form the basis for the remainder of the project. The goal of this thesis is to determine the feasibility of using a commercially available OODBMS. Evaluation of three systems shows that GemStone, a product of Servio Logic Corporation, Alameda, CA, best fits the requirements of this project.

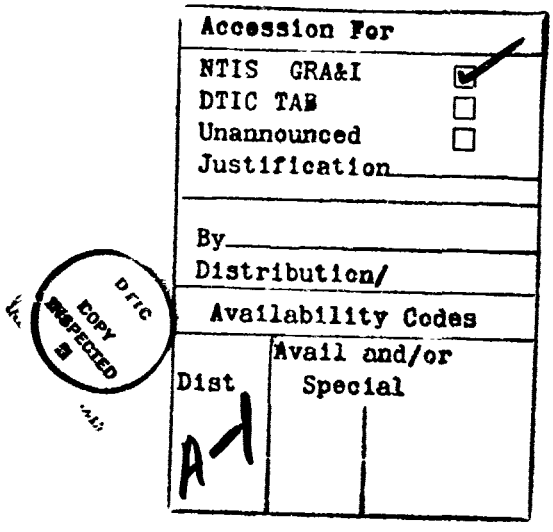


TABLE OF CONTENTS

LIST OF FIGURES

TABLE OF ABBREVIATIONS

AI .....artificial intelligence

CAD .....computer-aided design

CASE .....computer-aided software engineering

CDS .....Combat Direction System

CIM .....computer-integrated manufacturing

CLOS .....Common Lisp Object System

COP ....."C" Object Processor

DBMS.....database management system

ITS .....Integrated Toolset

LCCDS.....Low Cost Combat Direction System

NAVSEA ....Naval Sea Systems Command

NGCR .....Next Generation Computer Resource

NTDS.....Naval Tactical Data System

OM .....object manager

OO .....object-oriented

OODBMS .....object-oriented database management system

OOP.....object-oriented programming

OPE.....OPAL Programming Environment

OSQL.....Object SQL

PCLOS.....Persistent Common Lisp Object System

TDL.....Type Definition Language

I. INTRODUCTION

A. BACKGROUND

A modern Combat Direction System (CDS) must have the capacity to perform real-time processing of tactical data from more than twenty interfaces. The development of CDS have become an increasingly complex and expensive process.

Over thirty years ago it became obvious that the magnitude of operational data which must be assembled and analyzed had reached unworkable proportions. A definite need existed for a system that could gather the data, process it into meaningful and useful information, and present it in a useful manner to facilitate decision making. This need led to research which resulted in the Naval Tactical Data System (NTDS).

1. Naval Tactical Data System

The NTDS is defined as follows:

> Naval Tactical Data System (NTDS) consists of a complex of units of data inputs, user consoles, converters, adapters, and radio terminals interconnected with high-speed, general-purpose computers and their stored programs. Combat data are collected, processed, and composed into a picture of the overall tactical situation that enables the force commander to make rapid and accurate evaluations and decisions. [Ref. 1: p. A-4]

The original NTDS was designed with a building block system concept which the developers hoped would provide a long lasting and versatile system. Ideally, each ship was to be equipped with a NTDS system and have the capability to operate independently or as a unit of a task force. A modular design using the building block concept was followed. This allowed changes in varying requirements among ship types and changes to sensors and weapon systems to be accommodated over the long haul. System components were designed with flexibility to meet requirements of additional warfare missions. The system was to be reliable and able to operate continuously without interruption. Additionally, the system was flexible enough to accommodate evolutionary changes in tactics, weapons, or sensors. [Ref. 2: pp. 54-55]

When the NTDS was first deployed in the 1960's, it faced many problems including the NTDS integration role, shipboard computer configuration, and NTDS standardization. The NTDS generated inconsistent tactical information which reflected the limitations of digital technology of the time. The NTDS design was heavily biased towards data conversion and limited to serving as a conduit through which personnel

collected, processed and acted on tactical information. There were uncoordinated changes in the interfacing subsystems which compelled a continual catch-up design mode. The integration which resulted was not a functionally cohesive design. By the mid-1970's, dissimilar functions were assigned to the NTDS and many established NTDS functions had migrated to other systems. The role of the NTDS had become ambiguous. Uncontrolled shipboard computer configuration problems were further compounded by subsystem integration requirements. The NTDS had to contend with difficult design and costly life cyle support problems in an era of rapid technological advances in computer systems, displays, and communications. [Ref. 1: pp. 5-1 - 5-5]

The basic system building blocks of the NTDS, installed and in operational use on several hundred ships for over 25 years now, have remained essentially the same. Because designers followed the original system concepts, the overall system has demonstrated its inherent flexibility by accommodating numerous changes in sensors, weapon systems, and the tactical environment. [Ref. 2 : p. 60]

NTDS became the general term for a number of diverse systems which provided processing display in addition to other functions for shipboard installations. The systems were dissimilar in many features including hardware, capabilities, configurations, and functions. It became necessary to establish an effective classification system in order to refer to a specific NTDS type. The family of CDS (NTDS) Types can be classified by using a logical set of descriptors to compare systems. The descriptors used are system or main computer type, number of main processors, complement of auxilliary processors, complement of data links, and display types.

2. Combat Direction System

The formal definition of CDS is as follows:

> Combat Direction System (CDS) - Those combinations of men and data handling systems, either manual or automated, employed to execute the combat direction functions. They support Command at levels from the platform (ship, aircraft, submarines) up to, and including, task group, force. [Ref. 1: p. A-1]

The CDS is the integrating system between ownship (i.e., the platform that the CDS is on) and other tactical units, in addition to between ownship's sensors and weapon systems. The CDS consists of the hardware, software, and the personnel necessary to execute direction and employment of sensor and threat countering systems and prepare for avoidance of or execution of enemy air, surface, subsurface unit engagements and targets.

The CDS provides ship's personnel with the ability to monitor the overall air, surface and subsurface environment. Force and ownship sensor data is collected, correlated and evaluated by the CDS, then disseminated to ensure the most efficient use of all weapons systems.

Combat Direction Systems have been under development and evolving since 1958. Early systems provided basic ship to ship data link capability, analog to digital conversion of tactical data, and manual, rate-aided tracking. Today's systems consist of upwards of 20 tactical interfaces. These interfaces include multiple tracking sensors, multiple weapons interfaces, electronic warfare and multiple tactical data link systems. Along with the increased capabilities have come increased processing requirements and the need for greater sophistication in tactical display capability. The demands for efficient computation and lucid display in such sophisticated systems have greatly increased the cost of fielding a new CDS.

B. OBJECTIVES

The objective of this thesis is to evaluate commercially available object-oriented database management systems (OODBMS) to determine if any would meet the requirements to form the basis of the Low Cost Combat Direction System (LCCDS) Project.

C. ORGANIZATION

Chapter II is an overview and description of the LCCDS Project and the object-oriented database features which must be addressed. Chapter III surveys object-oriented concepts of database management systems and technology. This chapter concludes with a comparison of conventional and object-oriented database systems. Chapter IV is a survey of three commercially available object-oriented databases evaluated for the LCCDS project. Chapter V addresses the database selection criteria and recommendations for the appropriate database for this project. Conclusions and recommendations are summarized in Chapter VI.

II. LOW COST COMBAT DIRECTION SYSTEM PROJECT

A. BACKGROUND

The primary task of the Low Cost Combat Direction System (LCCDS) Project is to implement the basic features of a Combat Direction System (CDS) on a commercially available microprocessor workstation. Using equipment and technology that is commercially available could significantly lower the cost of fielding and maintaining new systems. The LCCDS would be installed on non-combatant ships or could augment the processing capability on board ships currently equipped with CDS. [Ref. 3]

The LCCDS follows the concept of the Next Generation Computer Resource (NGCR) program, a cooperative effort between the Navy and private industry to produce a set of state of the art computers for shipboard use in the late 1990's. Compatibility among the computers will be through standard protocol rather than through common instruction set architecture or form-fit-function replaceability. [Ref. 3]

Although standards for NGCR computers are not yet complete, the basic features which are envisioned are:

1. 32 Bit ISA
2. 1 - 100 MIPS performance depending on application requirement
3. 26,000 hour Mean Time Between Failure
4. One or more of the industry and government back plane bus standards including VME, IEEE 896 (FUTUREBUS), IEEE 1296 (Multibus II), and VHSIC PI-Bus
5. Three types of local area networks to choose from depending on the application requirement to include SAFENET I (< 10 Mbps), SAFENET II (> 100 Mbps), High Performance LAN (1 Gbps)
6. Network Database Management System [Ref. 3]

Consistent with the NGCR program, experimentation with implementing a CDS on commercial, microprocessor-based workstations may demonstrate a low cost approach to providing state of the art computers for shipboard use in the 1990's. This project will establish the feasibility of this approach by implementing a prototype LCCDS which contains the basic features of the CDS on a commercially available microprocessor workstation.

Ideally, the Naval Sea Systems Command (NAVSEA) prefers that the developed system be portable to several computer suites (systems) that conceivably could include

the Navy embedded computer family AN/UYK-44. Since Ada is proposed as the next generation tactical language for the Navy, an Ada implementation is also desired. [Ref. 3]

Relevant technologies for this effort are determined by some of the basic requirements of the LCCDS, which include:

1. The system must be able to rapidly summarize and present information needed by the operator in an interactive graphic display format of the operator's choice.
2. The system must be flexible to adapt to changing sensors and threats.
3. The system must be portable to allow distribution to many different environments and to enable incorporation of advances in hardware.

These requirements suggest that object-oriented techniques coupled with sets of reusable software components should be utilized. Object-oriented approaches for designing graphic iconic user interfaces have been shown to be effective and make it possible for the operator to build screen formats of their choosing. [Ref. 4: p. 157]

B. PROJECT DESCRIPTION

NAVSEA partitioned the LCCDS Project into five increments which should result in a continuously improving product that reflects the desired features of the CDS community. This thesis addresses an area in the first increment involving selection of an object-oriented database management system (OODBMS) to help form the basis of the project. The features and requirements as specified by NAVSEA for each increment are as follows [Ref. 3].

1. Increment One

During this increment, the LCCDS computer system, run-time operating system, software support environment, and OODBMS will be selected.

The OODBMS may be a commercially available system (with extensions, if necessary) or a newly designed system. The database manager should provide features for data entry, user friendly query language for building logical and arithmetic relationships between database elements, a powerful output generator for developing screen and hard copy formats, and a facility to allow the user to define new objects.

A display graphics capability must be designed or developed which interacts with the database manager and provides the user with the building blocks necessary to define their own customized screen formats for interactive operation of the system.

Display features must include:

1. Windows and pull down menus for operation of all program features including the operation features such as mode selection, range scale, track information, help commands, and display doctrine activation and deactivation.
2. Control of window and pull down menu selection via mouse, trackball, light pen, touch screen or any other feature deemed usable by the developer.
3. Graphics capability to display symbology.
4. Ability to overlay track and ownship position portion of the display with world vector shoreline maps as available from the Defense Mapping Agency.

The database manager must be directly coupled with pull down menu options and window spaces so that the user can define and change all information as required. The display capability should be built generically so that the user can tailor all display presentations including pull down menu options and window spaces effectively building a new run time Man-Machine Interface as desired.

Display doctrine is a feature which enables the user to define the conditions under which data will be displayed and how to display it. Doctrine statements should be IF - THEN - ELSE type statements where the IF clause provides for operations on tactical information such as type, speed, and bearing of tracks to be displayed. The THEN clause provides for the data presentation to blink the symbol, enlarge, bold type, etc. Doctrine should be simple to define and easily activated or deactivated. Doctrine statements should be able to be defined individually and combined, if desired, into a doctrine tree which consists of up to approximately twelve separate statements. The IF clause of a doctrine statement should allow for at least twelve qualifiers.

The general response time to any user selected display option should be no greater than one-half second.

2. Increment Two

During Increment Two, a Manual Tracking and Identification capability must be integrated to the basic display. This will allow the system user to build and display a set of geographically stable and/or moving points of information on the position portion of the display. Manual Tracking and Indentification will include, but is not limited to, the following features:

1. Maintain an ownship symbol in the center of the position display at all times.
2. Introduce a standard display symbol at the current cursor location. All symbols should be maintained relative to the ownship symbol.

3. Allow the user to assign a speed and bearing to a vehicular track. Display a proportioned speed leader on the track and change its position at least once every four seconds. Track position should be dead reckoned using the current track bearing.
4. Allow the user to assign additional information to any type of track.
5. Allow a currently displayed track to be selected and show designated additional information in a window. The additional information should be an object from a user-defined window.

3. Increment Three

Increment Three will integrate receive-only Link 11 capability to provide for the receipt and display of track information from the Ship to Ship digital data link. This will entail development of an interface to a Link 11 system via NTDS protocol, parsing link messages and displaying parsed track information on the position display. LCCDS is not expected to package and ship locally generated tracks for output on the data link as it is intended to be a receive-only system. For design purposes, twelve message types with six variations on each type are specified. As NTDS boards are commercially available, there will be no hardware development effort required.

4. Increment Four

This increment will integrate an ownship maneuvering capability which includes navigation capability and track interface information. Ownship maneuvering capabilities include:

1. Allow for specification of up to six steaming routes with up to 50 waypoints (i.e., destinations) per route.
2. Provide Closest Point of Approach geometry from ownship to any track and between any two tracks. Display bearing lines on the position display.

5. Increment Five

Increment Five will integrate an organic auto tracking capability using an as yet to be determined radar interface. This entails the development of an interface to a ship mounted radar, parsing the input information and displaying the radar contact information of the position display.

C. INCREMENT ONE - SPECIFIC TASKS

The first increment will involve selecting a suitable computer system, run-time operating system and software support environment. During this first period, the following tasks must be accomplished:

1. Evaluate available object-oriented database managers and choose the best available system to act as a basis for the rest of the project.

2. Identify and design any necessary extensions or enhancements.
3. Evaluate available systems for portable graphics support.
4. Design and develop a portable graphics facility supporting customized screen formats that is compatible with the available graphics and object management systems.
5. Demonstrate the capability to display the standard alerts and symbols used in a PPI console.
6. Demonstrate the capability to display track and ownship position on world vector shoreline maps.

This thesis will address object-oriented technology, evaluate commercially available systems and make recommendations as to which OODBMS should be selected as a basis for this project.

III. OBJECT-ORIENTED CONCEPTS

A. SURVEY OF OBJECT-ORIENTED CONCEPTS

Encapsulation has traditionally been an important concept due to the necessity of decomposing large systems into smaller encapsulated subsystems in order to facilitate development, maintenance and portability. Object-oriented systems formalize encapsulation and advocate the use of objects instead of programs and data. [Ref. 5: p.3]

An object is a description of an entity in an application domain. Each object has a unique system identifier along with a set of operations defined for that object. A class is a special object which describes similar objects by defining the property names (i.e., variables or slots) and operations (methods) of the objects it represents. A message is sent to an object to tell it to execute one of its operations. The concept of inheritance of properties via a hierarchy is characteristic of object-oriented systems.

1. Reusability

a. Instantiation

Objects can be instantiated statically or dynamically. Statically instantiated objects are allocated at compile time and exist for the duration that the program executes. Dynamically instantiated objects require run time support for allocation and for explicit deallocation or garbage collection.

Programmers may define their own classes so that they can instantiate and define their own objects. The class definition specifies the variables which make up the object and the methods which represent what it knows how to do. Depending on the language used, these variables and methods may be public (i.e., visible to the end-user), private (i.e., visible to the object only), or subtype-visible (i.e., visible to the object and its subclasses). Some languages support only public visibility, some support public and private visibility, while others support all three levels.

b. Inheritance

Inheritance is a reusability mechanism which allows behavior sharing between classes. It permits a class definition to be reused in its subclasses. This allows for the building of a new version of a program without affecting the old version. Classes naturally lend themselves to bottom-up design for reuse, extension and combination

[Ref. 6: p. 325]. An extension to simple (single) inheritance is multiple inheritance wherein a subclass may inherit from several parent classes.

Inheritence is affected by the following properties:

1. Does inheritance occur statically or dynamically?
2. What are the clients of inherited properties (i.e., classes or instances of classes)?
3. What properties can be inherited?
4. Which of the inherited properties are visible to the client?
5. Can the inherited properties be overridden or suppressed?
6. How does the system resolve name conflicts? [Ref. 5: p. 6]

c. Polymorphism

Polymorphism is a feature which permits the definition of flexible software elements responsive to extension and reuse. A polymorphic operation has multiple meanings depending on the type of its arguments.

Class inheritance is related to polymorphism in that operations that pertain to instances of the parent class also pertain to instances of its subclasses.

d. Genericity

Genericity is the ability to define parameterized modules. Genericity allows module implementors to write the same module code to describe all instances of the same implementation of a data structure, applied to various types of objects. The generic module is a module pattern and is not directly usable. The generic parameters stand for types. Instances of the generic module are obtained by providing real types for each of the generic parameters. Unconstrained genericity has no specific requirement imposed on the generic parameters. It is a technique used to bypass some of the unnecessary requirements of static type checking. Generic definitions which only make sense if the actual generic parameters satisfy a certain structure are of the constrained genericity form.

2. Typing

Each entity in a typed language is declared as being of a particular type. From this, it is possible to determine whether any operation is typewise correct directly from the program text at compile time (static typing) or by checking it at execution time (dynamic typing). Typed objects may be statically checked to ensure that objects do not have to protect themselves from unexpected messages.

3. Concurrency

Object-oriented design fosters decentralized software architectures. This decentralization also requires that control is decentralized and parallelism is backed. Communicating entities must work in parallel, each providing data when and where needed.

The two methods which programming languages use to deal with concurrency and communication are:

1. Active entities or processes communicate indirectly through shared passive objects. There must be a mechanism in place so that the active entities can synchronize their accesses to the shared objects.
2. Active entities communicate directly with one another by message passing. Explicit synchronization is not required because message passing packages communication and synchronization.

B. PROPERTIES OF OBJECT-ORIENTED SYSTEMS

1. Object Management

Object-oriented systems should provide the user with run-time support for objects. Typical support includes:

1. Persistence of objects between sessions via persistent virtual memories or object-oriented databases.
2. Garbage collection features which provide for automatic deletion of unreferenced objects.
3. Concurrency control and communications through message passing or object-oriented databases.
4. Distribution which provides for global object naming and remote message passing or method invocation.
5. Security which protects the ownership of objects through granting of access rights.

2. Object-Oriented Programming Environments

The programming environment must have mechanisms and paradigms for reusability as well as tools which assist with object design, object selection and reuse, and management of the evolving database.

C. CONVENTIONAL DATABASE MANAGEMENT SYSTEMS

There are a number of existing database technologies that could be included in this project.

1. Characteristics of Conventional DBMS

Conventional database management systems based on relational, hierarchical, or network data models are characterized by a fixed set of structures and operations.

An application program is written to interpret the data and implement structures an operations which are not in the fixed set.

The conventional database systems provide for a clear separation between actual data and the stored definition of the database (metadata). The schema is relatively static. There are usually few types with many instances of each type. There are usually short transactions and ad hoc query capabilities. Entity types are simple and there are a small number of simple constructs. The conventional DBMS is computationally weak with only passive data. Recovery and security are very important aspects of the conventional DBMS.

The conventional DBMS contributed the following to the object- oriented approach: persistency; abstractions which allow device, physical and logical independency; views; concurrency; authorization; recovery; efficient storage and retrieval of a large amount of regular simple data; and query languages.

a. The Relational Data Model

The only structure in this table-oriented model is the relation which consists of a set of homogenous tuples. These tuples describe entity instances, each of which is meant to be identifiable. The database schema is an unstructured set of relations.

The designer is restricted to a fixed set of predefined types. Relational models also provide redundancy of data. Each different view of the data requires a separate physical structure.

b. The Hierarchical Model

This is a powerful model which lacks abstraction capabilities. This model is difficult to use if the schema is large and complex. Once the schemas are in place they are fairly static.

Undesirable characteristics include redundancy, insertion and deletion anomalies, asymmetry of logically symmetric queries, range of information bearing constructs, fairly static schema, limited view capability, complexity and not enough powerful abstraction mechanisms.

c. The Network Model

This model is conceptually difficult to understand. It is noted for its lack of simplicity. Schema modifications are complex and difficult. Data in this model is not logically independent.

2. DBMS Facilities

A transaction is a unit of work and is also the unit of recovery and consistency. Internally the unit of work consists of several database operations that are logically indivisible.

Recovery is concerned with recovery from system failure which may be local, system or media failure. The basis of recovery is redundancy by writing old and new values onto a log and periodical archiving.

Security is the protection of the database against unauthorized disclosure, alteration or destruction.

Integrity in the database is concerned with accuracy, correctness and validity. Integrity is protection against invalid operations and consistency is the consequence of achieving integrity. Current conventional DBMS are very weak in this respect.

Concurrency occurs when multiple users use the same database simultaneously. Solutions to this problem include locking and timestamping mechanisms. Locking, however, may lead to deadlock. Relations, tuples and fields may be locked.

D. OBJECT-ORIENTED DATABASE MANAGEMENT SYSTEMS

1. An Overview

Object-oriented data modelling considers the database as a collection of objects where each object represents a concept, idea, event, or physical entity of the database application. Real world objects are represented directly by database objects rather than as a collection of record types stored in a file. The goal of the object-oriented approach is to maintain the direct correspondence between real world objects and their database representation. In this manner objects do not lose their integrity or identity.

Object-oriented database management systems (OODBMS) represent the next step in the evolution of database management systems. These systems are targeted to meet the data management requirements of complex data and process-intensive applications [Ref. 7]. Examples of complex data and process-intensive applications include computer-aided design (CAD), computer-aided software engineering (CASE), computer-integrated manufacturing (CIM), and computer-aided rapid prototyping. These applications have large, complex, and highly interrelated data objects.

OODBMS strive to provide facilities to define any structure and operation which can then capture the unlimited amount of semantics, simplify application development, and isolate applications from changes to the database.

OODBMS provide an extensible set of data structures, operations and abstractions. Descriptions of the entities may be type or instance data. The entities are composite and have different aspects. The nature of the process is tentative and iterative as their are refinements, alternatives, versions and releases. Reuse is a fundamental concern at the organization, project and group level. The operations may be long and complex transactions, interactive, concurrent, not very ad hoc and generally localized. Graphics may play an important role, too.

Any database management system, including an OODBMS, must provide the capabilities of persistence, concurrency, authorization and security, transaction management, and recovery. Additionally, the OODBMS is an object-oriented system and, as such, it must provide the capabilities of objects, composition, classification, active and passive data, generalization, extensibility, and encapsulation.

OODBMS can provide application-oriented capabilities such as the following:

1. Version and configuration control for CAD applications.
2. Dynamic creation of classes, promotion of an instance to class and demotion of a class to instance for artificial intelligence (AI) and expert systems.
3. Recursive classes, multiple inheritance, extensive tool interface capabilities for CASE.
4. Support for multimedia object, distributed environments, and graphics for CIM. [Ref. 7: p. 60]

The development of the database schema has the following goals:

1. Identify and define the objects.
2. Identify and define the class hierarchy among the objects.
3. Specify the properties associated with each object.
4. Specify the operations performed on the objects.

2. OODBMS Performance

One of the system requirements is speed of execution. The OODBMS could theoretically be faster at fetching and storing fields than a relational system [Ref. 8: p.577]. Maier [Ref. 8] proposes that object-oriented databases can handle individual object accesses quickly enough to keep design data under database control while it is being manipulated by tools. The reasons for this are:

1. By having methods in the database, one message sent from the application program can complete multiple field accesses in the database at a cost of one procedure call.
2. Objects can refer to subcomponents by identity instead of state or key values and thereby remove one level of mapping.

3. Complex design entities can be represented directly with less encoding and fewer levels of mapping to access one conceptual entity.
4. Long database transactions require that users know of one another and that the database support multiple versions of objects.
5. Query responses can be constructed by collecting references to objects in the database instead of copying the objects.

IV. COMMERCIALLY AVAILABLE OBJECT-ORIENTED DATABASES

A. GEMSTONE

The GemStone object-oriented database management system [Refs 9, 10, 11, 12, 13, 14, 15] is a trademark of Servio Logic Corporation of Alameda, California. The company advertises GemStone as:

> GemStone is a new kind of multiuser data management system that combines the security and integrity of mainframe DBMS, the flexibility of a file system, and the power of object-oriented programming. This unique combination of characteristics makes GemStone an ideal foundation on which to build data management applications in such areas as CAD, office automation, and computer-integrated manufacturing. [Ref. 9: p. 2]

1. System Overview

GemStone incorporates object-oriented language concepts with the capabilities of database management systems. It was designed for multi-user applications where concurrency and a high level of data protection are required. GemStone supports the notion of objects, object inheritance, encapsulation, and classification. The system follows the philosophy of sending messages to objects as in the object/message paradigm of the SMALLTALK-80 language. GemStone classes are organized in a class hierarchy with each subclass inheriting the behavior and structure of its superclass. The user may define new classes. The methods are defined as part of the classes. Both pessimistic and optimistic concurrency control of transactions are supported to ensure data integrity and persistence. Mechanisms for authorization and recovery are also provided.

The general purpose programming language for GemStone is the interactive data definition and manipulation language OPAL. This powerful language provides structures which are persistent and extensible. OPAL supports instance variable typing and paths to facilitate associative access of procedures with data structures.

2. System Architecture

The architecture of GemStone consists of a host computer running a single database monitor, Gemstone sessions, and application interfaces (see Figure 1 on page 18). The database monitor distributes object-oriented processes and disk pages and functions as the critical region monitor for commits and aborts. There is one GemStone session for each client application with each session providing full functionality of the GemStone system. Each GemStone session incorporates a data management kernel and

an interpreter for OPAL. Direct application interfaces are currently available for C, C++. LISP, Parc Place Smalltalk-80, and Smalltalk/V. GemStone also provides the capability to access and use data from SQL relational databases.

For application development work, communication with a GemStone session is through a set of special-purpose GemStone interface programs called the Opal Programming Environment (OPE).. OPE consists of an editor, a browser, and a bulk loader/dumper for GemStone databases to help the user write and debug OPAL database applications.

The workstations supported by GemStone currently includes IBM PCs and compatibles, Apple MacIntosh IIs, SUN3s, SUN4s, Tektronix 4300 and 4400 series workstations. The host server environments currently supported by GemStone includes VAX VMS, SUN3 and SUN4. When the SUN3 and SUN4 are used as workstations, GemStone sessions may be run on either the host computer or the client workstation. Workstation programs can communicate with host- resident software via a DEC net with a RS232 serial link or Ethernet link.

3. OPAL Programming Language

The OPAL programming language is used for data definition, data manipulation and general computation. OPAL syntax consists of symbols; temporary variables; and unary, binary, keyword, and mixed expressions. All operations are associative. The execution of any message returns an object.

Blocks are a sequence of instructions which can be executed by sending a message to the object. The messages can have arguments. The control structures in this language are constructed using blocks. Blocks include the IF-THEN-ELSE statement, the While loop, and the Repeat Until loop.

Each class has a unique superclass with the exception of the root (i.e, multiple inheritance is not supported). Class Object is the root of the hierarchy. The OPAL predefined class hierarchy is shown in Figure 2 on page 19 . A class and its subclasses form a tree hierarchy.

Methods may be defined by using the browser or by using the FILE IN command. The browser presents a form which the user fills in and then the OPE builds the method into the class. To use the FILE IN command, the user must enter the text of the method through the OPE workspace, add the FILE IN commands, and select the text and then execute the FILE IT IN command in the workspace execute menu. The OPE would then build these methods into the class.

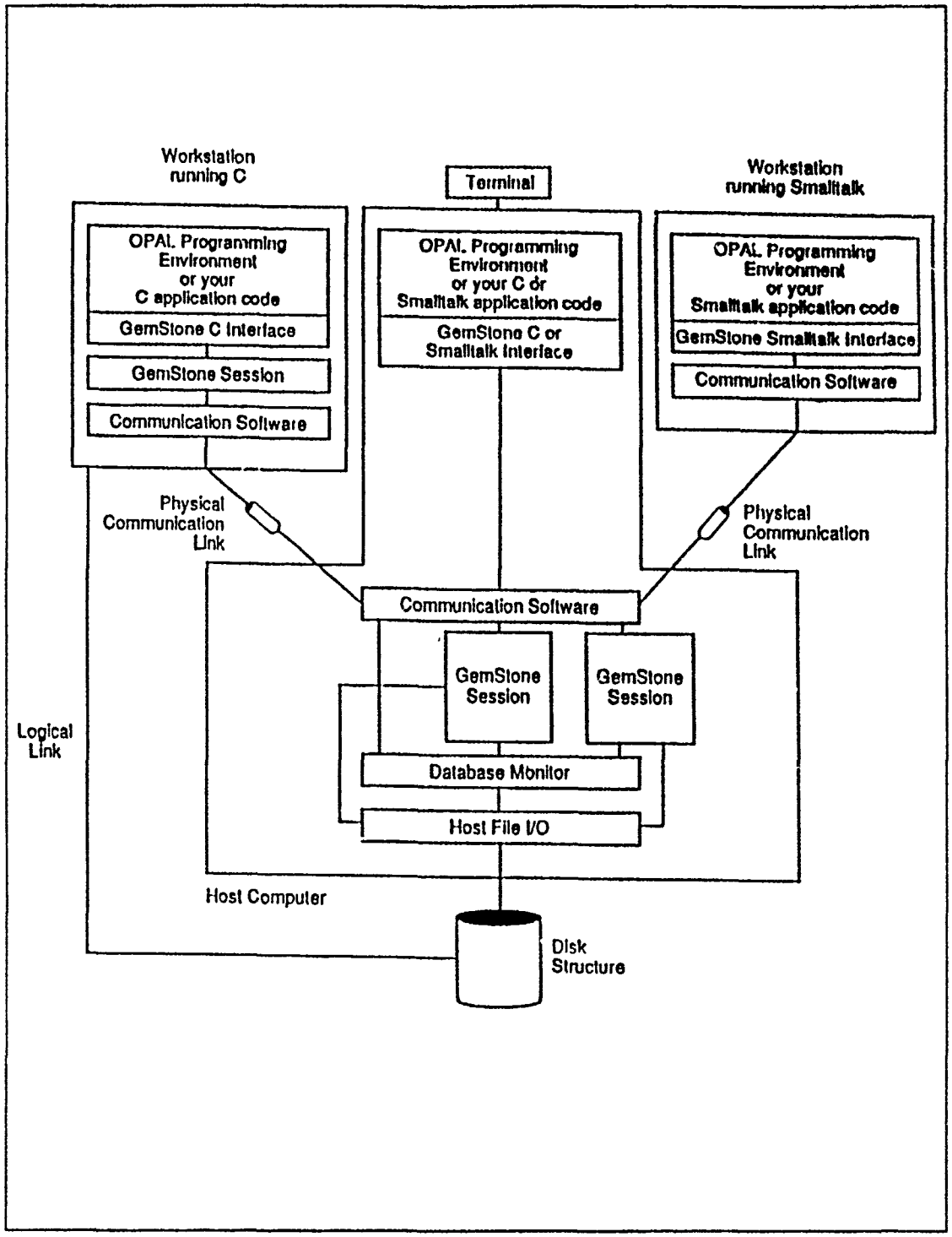


Figure 1. GemStone System Architecture: [Ref. 12: p. 296]

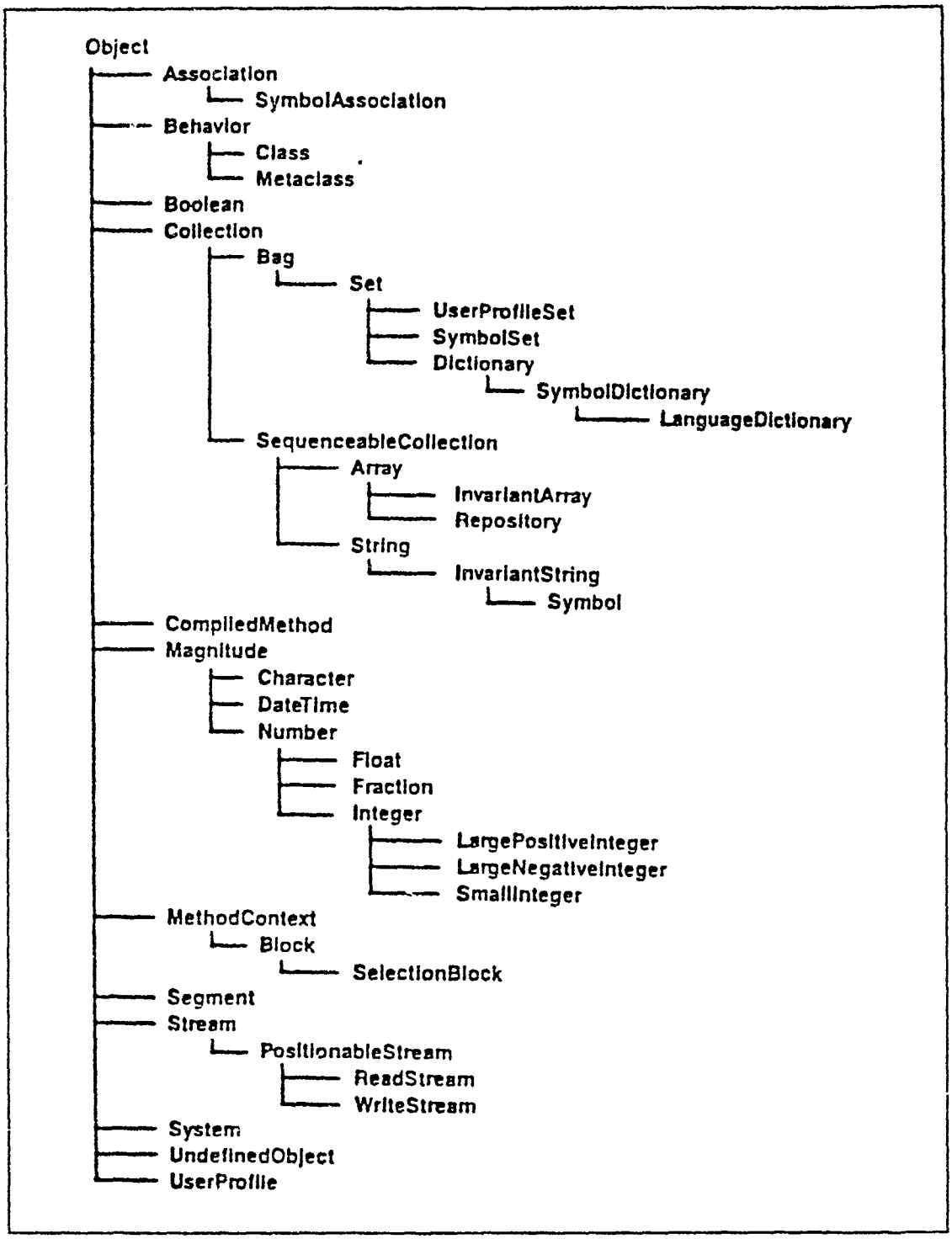


Figure 2. OPAL Class Hierarchy: [Ref. 9: p. 9]

A class may have instance variables which store a property of the instance of that class. This constrains that instance variable to variables of that type. Instance variables may be set and accessed but they need not be named. An anonymous instance variable can be referenced by index or value. The class may also have instance variables which are shared among all instances of the class (these are more commonly referred to as class variables). A given instance variable may have its constraint changed. When a new constraint is a subclass of the old constraint the constraint is termed specialized. If the new constraint is a superclass of the old constraint it is termed generalized. If the instance variable is inherited, then the constraint may not be generalized. This primitive preserves the class-kind invariant.

In addition to temporary, instance and class variables, classes may also have pool variables which are defined in dictionaries. The dictionaries are pairs of variable names and values in which any number of classes may share a group of the pool dictionaries.

OPAL variable scoping rules are as follows:

1. The scope of block variables is only to the blocks in which they are declared.
2. The scope of temporary variables is only to the methods in which they are declared.
3. The scope of instance variables is to the instance methods of the class.
4. The scope of class variables is to the instance and class methods of the class.
5. The scope of pool variables is to the instances of the classes which import them.
6. The scope of global variables is to the symbols placed in the symbol list dictionaries.

Class modifications which can be made to OPAL include adding or removing class variables, pool variables, and methods. All other features including class name, storage formats, instance variable names and superclass are fixed. Changes to a class do not affect the existing instances of a class.

Collection classes are objects which store groups of other objects in a hierarchy (see Figure 3 on page 21). Component objects of a collection can be accessed by their ordering or accessed associatively. Dictionaries in OPAL are collections of pairs of key and value, where each key is unique.

a. Path Expressions and Typing

The path expression in OPAL is a sequence of one or more instance variables separated by periods which returns the value of the last instance variable in the sequence. A path expression is used in selection blocks to form a query. Accessing

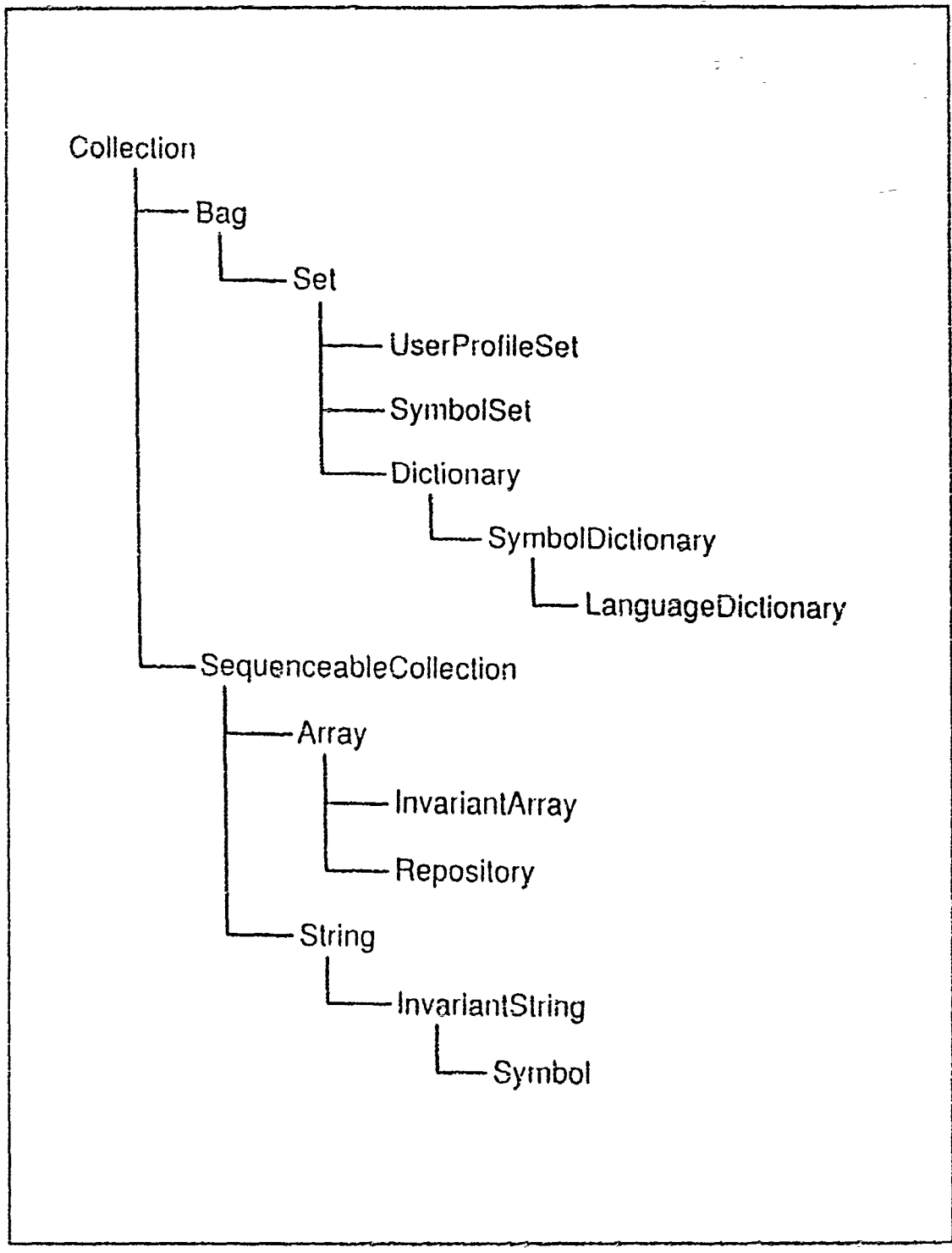


Figure 3. Collection Class Hierarchy: [Ref. 9: p.10]

objects through indexing and associative access may be accomplished by query, constraint, building indexes, path expressions, and selection blocks.

b. Associative Access and Indexing

In OPAL, only the nonsequenceable collection supports indexes when proper typing exists for the path being indexed. Indexes on paths are implemented by a sequence of index components with one for each link in the path suffix. The data structures used for implementing indexes are stored in object space and managed by GemStone.

The constraint limits the type of objects that an instance variable may take on as a value. The instance variables of a class are constrained when the class is defined. Once an instance variable of a class is constrained it cannot be changed. The constraints are inherited and can be restricted in the subclass. Currently, however, an instance variable of a class C cannot be constrained to that class C or any subclass of C.

After the instance variable is constrained, indexes may be built on it. Two types of indexes can be built on objects. An identity index is built automatically on the elements of constrained collection classes. Equality indexes can be built on instance variables of type boolean, character, number and string. An equality index may be built on the elements of a collection or on an instance variable.

c. Query Language

Associative access was designed with a limited calculus sublanguage in which associative queries are viewed as procedural OPAL code. This results in little impedance mismatch between OPAL and its query sublanguage.

Execution of a query by pure message passing can be very expensive if there are thousands of messages that are each sent thousands of times. The solution to this problem is to build indexes in order to avoid message passing overhead.

The selection block appears as a single argument block. Comparison operations in blocks are not messages. Query protocol allows for select, reject, or delete operations.

4. GemStone Database Features

a. Name Spaces and Workspaces

GemStone supports multiple name spaces through instances of the class UserProfile. Instances represent the properties of each user and include a list of dictionaries which resolve symbols when the user compiles OPAL code.

Multiple concurrent users are supported in GemStone by providing each user session a workspace. A shadow copy of an object is created whenever a session

modifies the object. The shadow copy is inaccessible to other sessions until the transaction is committed. Shadowing allows access of a consistent version of the database by different transactions.

b. Transactions and Recovery

A shadow copy of an object is created whenever a session modifies that object. The shadow copy is inaccessible to other sessions until the transaction is committed. Aborting a transaction will throw away objects shawdowed in the workspace. Committing a transaction will replace shared objects with their shadow. The GemStone unit of recovery is the transaction. Changes to the database which have been committed are kept; those changes not committed are lost.

In order to protect the database against media crashes GemStone supports replicates in a repository. Repository is an OPAL class which provides the internal representation for repositories. The Repository instance responds to the message replicate. The Repository may be instructed to replicate itself or the replicate may be disposed of.

c. Concurrency Control

GemStone provides optomistic and pessimistic concurrency control. A session operates optimistically on objects when there is no danger of conflict and thereby avoids incurring the additional overhead required for locking. A session operates pessimistically on objects which most likely will produce conflicts. A transaction can access some objects pessimistically and, at the same time, access others optimistically.

Optomistic concurrency control is transparent to programs with no conflict and it favors read over write transactions. This scheme guarantees that read-only transactions never conflict with other transactions and never deadlocks. GemStone maintains a read list and a write list in order to discover potential conflicts. If there are no conflicts with transactions that committed since the transaction began, the transaction commits by merging its changes with the other transactions.

Pessimistic concurrency control supports long transactions and is effective in class modifications. By locking an object the system can ensure that read-write conflicts and write-write conflicts do not occur. Transactions which hold a read or write lock on an object are guaranteed that the object cannot be written by another transaction. Modifying a class in a shared environment may cause violation of the representaiton invariant. If a user creates an instance and commits the transaction before another user completes modification of a class, the transaction performing the schema modification will be unaware of the new instance. A class must be exclusive-

locked before it can be modified in order to prevent creation of instances of the class and sending of messages to the instances.

Segments are also units of granularity in concurrency control. Reading or writing an object puts the segment containing the object into the read/write list. Objects which are updated frequently are placed in different segments.

d. Authorization and Security

The main mechanism to implement authorization is segments. A segment groups objects and is owned by a single user who has read and write authority over the segment. Access rights to an object can be changed by moving it from one segment to another. Permission on an object does not imply permission on all its subobjects.

e. Large Object Space

The GemStone design sought to set limits on object numbers and sizes such that physical storage limits would be met first. When an object is larger than a physical page, it is broken up and organized into a tree structure which can span pages.

The five basic storage formats for objects are self-identifying, byte, named, indexed, and nonsequenceable collections. The byte format is for classes whose instances are unstructured. The named format provides access to the components of an object by unique identifiers such as instance variable names. The indexed format provides access to the components of a class by number and supports insertion of components into an object and growth to accomodate more components. The nonsequenceable collection format is used for the collection classes in which the instance variables are anonymous.

The basic data structuring mechanisms that GemStone provides are sequencing for records, iteration for arrays, and collections for sets.

Performance can be significantly improved by clustering objects, that is, storing together those groups of objects that are most often accessed together or clustering. Only objects from the same segment can be clustered into a single page. Clustering is not maintained automatically but must be specified by the database administrator or the application programmer. After updates, the objects in the page must be reclustered.

B. IRIS

The Iris object-oriented database management system [Refs. 16 , 17, 18, 19] is currently under development at Hewlett-Packard Laboratories, and is described as follows:

Iris is intended to meet the needs of new and emerging database applications such as office information and knowledge-based systems, engineering test and measurement, and hardware and software design. [Ref. 16: p. 219]

1. System Overview

Iris is a prototype research system being developed at Hewlett-Packard Laboratories. The capabilities that are being developed include: rich data modeling constructs, direct database support for inference, novel data types (graphic images, voice, text, vectors, matrices), lengthy interactions with the database spanning minutes to many days, and multiple versions of data. Data sharing must be provided at the object level in the sense of both concurrent and serial sharing, allowing a given object to be accessed by applications that may be written in different object-oriented programming lanuages. The Iris DBMS is being designed to meet these needs. [Ref. 16: p.220]

The relational data model is the underlying model of computation and implementation of Iris. The functional data model forms the foundation of the logical model for Iris with enhancements such as generalization, inheritance and update capabilities. Functional data models use the concept of a mathematical function as the fundamental modeling construct. The functional data model provides for collections of domains and functions which are advantageous for function composition. The disadvantages of this approach are that there is lack of structure, difficulty of update, and metadata is different than the data.

The enhancements to the functional data model included generalization in order to impose structure on the database. Functors SET, ADD, REMOVE and SELECT were provided for update. Inheritance capabilities provide for a domain of domains for the metadata. The Iris data model provides for objects, types, operations, relationships and attributes, and update operations to the schema and database.

2. System Architecture

The Iris prototype is implemented in C on HP-9000,350 Unix workstations. It was designed with a server-workstation configuration to run in a Unix environment. It has a layered architecture consisting of an Object Manager, Storage Manager, and interfaces as shown in Figure 4 on page 27.

The Object Manager implements the Iris data model by providing support for schema definition, data manipulation, and query processing. Classification is by type and instance. Generalization is by type or subtype. Iris functions may be implemented as stored functions, foreign functions, rules or aggregate functions.

The Object Manager interface is a set of C routines. The Iris interfaces are built on top of these routines. Current interfaces implemented for Iris are Object SQL (OSQL), the Graphical Editor, OSQL embedded in LISP, and a PCLOS interface.

The Storage Manager is a conventional relational storage subsystem enhanced by parent-child links which support both a relational and a network query processor. It provides for creation and deletion of relations, transactions management, concurrency control, logging and recovery, archiving, indexing, buffer management, and tuple-at-a-time processing for retrieve, update, insert, and delete operations.

3. Iris Object Manager

a. Objects

Objects are entities and concepts from the applications domain. Each object has a unique identifier which supports referential integrity. Objects are described by their behavior and accessed by a set of functions and manipulated by predefined operations.

Objects may be classified by type. Those belonging to the same type share common functions. An object can have multiple types because the types are arranged in a type hierarchy with inherited functions. Objects may also be classified as literal or non-literal. Literal objects represent themselves so they cannot be created, destroyed or updated by users. Non-literal objects are represented internally in the database by an object identification.

b. Types and Type Hierarchies

Types are named collections of objects which provide functions for their instances. Functions are computations defined on types and are organized in a hierarchy of types and subtypes. The Iris type structure is a directed acyclic graph. The subtypes inherit all the operations of the supertype. Iris supports multiple inheritance in that a type may have multiple supertypes. Types can be overlapping or disjoint (disjoint types do not have a common subtype). Functions names may be overloaded. However, when an overloaded function is applied to a given object, a single specific function must be selected at the time of application.

The type Object is the supertype of all other types so it contains all other objects. Types are objects in themselves. Any function is an instance of the function and any type is an instance of the types. Functions are also considered to be instances themselves.

The Object Manager allows new types to be created, old types to be deleted, and objects to gain or lose types in order to support database evolution. It is not cur-

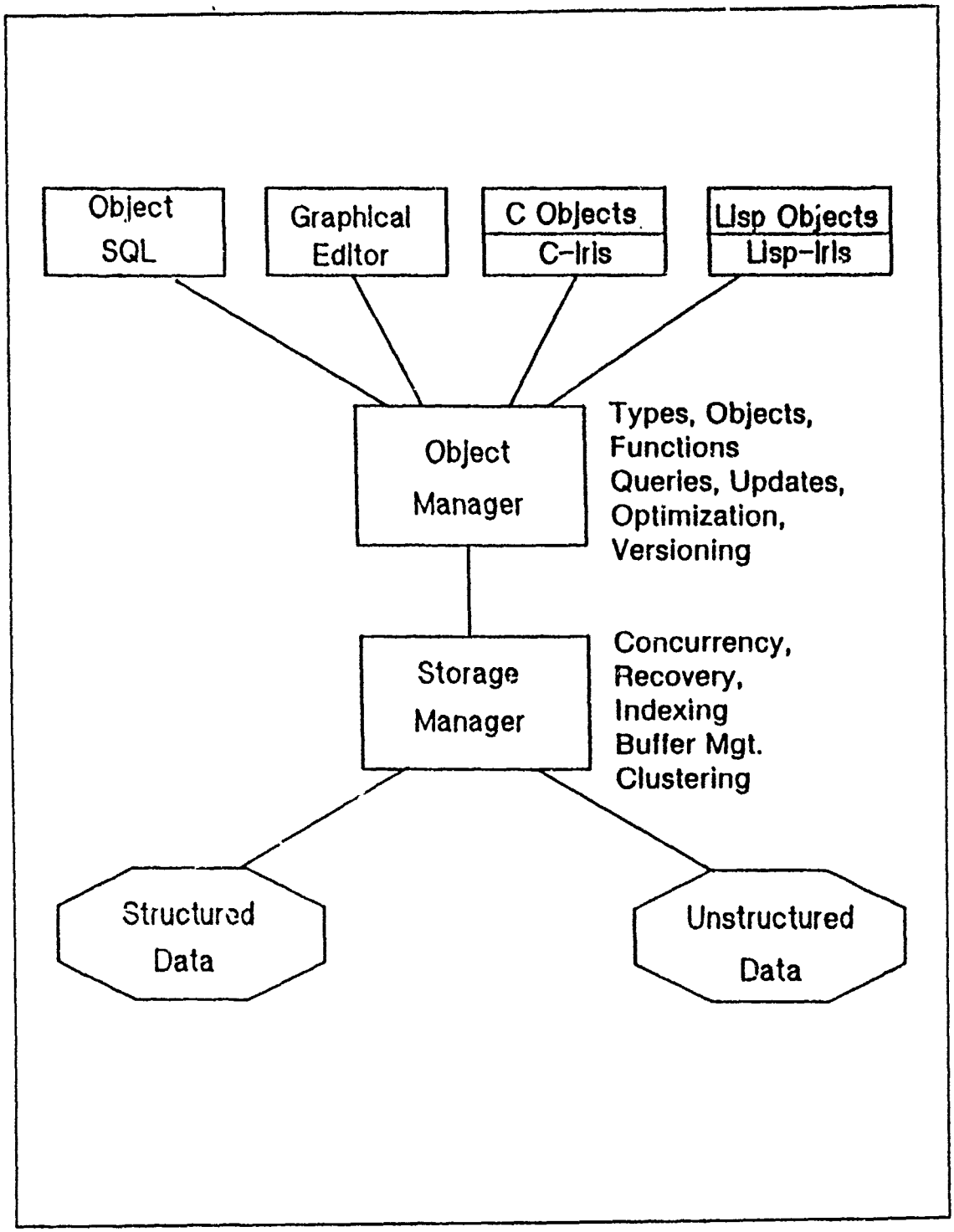


Figure 4. Iris System Architecture: [Ref. 16: p.220]

rently possible, however, to create new subtype/supertype relationships among existing types.

c. Functions and Rules

An Iris function is a computation that may or may not return a result. Functions consist of a function declaration and function implementation which are separated to support data abstraction.

The function declaration specifies the function name and the number and types of its parameters and results. Function declarations are used to define participation constraints in order to avoid meaningless interpretations.

The function implementation specifies how the results of the function are computed. Functions in the Iris system may be implemented as stored functions, derived functions, foreign functions, rules, or aggregate functions.

A function may be implemented by storing it as a table which maps input values to their corresponding result values. The table can then be accessed by standard relational database techniques. A stored function provides the basis for the implementation of a function and its inverses.

Implementation of a derived function is specified in terms of other functions. The definition is compiled and stored as an internal relational algebra expression and the definition of the invoked functions are combined as appropriate.

The foreign functions implementation are functions written in C or other programming languages. These functions are written in a language that Iris does not understand and cannot optimize the implementation. Iris can, however, optimize the usage of a foreign function.

Functions may also be implemented by rules. Rules are modeled as functions. Iris makes the closed world assumption, i.e., that any fact not deducible from the database data is assumed to be false. The Iris prototype supports conjunctive, disjunctive, and nonrecursive rules.

Functions are necessary over bags or multi-sets of objects. However, bags are not Iris objects and may not be stored directly in the database. Aggregate functions are implemented as foreign functions with bag operands. This implementation simplifies query translation and execution by reducing the number of possible operators.

d. Update Operations

Update operations in IRIS change the future behavior of a stored or derived Iris function. Values can be added to or removed from single and multi-valued functions. The delete operation can delete any user-defined object, type or function. Delete

is propagated to all related information; however, instances of the type are not deleted but lose their type.

Procedures are a group of update operations collected into a single parameterized function. In the current implementation of Iris, update procedures do not have a return value.

e. Version Control

A version control mechanism has been implemented in the Iris Object Manager. An object retains its identity throughout its existence even though its state may change. Versions capture the object in certain states and are modeled by distinct objects Separate objects correspond to each version and to the entity of which they are versions.

Version control in Iris provides a means of indirect addressing in that it allows objects to make generic references to other objects. Iris is also capable of creating versioned objects from unversioned objects. Versions of objects must be created explicitly by the user. Operations on versioned objects are further constrained. Controlled sharing among users is possible through the use of version locks which the user sets.

f. Query Processing

Iris queries are declared in terms of functions and objects. The Storage Manager uses relational algebra and tables. Query processing in Iris handled by the Query Translator and Query Interpreter modules. The Query Translator converts the queries from object to relational algebra representation. The Query Interpreter then evaluates the converted query by invoking the Storage Manager to access the database and foreign functions to access other data sources.

4. Iris Interfaces

The Iris database can be accessed via interactive or programming language interfaces. The interfaces are implemented using a library of C subroutines that define the Iris Object Manager interface.

a. Object SQL

OSQL has three main extensions over SQL so that it may adapt to the object and function model. One is through the use of direct references to objects through their keys. The second is that user defined and Iris system functions may appear in WHERE and SELECT clauses to give concise and powerful retrieval. The third extension is that users manipulate types and functions rather than tables.

b. Iris Graphical Editor

The Iris Graphical Editor is the graphical interface which allows a user to browse and update an Iris database. Type and function creation and deletion schema, plus updates and session control operations such as commit and rollback, are supported in the editor.

c. Iris Programming Language Interfaces

Interfaces which have been implemented in Iris are an embedded OSQL interface and loosely and tightly integrated object-oriented interfaces. OSQL was implemented as a stand-alone interactive interface and as a language extension. OSQL is currently embedded in COMMON LISP via a macro extension. A loosely coupled interface to Iris has been implemented in LISP in order to define a layer of abstraction which allows the programmer to access the Iris data model. A tightly coupled interface to Iris has been implemented with Persistent-CLOS (PCLOS) in order to support persistent object extensions.

3. **Iris Storage Manager**

The Iris Storage Manager is a conventional relational storage subsystem manager. It supports creation and deletion of relations, transactions management, concurrency control, logging and recovery, archiving, indexing, buffer management, and tuple-at-a-time processing for retrieve, update, insert, and delete operations.

C. VBASE

The Vbase object-oriented database manager [Refs. 20, 21, 22, 23, 24] is a product of Ontologic Corporation. The company advertises Vbase as follows.

> Vbase is an object database system, providing an integrated software development platform which combines the latest advances in compiler design and database management techniques. [Ref. 24: p. 1]

> The Vbase Integrated Object System is a database and language platform for rapidly and inexpensively building sophisticated commercial and engineering applications. [Ref. 24: p. 5]

1. System Overview

Vbase is a high performance open and extensible, multi-user object-oriented database. The distinguishing features of this object-oriented database management system are that it is strongly typed and compiled, it follows a layered approach to system architecture, and it supports type subtype hierarchy and inheritance. The system follows the philosophy of invoking operations of objects instead of passing messages to objects.

Vbase supports the functionality of the relational database model. Existing file structures can be integrated with Vbase by creating file objects. In addition to retaining desirable object-oriented characteristics, Vbase also provides traditional DBMS characteristics such as sharing of object data among multiple processes and users, handling large object storage spaces, and providing transaction support by means of concurrency control and recovery.

2. System Architecture

Vbase currently runs on Sun3 workstations under the Sun OS 3.2 Unix operating system or on Digital VAX machines under the VMS operating system.

The Vbase design follows a layered approach of an open modular architecture. This allows layers of the system to be replaced with versions tuned to particular applications. A flexible architecture is accomplished with a clear separation of specification and implementation. The language layer supports the specification of objects in terms of what they are and what they do. The abstraction layer supports modeling features of the object-oriented approach. The representation layer supports the semantics of the mapping to stored objects. The lowest storage layer provides for object persistence essential to data management. Each layer in the architecture was implemented in Vbase.

The Vbase environment shown in Figure 5 on page 32 consists of the Vbase database, Type Definition Language (TDL), "C" Object Processor (COP), Integrated Toolset (ITS), compilers for TDL and COP, and Object Manager kernel (OM).

The Type Definition Language (TDL) is the Vbase data definition language. TDL is used to define object types in the database, along with object properties, operations, and inheritance. Its role is similar to that of the data definition languages of traditional databases. It creates a typing structure to serve as a data model or conceptual schema for applications.

The "C" Object Processor (COP) is the Vbase data manipulation language. COP is an object-oriented extension of Kernighan and Ritchie Standard C and it provides access to the database via methods. Methods are issued to implement the operations of the object types defined using TDL.

The Integrated Toolset (ITS) is a debugging and application tool environment that runs on top of Unix.

Object SQL is the Vbase query facility which provides the retrieval features of the SQL relational query language with extensions relevant to an object-oriented database system. OSQL is advertised as an ad hoc query tool as well as a report generation tool.

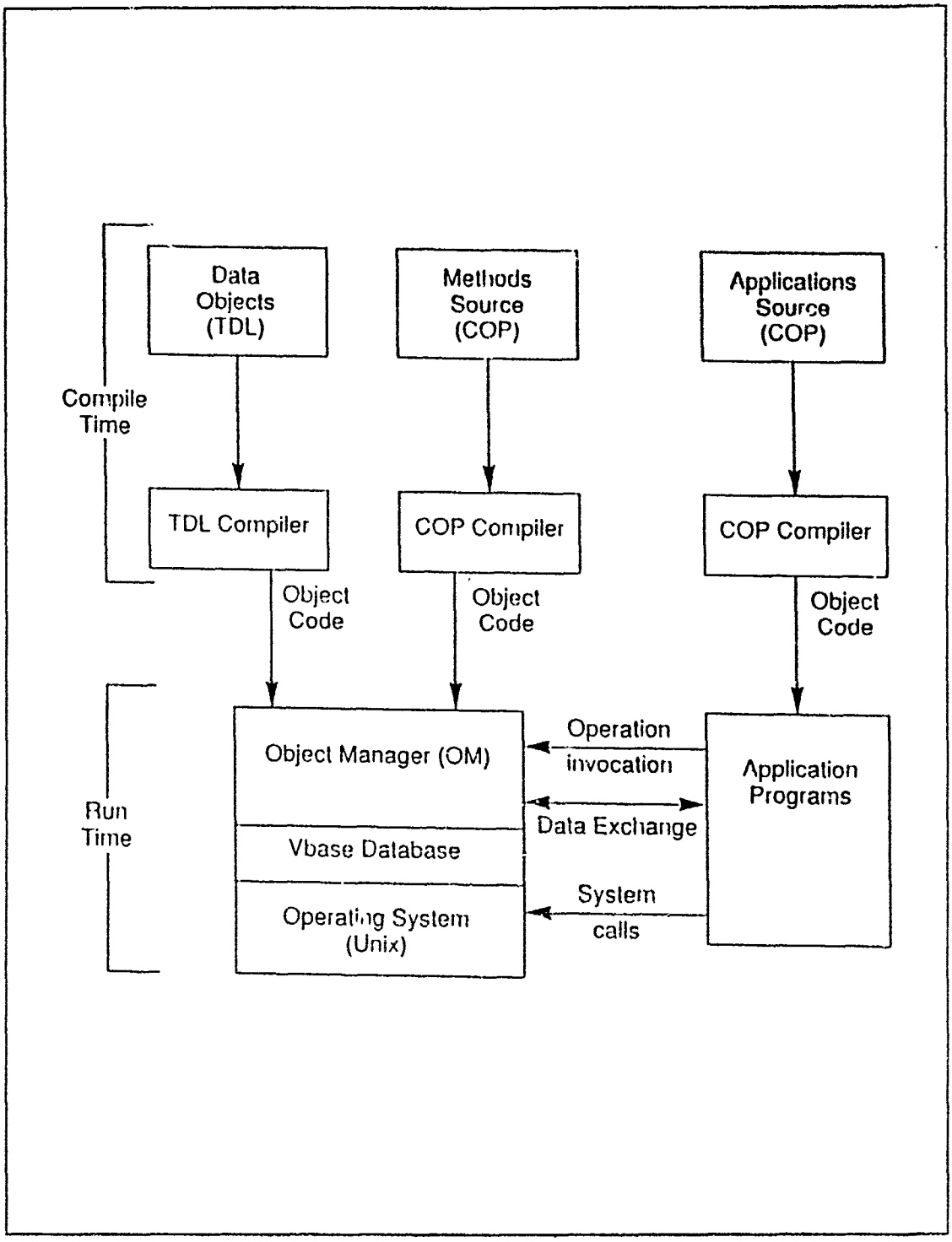


Figure 5. Vbase Compile and Runtime Architecture: [Ref. 24]

3. Vbase Database

Kernel_db is the subdirectory of the vbaseSys system directory which contains the kernel database schema and information used by the object manager. Each Vbase user must maintain a private copy of the database within their personal directory.

The Vbase object database can be either a Unix file or a Unix partition. The storage can be visualized as secondary storage on a disk and a cache area in the virtual address space in the user's program. The database resides as much as possible within the process memory of the application which invokes it. The storage management of the object manager uses the process memory as a cache of the database. The physical unit of transfer between the disk and database cache is called a segment.

Large objects may span segments in memory; however, they are constrained in that they must fit into available main memory. Persistent objects may be clustered in storage by the user if it is likely that the objects will be accessed together.

Delete operations make the targeted object inaccessible. Delete cannot be used on universal objects or when the resources necessary to perform the delete are not available. References to an entity continue to exist after it is deleted. The programmer must unset or reset the references of deleted entities.

4. Type Definition Language

Vbase types are classifications of similar instance objects. Each Vbase object has a type and each type is an instance of type Type. Every object including Type is ultimately an instance of Entity (see Figure 6 on page 34). The type Entity heads the Vbase hierarchy. In Vbase types define properties and operations of their instances. The types are also organized in hierarchies of supertype/subtype. Subtypes inherit all the properties of the supertype. Although Vbase provides for inheritance of types down the hierarchy, multiple inheritance is not currently supported. Complex object types may be created by using the hierarchies. Specification is strictly separated from implementation. Vbase provides a system type library of over sixty-five predefined object types.

Vbase allows objects to be referred to by name. TDL is a block structured statically scoped language. A type definition defines the name scope. Types can be nested inside modules to create name scopes on top of the scopes of the types. There is also a global name context which contains all the names not contained inside a type or module. A reference to a name may be absolute or relative.

Properties of objects are defined in their type and capture static behavior. Properties of a type are similar to attributes of entities in the relational data model.

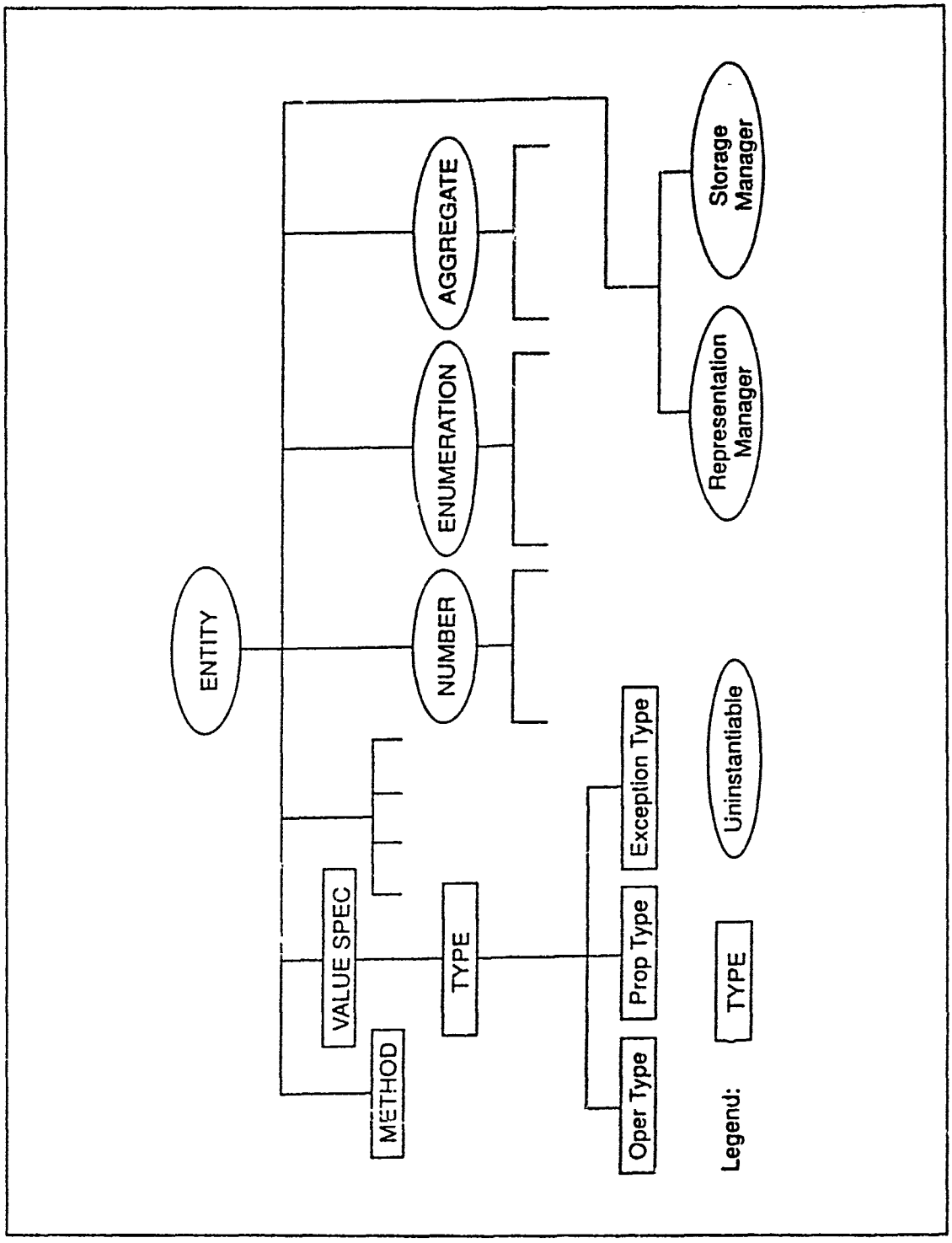


Figure 6. Vbase Type Hierarchy: [Ref. 23: p. 719].

Properties contain state information and model the relationship between objects (association abstraction). The name of the property within a type can be used to relate it to another type as in the functional data model. Object behavior is elicited by a combination of properties and operations. All system defined or user-defined types can be used for value specifications. Inverse relationships between data elements are specified as part of the data definition and maintained automatically at runtime.

The property declaration options available in Vbase are as follows:

1. Aggregates are composed of a group of objects that can be treated as a single object.
2. The union type option allows a property to be one of several types. The types in the union type must have a common supertype. Properties must have values unless they are declared optional.
3. The inverse property option allows the system to keep a property and its inverse consistent.
4. Another option is to use distributed inverses. The only property of type aggregate that can have an inverse is Set. The inverse specification for set valued properties apply to the elements of the set.
5. Allow disallows is the entity type that provides for the operation of Get, Set and Init for all objects. Any subset of these operations can be disallowed in an object.
6. Default values are literals of type string, integer, or real. Default values are defined for optional properties only.
7. Refines indicates that the property is refining the previously defined property.

All interactions with the Object Manager are performed by executing an operation which is specified in TDL and implemented by a method in COP. Dynamic behavior is captured by operations. Operations are implemented by methods. All operations must have at least one dispatch argument of the type which contains the operation. The specific operation invoked is determined by the actual type of the object. An operation can have zero or one result. Arguments of the operation are passed by reference. The operation may have optional keyword arguments with default values. An operation can have only one method but a method can be used by more than one operation.

A subtype can refine the operations of its supertype by modifying the specification and or implementation of the operations. A refining operation must refine the dispatch argument to match the subtype.

Operations may raise exceptions. All the exceptions are subtypes of a system defined exception. When an operation encounters an error condition, transfer of control

is passed to an out-of-line exception handler specified in the user's code. This technique improves error handling, reduces the control overhead required for error conditions and makes robust code easier to write.

Any operation may have a trigger which can enforce constraints, provide rules or automatic side effects. Triggers allow users to add code pieces to the methods. Triggers specify the methods that are invoked before the base method specified in the method clause in order to customize the base method's behavior.

Instance of types can be defined in TDL. Named instances can be accessed individually by name in other TDL definitions and COP programs.

TDL also provides an iterator. An iteration of a loop in conventional programming languages consists of two parts. These are the selection of a data object and processing of the data object. An iterator in TDL abstracts the selection away and allows concentration on the processing. This allows the user to deal with multi-valued objects abstractly without having to know about the underlying data structure. Each time an iterator is called it yields an element.

5. "C" Object Processor (COP)

The COP language is a strict superset of the C language. It is used to build methods that are executed internally within the DBMS and used to build application modules that will access the database. Methods are specified in TDL and then implemented in COP. Arguments to a method must be Object Manager entities rather than C values.

COP objects are C programming language objects and are declared as such. Database objects are TDL types or their instances. Variables declared as obj are references to database objects. Vbase automatically converts COP values to database objects except object manager strings are converted to C strings. The COP program may access names and named objects directly from the database.

Exceptions are defined in TDL, then raised and handled in COP programs. When an exception is raised an instance of the designated exception is created which is passed through the flow of control. If there is a handler it is caught, otherwise no handler exception is raised.

The COP compiler works with the system catalog and can bind an application to its data and operations at compile time or at run time. The programmer controls this optimization.

Iterators are defined in TDL and implemented in COP. Iterators preclude the necessity of writing loops.

6. **Database Updates**

Two techniques used to create instances in Vbase are invoking a create operation or using a type constructor. The user may specify whether or not newly created objects persist (i.e., permanently stored in secondary storage). Instances may be deleted by the system provided delete operation. If a delete object has inverses they are modified automatically. Vbase does not have garbage collection capabilities. However, the space freed by deleting an object is reused by the object manager.

The Get and Set updates are inherited from type Entity. The programmer may replace the system defined Get and Set.

7. **Tools**

The ITS is a single tool associated with Vbase which includes a source browser, a database browser and a debugger. ITS performs the three functions listed below:

1. Browse the source COP programs.
2. Browse the database to display and modify database objects.
3. Debug COP application programs by a controlled execution.

The browser can be used as a standalone interactive browser or from within the debugger. The standalone browser can look at the type and instance objects, look at the property values and operation definitions, set the property values and invoke operations. The debugger is a superset of the Unix dbx debugger which has been implemented as a preprocessor to dbx. The debugger can be used to execute all dbx commands, set and get property values, supervise application program execution and invoke the browser.

8. **Vbase Database Features**

a. Typing

Strong typing is used throughout the Vbase system. Typing is advantageous where objects are known to have certain value constraints and fixed behavioral properties. The advantages of strong typing are as follows:

1. Many errors may be resolved at compile time.
2. System performance is improved due to analysis of specification by the language processor at compile time.
3. There is less reliance on user-enforced naming and other constraints to convey typing information.

b. Concurrency Control

Vbase supports concurrency control to protect the integrity of the database. The concurrency control facility has two components: a base mechanism to ensure

transaction level database integrity under all circumstances and synchronization primitives for cooperating processes. The base mechanism provides failsafe protection against interference by preventing interfering transactions from committing successfully. The base mechanism should suffice if transactions are short or inexpensive, with a low probability of read/write or write/write conflicts.

When transactions are long or expensive, or there is a high probability of conflicts, the synchronization primitives (semaphore and lock primitives) should be used in conjunction with the base mechanism. The system supplied type primitives are Lock, Sequencer and Event-Count.

Lock is at a higher level of abstraction and is implemented in terms of the lower level abstraction, Sequencer and Event-Count. Users should not work at both levels of abstraction in the same program.

c. Database Recovery

Vbase supports data backup and restore (i.e., Media Recovery) to facilitate logical or physical backup and restoration of specific data in the database. The backup feature copies an entire database as a snapshot of the database at the time of the copy. The snapshot is immutable. There are two types of restoration: database restore, which restores an entire database, and object restore, which restores a selected set of objects. Object restore poses problems in maintaining referential integrity. Referential integrity is maintained automatically in the case of properties with inverses. In the case of properties without inverses, referential integrity is the responsibility of the user's application.

d. Version Control

A system supplied version mechanism is not implemented in current release versions of Vbase. However, the set of tools required to implement version control are available within Vbase and could be implemented by the designer.

e. Database Design

The steps to design the typical database using Vbase are as follows:

1. Identify the objects.
2. Identify their properties as much as possible.
3. Identify the frequent operations performed on the objects.
4. Define the objects using TDL.

5. Compile and debug the TDL definition of the objects.
6. Develop COP routines which implement the operations of the object.
7. Compile and debug the COP programs. [Ref. 7: p. 8]

V. SELECTING AN OODBMS FOR THE LCCDS PROJECT

A. DATABASE SELECTION CRITERIA

In order to make a selection recommendation for an appropriate OODBMS for the LCCDS Project, the basic characteristics of an OODBMS must be addressed.

1. Does the database support user-defined objects?
2. Does the database support user-defined collections of objects?
3. Does the database allow new methods to be defined for existing classes?

Any database which meets these requirements could marginally serve as the basis for the LCCDS Project. However, there are many additional features and enhancements which would be very advantageous to this project.

Object-oriented databases are favored because they provide the designer with a common model in which he can map the mini-world he is concerned with into the database objects by a one-to-one mapping. There is a uniform interface in that all objects are treated consistently by accessing them through their methods (operations). Object-oriented models also provide for designs which allow creation and support of complex objects via a class hierarchy. The object-oriented model also provides for information hiding (through encapsulation) and the support of data abstractions. The internal representation (implementation details) of an object are hidden as the user is presented only a set of methods (called the external interface) to manipulate the object. An advantage that object-oriented databases have over conventional database models is that they facilitate schema modification through their concepts of modularity, flexibility, and extensibility. New objects and operations can be added and old ones modified or deleted, as necessary.

The following system factors must be considered when examining databases:

1. Hardware and software configurations.
2. Storage structures and access paths supported by the DBMS.
3. Types and richness of user interfaces and language interfaces available.
4. Types and richness of query languages available.
5. Recovery, backup, performance, integrity and security mechanisms available.
6. Editors, browsers, graphical design tools, etc., which are available.
7. Ease of use for designers and end users.

B. SPECIFIC REQUIREMENTS OF THE LCCDS PROJECT

During Increment One of the LCCDS Project, an object-oriented database must be chosen to help form the basis of the rest of the project. As the precise requirements, specifications and documents are currently under development, this thesis sought to explore various object-oriented databases which could be adapted to meet the object-oriented specifications of the project. NAVSEA delineated specific requirements which the OODBMS must meet as follows:

1. The database should be object-oriented.
2. The database must provide features for data entry.
3. The database must provide features for a user-friendly query language for building logical and arithmetic relationships between the database elements.
4. The database must provide features to support a powerful output generator for developing screen and hardcopy formats.
5. The database must provide a facility for the user to define new objects.
6. A display graphics capability must interact with the database manager which provides the user with building blocks to define their own customized screen formats.
7. The database manager must be directly coupled with pull down menu options to allow the user to define and change all information as required.
8. The general response time to any user selected display option should be no greater than one-half second. [Ref. 3]

In summary the LCCDS system must be:

1. Able to rapidly summarize and present information needed by the operator in an interactive graphic display format of the operator's choice.
2. Flexible to adapt to changing sensors and threats.
3. Portable to allow distribution to many different environments and to enable incorporation of advances in hardware.

C. COMPARISON OF GEMSTONE AND VBASE

As shown in the Chapter IV, the Iris database management system has many innovative and useful features. It has not as yet, however, been sufficiently tested under operational conditions to allow a complete and unbiased evaluation. This system bears close observation in the future. After planned enhancements are completed and production versions are released, it should be re-evaluated for possible use in the LCCDS Project if the chosen system does not live up to expectations or has not yet been purchased. Since the Iris system is an unproven prototype, the remainder of this chapter will focus on using GemStone or Vbase for this project.

1. **Capabilities**

Both GemStone and Vbase support object inheritance making data structures and code easily reusable. However, a major difference in the GemStone and Vbase database management systems are in their philosophies of object access. In Vbase, the object behavior is elicited by a combination of properties and operations. Properties represent static behavior whereas operations represent dynamic behavior. The definition of a property and its access are syntactically differentiated from those of operations. In the GemStone system, the object/message paradigm of Smalltalk-80 is followed whereby messages are passed to objects via a uniform message syntax to elicit behavior. The programmer must write more code to get and set the values of properties than in Vbase.

Vbase is a strongly typed system which allows for analysis and correction of typing errors at compile time. This often reduces the need for runtime type checks and allows methods to be statically bound. This type checking guarantees that an object will obey a certain protocol (set of operations on a type). When several types implement a common protocol, the types have a common supertype that specifies the protocol. This is contrary to the more accepted OOP terminology in which different classes may respond to the same set of messages (i.e., a common protocol) without having a common ancestor.

GemStone is not strongly typed. This allows variables to be assigned values of different type at different points of execution (thus achieving dynamic binding). Since the GemStone language does not use type declarations, an object can satisfy the set of operations in a type if the type implements (or inherits the implementations) every operation in the set of operations of the type. Performance degradation may occur due to this dynamic binding of methods with type checking completed at run time to achieve the object message behavior. In GemStone, messages are bound to methods late as the lookup cannot be performed until the class of receiver is known. Modifying a class changes the environment and may invalidate bindings. Dynamic (late) binding will attempt to adapt to any changes in the environment. Dynamic binding provides increased flexibility at the expense of efficiency.

GemStone facilitates schema modification through its support of dynamic method creation which makes those methods available immediately to the entire system. This advantage speeds prototyping and application development.

Vbase does not have the capability to easily perform schema modification. Modifications made through the TDL or COP code must be recompiled. Behavioral rigidity enforced by predetermining and prespecifying all operations by a fixed set of

methods is counter to the evolving nature of database technology. This strong typing of Vbase is not an unqualified benefit. It involves a tradeoff between structure and discipline on one hand and flexibility and efficiency on the other.

GemStone and Vbase are both designed for a multi-user, shared environment supporting concurrency, data sharing and data protection. The architectures of both GemStone and Vbase provide for distributed processing.

Both GemStone and Vbase provide interactive data definition, manipulation and query languages, independent of the underlying database. GemStone implemented its query language as a sublanguage of OPAL so that there would be little impedance mismatch. OPAL statements may be executed interactively which should significantly reduce application development time. Object SQL is the Vbase extension of the SQL query language.

A robust OODBMS must be able to store and manipulate large numbers of objects as well as large objects. GemStone can manage up to two billion data objects, and collections of objects may contain up to a billion elements. (It is doubtful that any application will actually reach these limits - it is the intention of GemStone that the user is limited only by their available hardware and their own imagination.) Information about Vbase storage capabilities could not be obtained.

GemStone and Vbase both provide powerful indexing algorithms which eliminate the need to conduct sequential searches of large databases. A clustering feature allows objects to be physically placed on a disk that minimizes retrieval time for specific data access patterns. This feature is user-controlled in both GemStone and Vbase. The client-server architecture of GemStone allows users to distribute the processing load over multiple nodes in the network to enhance processing speed and provide concurrent users with efficient access to objects in the same database.

2. **Product Quality**

The OODBMS must guarantee data integrity and object persistence in spite of machine failure. GemStone preserves all data that has been committed to the database. Vbase preserves all data which have a backup copy.

To guard against disk corruption, facilities must be provided to automatically create and maintain multiple copies of objects on the disk. GemStone allows up to six replicates to be automatically maintained.

GemStone is a transaction based system which controls concurrent access for users in different physical locations and prevents corruption of data. Changes to the GemStone database take place only after a transaction is committed.

3. **Connectivity**

The OODBMS must run on multiple hardware platforms. GemStone currently runs on the full line of VAX and Sun computers. Vbase runs on some of the VAX and Sun computers. Additionally, heterogenous workstations such as those supported by GemStone are necessary to provide flexible development options for the host processor and application environment interfaces.

The OODBMS must be able to interface with popular host languages (with a directed interest in ADA). GemStone databases may be accessed through applications written in Smalltalk, C, FORTRAN, Pascal, Ada, C++, and Objective C. There is a formal interface library to provide direct access for Smalltalk, C, and C++. Vbase does not support any language interfaces other than its own TDL and COP languages.

The OODBMS must be able to utilize the supporting services provided by a collection of heterogenous hardware and software. Further, it must be able to populate the database from outside sources and to extract data for use in other applications. GemStone and Vbase both provide facilities for bulk import/export of data between the database and the outside world. They each also provide interfaces to conventional relational DBMSs. This is a very useful feature because certain aspects of the LCCDS may be better suited to conventional data and applications. Data could be extracted and used from conventional relational databases with a SQL interface.

Integrated development tools for database browsing and debugging at the object model level are also necessary. Options available for editors, browsers, and graphical design tools are provided by both GemStone and Vbase.

4. **Vendor Support and User Satisfaction**

Vendor support is an important aspect for an ongoing project. Technical support and customer service are vital aspects to the functional usefulness of a commercial database management system. Without effective support the user is left to debug, decipher and demystify the operational parameters as intended by the company. The responsiveness of the company should be such that rapid open channels of communications are maintained whereby problems and questions can be solved with minimal time and efforts. In researching this thesis, the manufacturers of Vbase did not meet these parameters. This raises the question of their ability to service and support their product over the long run.

Documentation in the form of effective users manuals is necessary. The manuals must be complete, accurate, understandable and easy to use. They should also include practical examples. Both GemStone and Vbase provide extensive user manuals

with practical examples. There were, however, errors noted in the practical examples included in the Vbase manual.

The OODBMS users (both designers and end users) must be satisfied with the overall system and its performance.

User references provided by Servio Logic Corporation for its GemStone product provided excellent input into the advantages of the system from a practical experience standpoint. Comments were highly favorable on the topics of user satisfaction, quality of manuals, and vendor support. Mr. Trosper's [Ref. 25] favorable review of GemStone was typical of those contacted.

Vbase did not receive favorable reviews from prior Vbase users contacted at the Naval Postgraduate School. MAJ Huskins' [Ref. 26] views and comments were typical of the user views which follow. The users felt that the concepts of Vbase were sound but they were not satisfied with the system itself. Many of the practical examples in the user's manual did not function properly due to numerous errors. The user satisfaction was very low due mainly to the difficulty of programming the system and the slow compilation times.

D. OODBMS RECOMMENDATIONS

The complex nature of this project and data intensive operations required definitely point to the need for an object-oriented database. Those systems available commercially which might sufficiently address the needs of the LCCDS Project are GemStone and Vbase.

As of October 1989, Ontologic no longer sold nor provided support for its Vbase Database Management System. However, in November 1989, Ontologic released an OODBMS named Ontos as a follow-on product. According to Mr. Hanna of Ontologic [Ref. 27], Ontos has a client-server architecture with a backend similar to Vbase and uses a C++ Glockenspiel compiler instead of separate TDL and CCP compilers. The company claims that this has greatly reduced the prior Vbase problems with large compilation times. If more information becomes available on this product, Ontos should be evaluated for possible use in the LCCDS Project if the chosen system does not live up to expectations or has not yet been purchased.

I believe that the OODBMS which currently demonstrates the best potential for the LCCDS project is the GemStone Database Management System. Of the two systems, GemStone provides for the needed flexibility through its implementation of the object message behavior paradigm. It also supports numerous language interfaces (in-

cluding an ADA interface) and runs on several hardware platforms. Vendor support and the high level of user satisfaction with GemStone have also been taken into account.

VI. CONCLUSIONS AND RECOMMENDATIONS

A. SUMMARY AND CONCLUSIONS

This thesis concentrated on a survey of three object-oriented database management systems which might be used as a basis for the Low Cost Combat Direction System Project. Conventional databases only handle conventional data. Advantages of using an object-oriented system are that it offers the designer a high level of flexibility and power to implement arbitrarily complex and data intensive operations. This approach offers increased modeling power, a high level of abstraction, and the ability to inherit and refine object properties

The object-oriented databases presented in Chapter IV are typical of the databases commercially available which might be used to form the basis of the LCCDS Project.

While the Iris database has many innovative and useful features, it has not been sufficiently tested under operational conditions to allow a complete and unbiased evaluation. However, Iris served as a useful comparison with the other two systems.

As of October 1989, Vbase was no longer available commercially nor was support available for this product from Ontologic. A follow-on OODBMS called Ontos was recently introduced by the company. Unfortunately, lack of information about Ontos and time constraints did not permit an evaluation of this new product in this thesis. Ontos should be evaluated for use in the LCCDS if further information becomes available prior to the purchase of the chosen system. Since the functionality and many of the object manager concepts of Ontos are supposed to be similar to Vbase, a portion of the comparisons between Vbase and GemStone may be relevant.

Key areas which differed between the GemStone and Vbase databases included.

1. Object access philosophies.
2. Ease of database schema modifications.
3. Ability to run on multiple hardware platforms.
4. Availability of language interfaces (particularly ADA).
5. Level and responsiveness of customer support.

A major difference in the databases is in their philosophies of object access. Vbase invokes operations on objects while GemStone passes messages to objects.

Vbase does not have the capability to easily perform schema modification (a supposed advantage of OODBMS). Modifications made through TDL or COP code must be recompiled. Behavioral rigidity or predetermining and prespecifying all operations by a fixed set of methods that is counter to the evolving nature of database technology. GemStone, on the other hand, facilitates schema modifications.

B. RECOMMENDATIONS

In order to recommend an OODBMS for this project, factors besides whether the database met object-oriented guidelines were also considered. These included assessing the various platforms the system could operate on (both hardware and software). The storage structures and access paths supported by the DBMS were investigated as were the database applications for recovery, backup, performance, integrity, and security. The user interfaces and programmer interfaces were also important facilities as well as the query languages available. Options important to the database designer and maintainers such as editors, browsers, and graphical design tools were also considered.

In conclusion, after a studied analysis of major commercial suppliers of OODBMS, it is my opinion that the system of choice for the LCCDS Project should be the GemStone Database Management System.

LIST OF REFERENCES

1. Department of the Navy NAVSEA Publication 0967-LP-027-8602, *Combat Direction System, Model 5, Systems Engineering Handbook*, v. 1, February 1985.

2. Swenson, E., and others, "NTDS-A Page in Naval History," *Naval Engineers Journal*, pp. 53-61, May 1988.

3. Commander, Naval Sea Systems Command UNCLASSIFIED Letter 9410 OPR:61Y Serial 61Y 1036 to Superintendent, Naval Postgraduate School, Subject: Statement of Work for Low Cost Combat Direction System, 20 December 1988.

4. Cox, B., *Object Oriented Programming. An Evolutionary Approach*, Addison-Wesley Publishing Company, Menlo Park, CA, 1987.

5. Nierstrasz, O., "A Survey of Object-Oriented Concepts" in *Object-Oriented Concepts, Databases, and Applications*, ACM Press, New York, NY, 1989.

6. Meyer, B., *Object-Oriented Software Construction*, Prentice Hall Inc., Englewood Cliffs, NJ, 1988.

7. Ketabchi, M., *Object Oriented Database Management Systems for Complex Data and Process Intensive Applications* (Course Notes), Santa Clara University, Santa Clara, CA, September 1987.

8. Maier, D., "Making Database Systems Fast Enough for CAD Applications" in *Object-Oriented Concepts, Databases, and Applications*, ACM Press, New York, NY, 1989.

9. *GemStone Product Overview*, Servio Logic Corporation, Alameda, CA, 1988.

10. Maier, D., and others, "Development of an Object-Oriented DBMS," *SIGPLAN Notice*, v. 21, pp. 472-482, September 1986.

11. Purdy, A., and others, "Integrating an Object Server with Other Worlds," *ACM Transactions on Office Information Systems,* v.5, No. 1, January 1987.

12. Bretl, R., and others, "The GemStone Data Management System" in *Object-Oriented Concepts, Databases, amd Applications,* ACM Press, New York, NY, 1989.

13. Penney, D. and Stein, J., "Class Modification in the GemStone Object-Oriented DBMS," *SIGPLAN Notices,* v. 22, pp. 111-117, December 1987.

14. Ketabchi, M., *Introduction to GemStone* (Course Notes), Santa Clara University, Santa Clara, CA, September 1987.

15. *Selecting an Object-Oriented Database Management System,* Servio Logic Corporation, Alameda, CA 1988.

16. Fishman, D.H., and others, "Overview of the Iris DBMS" in *Object-Oriented Concepts, Databases, and Applications,* ACM Press, New York, NY, 1989.

17. Freytag, J.C., and others, "Mapping Object-Oriented Concepts into Relational Concepts by Meta-Compilation in a Logic Programming Environment" in *Lecture Notes in Computer Science - Advances in Object-Oriented Database Systems,* Springer-Verlag, New York, NY, 1988.

18. Ketabchi, M., *Introduction to IRIS* (Course Notes), Santa Clara University, Santa Clara, CA, September 1987.

19. Fishman, D.H., and others, "Iris: An Object-Oriented Database Management System," *ACM Transactions on Office Information Systems,* v. 5, No. 1, January 1987.

20. Andrews, T. and Harris, C., "Combining Language and Database Advances in an Object-Oriented Development Environment," *SIGPLAN Notices,* v. 22, pp. 430-440, December 1987.

21. Ketabchi, M., *An Introduction to Vbase* (Course Notes), Santa Clara University, Santa Clara, CA, September 1987.

22. *Vbase For Object Applications,* Ontologic Inc., Billerica, MA, 1988.

23. Elmasri, R. and Navathe, B., *Fundamentals of Database Systems,* The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, 1989.

24. *Vbase Integrated Object Database User's Manual,* Ontologic Inc., Billerica, MA, 1987.

25. Telephone conversation between Robert Trosper, Evans & Sutherland, Salt Lake City, UT, and the author, 5 December 1989.

26. Telephone conversation between Jim Huskins, Major, USA, Code 37, Naval Postgraduate School, Monterey, CA, and the author, 5 December 1989.

27. Telephone conversation between Bruce Hanna, Ontologic Corporation, Billerica, MA, and the author, 8 December 1989.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Chief of Naval Research 800 Quincy Street Arlington, VA 22217	1
4.	Center for Naval Analysis 4401 Ford Avenue Alexandria, VA 22302-0268	1
5.	Naval Sea Systems Command Attn: Code 61Y National Center #2, Suite 7N06 Washington, D.C. 20363-5100	1
6.	Office of Naval Research Attn: CDR Tim Wells Code 1224 800 N. Quincy Street Arlington, VA 22217	1
7.	Director of Research Administration Attn: Professor Howard Code 012 Naval Postgraduate School Monterey, CA 93943	1
8.	Chairman, Code 52 Computer Science Department Naval Postgraduate School Monterey, CA 93943-5100	1
9.	Dr. Luqi Code 52LQ Computer Science Department Naval Postgraduate School Monterey, CA 93943	14

10. MAJ Michael L. Nelson 4
Code 52NE
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

11. LT Debra L. Ross 2
6417 Copperhead Court
Waldorf, MD 20603