NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

INTERCONNECTION OF THE GRAPHICS LANGUAGE FOR DATABASE SYSTEM TO THE MULTI-LINGUAL, MULTI-MODEL, MULTI-BACKEND DATABASE SYSTEM OVER AN ETHERNET NETWORK

by

Thomas Rivero Hogan

December 1989

Thesis Advisor: C. Thomas Wu

Approved for public release; distribution is unlimited

Unclassified

Security Classification of this page

REPORT DOCUMENTATION PAGE

1a Report Security Classification Unclassified	1b Restrictive Markings
2a Security Classification Authority	3 Distribution Availability of Report
2b Declassification/Downgrading Schedule	Approved for public release; distribution is unlimited.
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*) 52
6c Address (*city, state, and ZIP code*): Monterey, CA 93943-5000	7a Name of Monitoring Organization: Naval Postgraduate School
	7b Address (*city, state, and ZIP code*): Monterey, CA 93943-5000
8a Name of Funding/Sponsoring Organization	8b Office Symbol (*If Applicable*)
8c Address (*city, state, and ZIP code*)	9 Procurement Instrument Identification Number
	10 Source of Funding Numbers

Program Element Number	Project No	Task No	Work Unit Accession No

11 Title (*Include Security Classification*) Interconnection of the Graphics Language for Database System to the Multi-Lingual, Multi-Model, Multi-Backend Database System Over an Ethernet Network

12 Personal Author(s) Hogan, Thomas R.

13a Type of Report	13b Time Covered		14 Date of Report (*year, month,day*)	15 Page Count
	From	To		
Master's Thesis			1989 December	136

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 language, graphical interface, database language, GLAD, MBDS, THESIS, KMS, [illegible]

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

In recent years, the proliferation of database models required to meet today's ever changing database needs has led to a variety of Database Management System (DBMS) designs. This situation has presented a challenging problem for database managers. How does one access information from heterogeneous databases without having to learn and utilize the host's database model and language? The answer seems to lie in the area of multi-lingual, multi-model database systems. Such systems allow the user to access data from any database, using any language the user is familiar with, regardless of the model used to create the database.

An experimental system has been developed and implemented at the Naval Postgraduate School in Monterey, California. One problem with the current system is the lack of a consistent and user-friendly interface to interact with the system. Many different visual interfaces for databases have been proposed in recent years, and one of the most promising is GLAD (Graphics Language for Database). GLAD utilizes direct manipulation of database objects through the use of buttons and other controls. The purpose of this thesis is to enhance the GLAD interface and allow it to transparently communicate with a backend data server, in this case an ISI mini-computer running the Multi-Lingual, Multi-Model, Multi-Backend Database System (MBDS) software, allowing users to access heterogeneous databases in a simple and intuitive manner.

20 Distribution/Availability of Abstract	21 Abstract Security Classification
[X] unclassified/unlimited [ ] same as report [ ] DTIC users	Unclassified

22a Name of Responsible Individual	22b Telephone (*Include Area code*)	22c Office Symbol
Professor C. Thomas Wu	(408) 646-3391	Code 52 Wq

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.

Interconnection of the Graphics Language for Database System to the Multi-Lingual, Multi-Model, Multi-Backend Database System Over an Ethernet Network

by

Thomas Rivero Hogan
Lieutenant, United States Navy
B.S., University of California at Los Angeles, 1982

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: _____
Thomas Rivero Hogan

Approved by: _____
C. Thomas Wu, Thesis Advisor

_____
David K. Hsiao, Second Reader

_____
Robert B. McGhee, Chairman, Department of Computer Science

ABSTRACT

In recent years, the proliferation of database models required to meet today's ever changing database needs has led to a variety of Database Management System (DBMS) designs. This situation has presented a challenging problem for database managers. How does one access information from heterogeneous databases without having to learn and utilize the host's database model and language? The answer seems to lie in the area of multi-lingual, multi-model database systems. Such systems allow the user to access data from any database, using any language the user is familiar with, regardless of the model used to create the database.

An experimental system has been developed and implemented at the Naval Postgraduate School in Monterey, California. One problem with the current system is the lack of a consistent and user-friendly interface to interact with the system. Many different visual interfaces for databases have been proposed in recent years and one of the most promising is GLAD (Graphics Language for Database). GLAD utilizes direct manipulation of database objects through the use of buttons and other controls. The purpose of this thesis is to enhance the GLAD interface and allow it to transparently communicate with a backend data server, in this case an ISI mini-computer running the Multi-Lingual, Multi-Model, Multi-Backend Database System (MBDS) software, allowing users to access heterogeneous databases in a simple and intuitive manner.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to express my sincere thanks to the many people who helped me prepare and implement this thesis. In particular, I would like to thank Debbie Gaiser, Tom Chu, and Marciano Pitargue for their invaluable help and encouragement from the birth of this idea to its fruition. I would also like to thank Bill Sympson for his patience and suggestions, and my advisor Dr. C. Thomas Wu for his help in the conception and preparation of this thesis.

I would also like to thank my family, particularly my beautiful wife Janet for her patience, encouragement and love, without which none of this would have been possible.

I. INTRODUCTION

A. MOTIVATION

In recent years, the proliferation of database models required to meet today's ever changing database needs has led to a variety of **Database Management System (DBMS)** designs. This situation has presented a challenging problem for database managers. How does one access information on heterogeneous databases without having to learn and utilize the host's database model and language? The answer seems to lie in the area of multi-lingual, multi-model database systems. Such systems allow the user to access data from any database, using any language the user is familiar with, regardless of the model used to create the database. For example, a user may utilize SQL transactions to access a hierarchical database thereby relieving the user of the burden of learning the DL/I data manipulation language. As the number and complexity of data models and languages increase, the benefits of such a scheme become increasingly obvious.

Such a system has been developed and implemented at the Laboratory for Database Systems Research at the Naval Postgraduate School in Monterey, California. One problem with the current system is the lack of a consistent and user-friendly interface with which to interact with the system. Many different visual interfaces for databases have been proposed in recent years and one of the most promising is **GLAD** (**G**raphics **LA**nguage for **D**atabase), which supports a high-level semantic data model. The GLAD interface utilizes direct manipulation of objects in the database through the use of buttons and other controls which make using and learning the system extremely easy. Direct manipulation consists of the following features (Shneiderman, 1987, p. 201):

- continuous representation of the objects and actions of interest
- physical actions or labeled button presses instead of complex syntax
- rapid incremental and reversible operations whose impact on the object of interest is immediately visible.

The ease of use of direct manipulation allows designers to create systems with the following beneficial attributes (Shneiderman, 1987, pp. 201,202):

- novices can learn basic functionality quickly, usually through a demonstration by a more experienced user
- experts can work rapidly to carry out a wide range of tasks even defining new functions and features
- knowledgeable intermittent users can retain operational concepts
- error messages are rarely needed
- users can immediately see if their actions are furthering their goals, and, if not, they can simply change the direction of their activity
- users experience less anxiety because the system is comprehensible and because actions are so easily reversible
- users gain confidence and mastery because they are initiators of action, they feel in control, and the system responses are predictable.

Currently a basic system has been developed which is limited to accessing database files which exist on the personal computer the GLAD system is being run on. The purpose of this thesis is to enhance the GLAD interface and allow it to transparently communicate with a backend data server, in this case an ISI mini-computer running the Multi-Lingual, Multi-Model, **Multi-Backend Database System (MBDS)** software, to build a consistent and user-friendly interface which could later be ported to other existing backend data servers. The goal of this thesis is twofold; the enhancement of the current MBDS interface, allowing users to utilize a consistent and user-friendly interface to

access multi-model databases, and the expansion of GLAD's communication capabilities in order to allow users to utilize its interface to access remote databases in a transparent fashion.

This thesis focuses on the implementation of a "server" version of the **Attribute Based Data Language (ABDL)** interface of MBDS which receives requests (generated by GLAD) for opening and querying databases. These requests are serviced and the results are then sent over the Ethernet to an IBM-compatible personal computer by **TCP/IP (Transmission Control Protocol/Internet Protocol)** socket interfaces that are implemented on both sides of the network. The opening and querying of databases is controlled by the GLAD interface, utilizing messages sent through the socket interfaces to the MBDS server.

B. THESIS OVERVIEW

Chapter II will present some background information for the thesis, beginning with an overview of the Microsoft Windows Operating environment. Discussion will include Windows' **Dynamic Data Exchange (DDE)** which allows applications to communicate and exchange information with other Windows applications. We then present a discussion of Object oriented programming and the Actor software development environment. Next we will describe the Multi-Backend, Multi-Lingual, Multi-Model Database System and its Attribute Based Data Language and Model. Lastly we will discuss the Transmission Control Protocol/Internet Protocol and the socket abstraction.

Chapter III will explain the implementation of this thesis and its three distinct programming environments (UNIX C, Microsoft C with the Windows Software Development Kit, and Actor). We will begin by discussing the MBDS Server code which allows GLAD to manipulate MBDS databases. Next, we present a discussion of the Socket Interface code which allows GLAD and MBDS to communicate. Finally, this

chapter presents the GLAD interface code and contrasts the old MBDS interface with the "direct manipulation" style GLAD interface.

Chapter IV provides a summary discussion of the research as well as a complete presentation of many future enhancements which can, and are being implemented in this area.

II. BACKGROUND

A. MICROSOFT WINDOWS OPERATING ENVIRONMENT

User interfaces have come a long way since the introduction of the personal computer, and although the Microsoft Windows environment is far from ideal, it provides an excellent operating environment for the GLAD system. Windowing interfaces got their start at the XEROX Palo Alto Research Center (PARC) in the mid-1970's. They were popularized by Apple Computers, particularly the Macintosh line, and later by Microsoft and other developers. The Windows environment was chosen for this thesis primarily because of its user-friendly and intuitive graphical interface, but it offers numerous other advantages including multitasking, hardware independence, and dynamic data exchange.

1. Graphical User Interface

The most readily apparent advantage of the Windows environment is the graphical user interface. The interface is a stark contrast to the infamous C> prompt of the DOS Operating System. The Windows interface is based on *graphics objects* which are described by David Durant as "a collection of data that can be manipulated as a whole entity and that are presented to the user as part of the visual interface." (Durant, 1987, p. 3) Examples of these graphics objects include menus, dialog boxes, and icons. There is even a class of graphics objects which are used to create other graphics objects.

In the Windows environment, menus and direct manipulation of objects have replaced the cryptic DOS command line syntax. The window is the basic unit of any Windows program, and users can have many windows on the screen simultaneously. Although several Windows programs may be running simultaneously, only one can have

the user input focus at any one time. Users are often able to interact with their programs without ever touching the keyboard by using a mouse or other pointing device. Some of the features which allow this flexibility are pictured in Figure 1, and include:

- **Minimize Box** - reduces the window to an icon.
- **Maximize Box** - enlarges the window so that it takes up the entire screen.
- **Menu Bar** - displays the top-level menu choices.
- **Scroll Bars** - allow the user to page up or down through text or lists.
- **System Box and Menu** - allows the user to close the window and perform functions such as minimization, maximization, moving and restoration of the window.

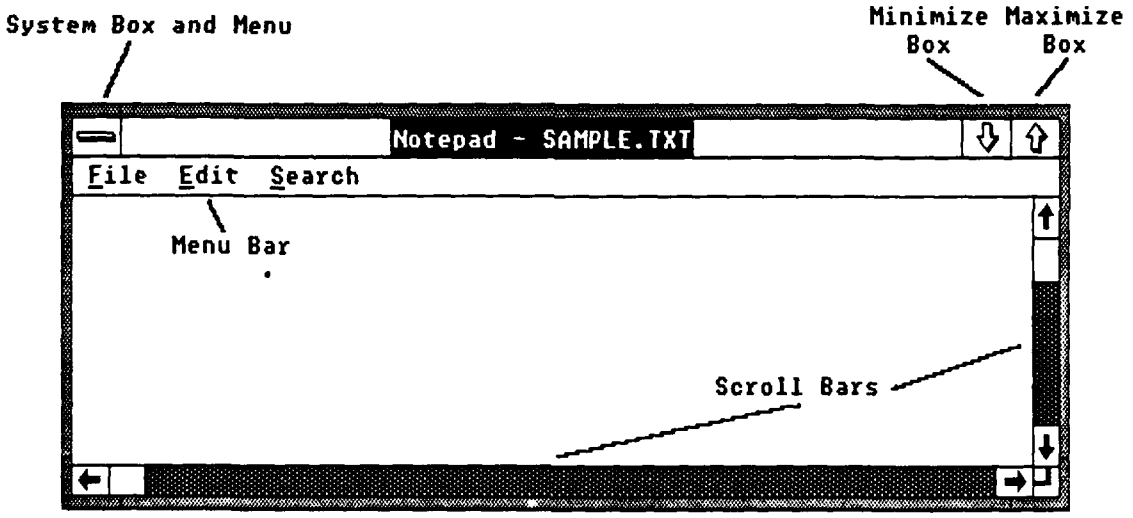


Figure 1. Typical Window

These interface and application tools are not only consistent within each application, but from application to application and therefore new programs are extremely easy to learn, reducing the need for costly training, and increasing user satisfaction. The visual interface also has benefits for the programmer since Windows

provide the **Graphics Device Interface (GDI)** which contains the routines for drawing menus, windows, dialog boxes and many other graphics objects, thereby relieving the programmer from the burden of doing so, making the design and implementation of an intuitive and elegant user interface almost effortless.

2. Multitasking

One of the most significant buzzwords regarding operating environments today is multitasking. Unlike the terminate and stay resident (TSR) programs of DOS, several windows can be displayed and running at any one time. They do not run concurrently, nor do they use a preemptive type of multitasking, but instead run in a message-driven environment. Every action or process within a Windows program must be initiated by a message. These messages can come from the Windows operating system, user input, other programs, or even the program itself. Most Windows programs are simply a loop which receives and translates messages, taking action according to the content of those messages. Windows multitasking environment means one can be downloading information via modem, recalculating a spreadsheet and typing a thesis without having to switch from program to program. This feature is no small feat considering the fact that the DOS system under which Windows runs is not a multitasking environment. Windows multitasking also has another advantage; it allows a user to have several copies, or instances, of a program running at the same time with all the instances sharing the same code segment thereby reducing the memory requirements of each instance.

3. Hardware Independence

An equally important characteristic of the Windows environment is hardware independence. What this means, basically, is that if your program will run under Windows on a particular computer, then it will work on any computer that Windows supports, provided it has sufficient memory. This also means that your screen layouts

will work on any monitor (monochrome, EGA, VGA, etc.), or printer output on any brand of printer, as long as you have coded your program to take advantage of the hardware-independent capabilities of Windows. This relieves the programmer of the burden of writing code to support the incredible number of different peripheral devices available and allows him to spend more time on the interface and functionality of his program.

4. Dynamic Data Exchange

The last important aspect of the Windows environment that we will look at is it's ability to pass information between applications that are currently running through the use of **Dynamic Data Exchange Protocol (DDE)**. This information can be passed without the user explicitly asking for it, and can be used to dynamically link applications. An example of this would be a spreadsheet program that receives information from a communications program that has downloaded the latest information on the user's stock market portfolio, allowing the spreadsheet to automatically update itself. Data is passed in global memory objects which are accessible to all currently running applications.

Programs using DDE must follow the protocol established by Windows to pass their information. Typically there is a *client* program and a *server* program. The client program will initiate a "conversation" with a WM_DDE_INITIATE message which will specify the requested application name and topic. The application name and topic are referenced by *atoms* which are stored, appropriately, in an atom table. Atoms are simply a word-length integer reference to a string that avoids the problems inherent in handling variable length strings when passing DDE messages. There are two types of atom tables in Windows, *global* and *local*. Global atoms are accessible to any application that is running, whereas local atoms are accessible only to the application that created them. In general, only global atoms are used. The WM_DDE_INITIATE message is usually sent

to all windows, and if an appropriate server exists that can service the requested topic, it will send a WM_DDE_ACK message containing it's window *handle*. Most objects in the Windows environment have a handle which, like an atom, is a more convenient way to reference an object. After the acknowledgment is received all requests for information will be sent to the acknowledging window. There are several ways to request data, however we will limit our description to the method utilized in the implementation of this thesis which is the WM_DDE_REQUEST message.

When the client application requires data it sends a WM_DDE_REQUEST message to the server application. The message can request acknowledgement of receipt of the message, and a particular data format for the data. If no acknowledgement is required, the server gathers the data in the requested format, if possible, and sends a WM_DDE_DATA message which contains the identity of the data and a handle to the memory object which contains the data. If acknowledgement is required the server first sends the WM_DDE_ACK message and then gathers and sends the requested data. The conversation continues in this manner until the client no longer requires the services of the server and sends a WM_DDE_TERMINATE message.

B. OBJECT-ORIENTED PROGRAMMING (OOP) AND ACTOR

There has been much excitement in recent years among programmers over the introduction of object-oriented languages. These languages promote the reusability of code and a highly modular view of programming. The design of object-oriented languages varies greatly, and there has been much confusion as to what features a truly object-oriented language should contain. In the following section we shall attempt to clarify this question while familiarizing the reader with the basics of object-oriented programming in general, and the Actor language specifically.

1. Elements of Object-Oriented Programming

a. Objects

The basic building block of any OOP program is the object. In our daily existence we are used to dealing with objects (e.g., houses, airplanes, computers, people). OOP languages attempt to model real world or conceptual objects through the use of data abstraction. In an OOP language an *object* can be defined as a set of "operations" and a "state" that remembers the effect of operations. "Objects may be contrasted with functions, which have no memory." (Wegner, 1987, p. 168) In procedural languages, functions work on data that has been passed to them as parameters. As an example, to find the log of an integer x, you would have to send x as a parameter to the log function. Object-oriented languages use a data or object-centered approach to programming. "Instead of passing data to procedures, you ask objects (data) to perform operations on themselves." (Pascoe, 1986, p. 139) In the above example, you would send a log message to the data object x. In the Actor programming language, everything is considered to be an object, including integers, strings, arrays and windows.

b. Classes

Each ship in the U.S. Navy belongs to a certain class, and therefore has a structure and behavior similar to other ships in the same class. Similarly, in object-oriented programming, objects belong to certain *classes* which characterize their behavior and structure. Objects which are defined to be of the same class (also called the *instances* of the class) should contain the same type of information and react similarly to any particular message that is sent to them. As an example, if we defined a ship class, we might expect all objects declared to be of this type to have a name, hull number, and ship type. We also would expect them to react similarly to a "change_name" or "change_hull_number" message. The structure of a class is defined by variables which

contain the data specific to that class. In Actor this data is private to each class and is stored in *instance variables*. The only way that other objects may access this data is through the *methods* which define the behavior of an object of the class.

c. Methods vs. Messages

One source of great confusion in OOP is the difference between methods and messages. To put it simply, *methods* define the behavior of an object in response to a *message* that is sent to the object. One constraint placed on methods and messages is that the method and message name must be the same. Continuing with our ship class example, if we wanted to change the hull number of a ship object "Big_Ship", we would send a change_hull_number message to the object Big_Ship, which would then perform the change_hull_number method that is defined for an object of that class (assuming it has been defined). The Actor syntax for such a message would be

```
change_hull_number(Big_Ship, 17)
```

assuming we wanted to change the hull number of Big_Ship to 17. Every action that occurs in Actor (except for calling Microsoft Windows or Microsoft DOS) is the result of sending a message to an object, which responds to it by executing a method (Duff, 1989, p. 29).

Methods defined for instances of a class (called *object methods*) are not the only kind of methods in OOP languages. For example, if a new instance of a class is required we must send a *new* message to the class itself. The methods that respond to messages sent to the class are referred to as *class methods*.

d. Inheritance

Perhaps one of the most powerful features of OOP languages is *inheritance*. Inheritance allows objects to inherit the structure and behavior of other objects, thereby reducing the amount of recoding necessary to create objects which are similar to

previously defined objects. This greatly enhances the reusability as well as the maintainability of the code. As an example, we could define the class "carrier" which would inherit the structure and behavior of the ship class, including all its instance variables and methods, and then simply add data and methods that are unique to this type of ship to the new class. The ship class would then be referred to as an *ancestor* class of the carrier class. For example, the carrier class might have a "number of aircraft" instance variable, and an "add_aircraft" method. If we defined an object "Nimitz" of the carrier class we could now use any of the methods defined for the ship and carrier classes to manipulate this object.

Using the above example, if we were to send a change_name message to the object Nimitz it would first look for a change_name method in the carrier class. In this case we have not defined one so it will check its ancestor class, ship, to see if it has defined such a method. This will continue until the correct method is found or there are no more ancestors to investigate and an error message will be generated. Actor only supports *single inheritance*, in which a class can have only one ancestor class. Some OOP languages support *multiple inheritance* in which each class can have more than one ancestor class. Actor utilizes the concept of a *class tree* whose root is the Object class. All other classes are descendants of the Object class, and any classes created using Actor must be descendants (not necessarily direct descendants) of Actor's previously defined classes. Actor provides over 100 predefined classes including over 1000 methods (Duff, 1989, p. 47).

e. Polymorphism

"Polymorphism allows program entities to refer to objects in more than one class, and through dynamic binding, the ability for those objects to respond uniquely to the same message." (Floyd, 1989, p. 60) Polymorphism allows programmers to avoid

the complex control structures which would be required to ensure that a correct method is executed if unique method names were required. For example, polymorphism allows us to define separate "print" methods for a ship and a carrier even though these methods may be totally different. The program determines at run-time who the correct receiver of the message should be ensuring that the correct "print" method will be executed.

f. Information Hiding

"Information hiding is important for ensuring reliability and modifiability of software systems by reducing interdependencies between software components." (Pascoe, 1986, p. 140) The current state of an object is contained in private variables (In Actor, these are called instance variables). These variables are only accessible to that particular instance of the object. One normally only accesses these variables through the methods within the object which have been defined to manipulate the data. Actor also allows access to the instance variables using a "dot" notation. For example, if we had an object "John" which had an instance variable "Age", we could assign it the value of 25 by the following statement.

```
John.Age := 25
```

As discussed previously, the variables and methods of objects are defined in software constructs known as a classes. These classes are "software modules" which hide design decisions so that later changes in these design decisions will not effect other objects which might use the module. As long as the names of the methods remain unchanged, there will be no need to change other modules which use an object simply because the algorithm used to accomplish a task has changed. (Pascoe, 1986, p. 140)

2. The Actor Development Environment

The Actor development environment is unique in that it is currently the only program development environment which has taken advantage of the Microsoft Windows

operating environment. Windows' ease of use, multitasking and multiple windows make the environment a pleasure to use. Prior to the introduction of the Actor object-oriented development environment, Windows programs were primarily written in C. Programmers required extensive knowledge and patience, since about two pages of code were required to do something as simple as place a window on the screen. Actor's more than 100 predefined classes makes this an easy task, requiring only two lines of code. As a matter of fact, with two lines of code you can create a simple text editor with cut, copy and paste capabilities. This type of power and simplicity made Actor the perfect choice for the rapid prototyping desired for the GLAD system. Actor also gives you access to over 600 Windows Software Development Kit routines available to aid in creation of Windows applications.

The Actor environment provides numerous tools for creating and debugging programs and even allows "on the fly" programming, making changes while the program is actually running, so you can immediately see the effects of your changes. Figure 2 shows some of the tools included with Actor which are described below.

- **Workspace Window** - Actor's "home base" from which you can run and create programs, perform static memory garbage collection, and access Actor's other powerful features.

- **Browser Window** - Perhaps Actor's most useful and powerful tool, the browser window allows you to display, edit, and add to Actor's source code, including that of the development environment itself.

- **Inspector Window** - Allows users to inspect the methods and data of Actor's objects, and even allows you to send messages to the object being inspected.

- **Debug Window** - Similar to the Browser window, this window allows you to inspect and change methods or values of instance variables in the middle of program execution. It then allows you to resume the execution of the program so you can see the effect of your changes.

In addition to Microsoft Windows, Actor version 1.2 requires a hard disk, 640K RAM, graphics display and adapter, a mouse (or other pointing device) and MS-DOS version 2.0 or higher.

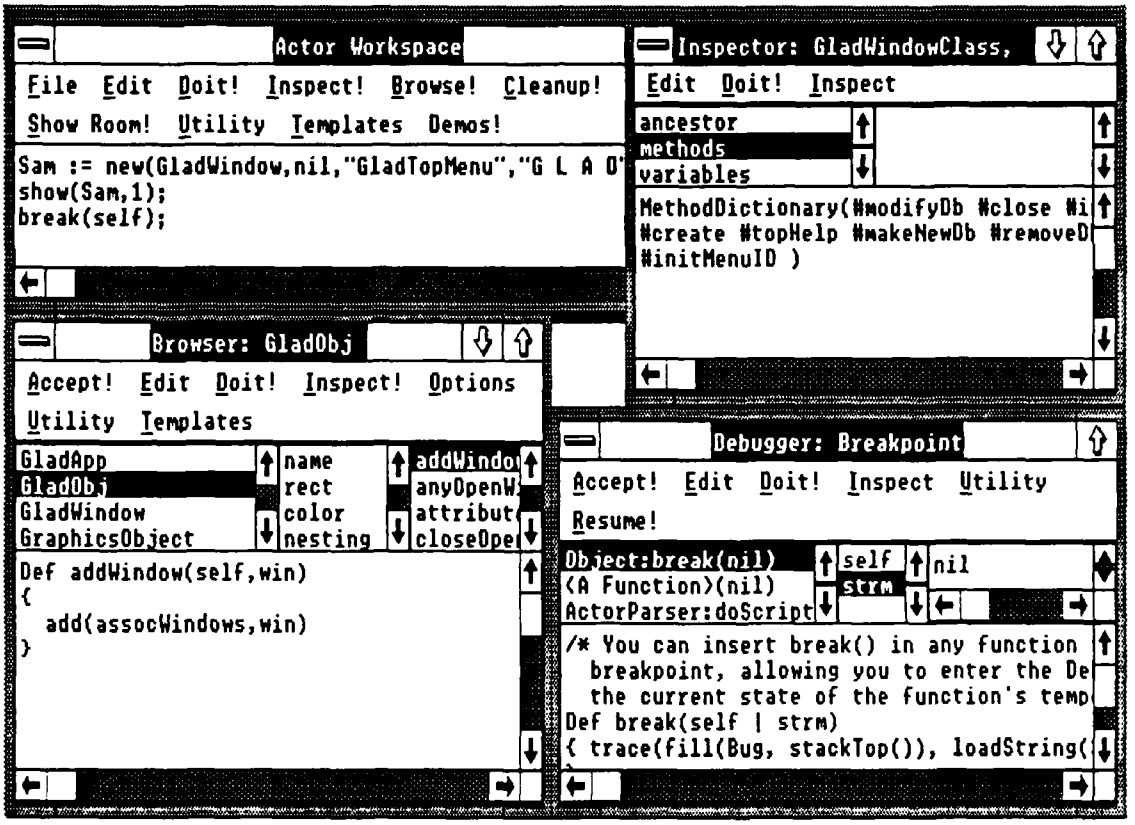


Figure 2. Actor Development Environment

C. THE BACKEND DATABASE SYSTEM

Since the introduction of the database applications, many different data models and languages have been developed and have presented the database manager with a difficult question. Which data model and language should be used for a particular database need? Historically this question was answered in a trial-and-error fashion or through the high pressure pitch of the local database salesman. There was no simple way to do a hands-on

evaluation of contrasting models and languages, and once a model was chosen, there was no way to change, other than creating a new database from scratch using a different database model. Errors in judgment proved costly from both an efficiency and financial standpoint.

Updating of hardware or DBMS software also proved a problem since a change in data model or language meant that previously developed transactions were no longer useable. With the proliferation of data models and languages came the problem of learning, much less mastering, the intricacies of each new language. Networking made the problem even more obvious, since users now had access to databases worldwide, but lacked the skills necessary to query databases which used unfamiliar models or languages. The **Mult-Lingual, Multi-Model, Multi-Backend Database System** has presented an exciting solution to these problems by allowing users to query databases designed utilizing any model, using the data manipulation language of their choice. For example, it is possible to query hierarchical databases via SQL transactions. We will now describe the components of this innovative system.

1. Multi-Lingual Database System

The **multi-lingual database system (MLDS)** allows users to experiment with various data models and languages within an integrated database development environment. Although the system supports many different data models and languages, the underlying system supports only a single data model and language. These are referred to as the **kernel data model (KDM)** and the **kernel data language (KDL).** The various models supported are mapped to the KDM through a *data-model transformation.* Languages are mapped to the KDL through *data-language translation.* Let us look at how this transformation occurs.

Figure 3 shows the structure of the multi-lingual database system. Users interact with the system through transactions developed in the **language interface layer (LIL)**

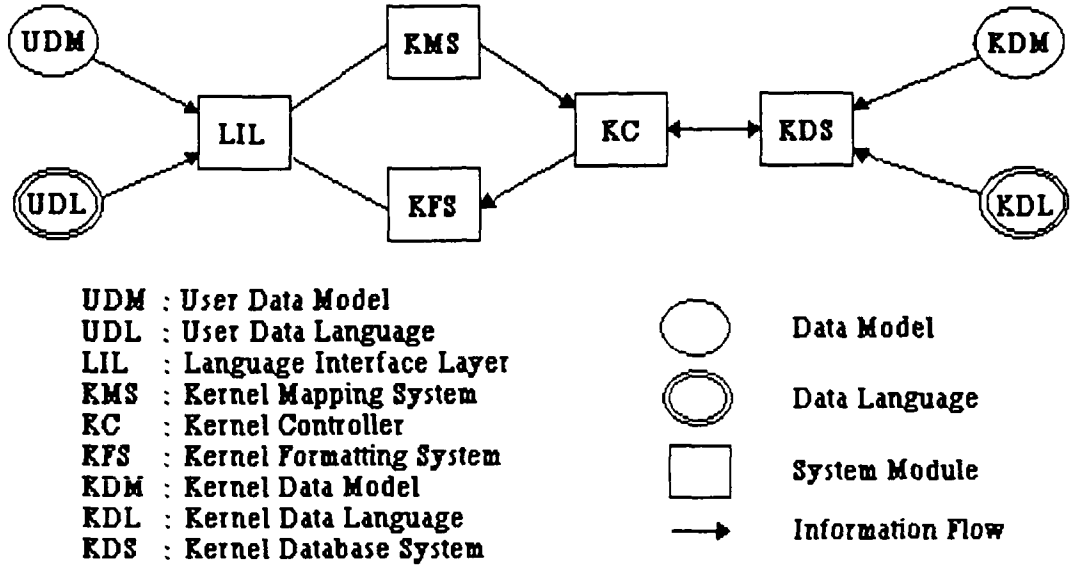


Figure 3. The Mult-Lingual Database System

utilizing a **user-chosen data model (UDM)**. These transactions are written in a **model-based data language (UDL)** that corresponds with the chosen UDM. These transactions are of two general types, either database definition or database manipulation requests. Based on the type of transaction the LIL routes the transaction to the the **kernel mapping system (KMS)** for processing.

If the transaction is a request for database creation, the KMS transforms the UDM-database definition to an equivalent KDM database definition. Next this KDM definition is sent to the **kernel controller (KC)** which forwards it to the **kernel database system (KDS).** When the KDS has finished processing it signals the KC, which informs the user that database loading may begin.

If the transaction is a query of an existing database, the KMS translates the query into an equivalent KDL transaction. This transaction is then sent by the KMS to the KC, which forwards it to the KDS for execution. The query results are sent in KDM form back to KC, which passes them to the **kernel formatting system (KFS)** which translates the query results to the UDM format for display through the LIL.

Figure 4 shows that for each data language that is to be implemented, a new language interface must be constructed. The LIL, KMS, KC, and KFS are the building

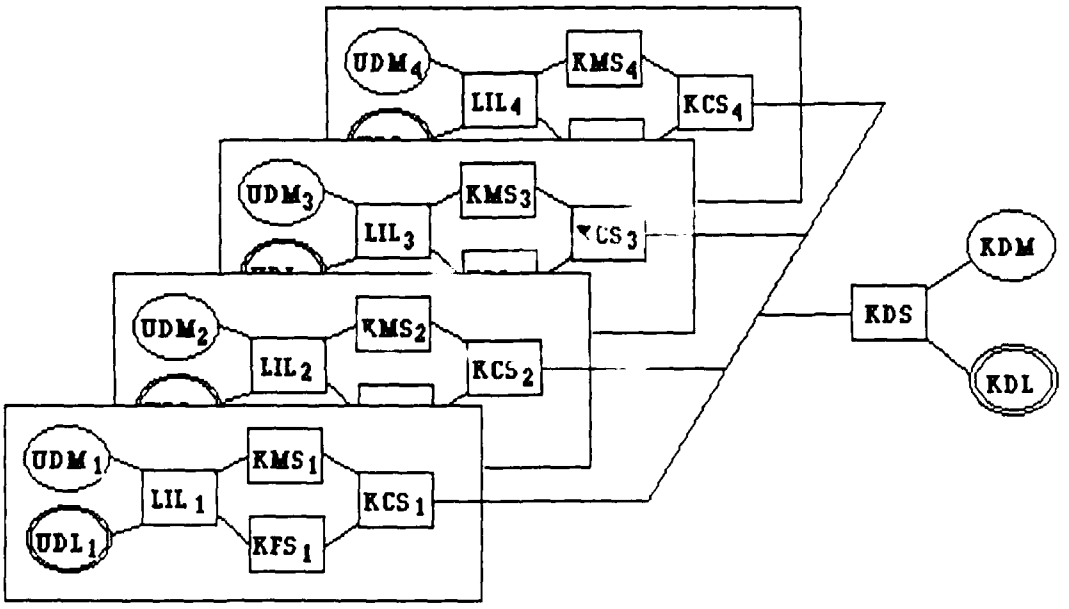


Figure 4. Multiple Language Interfaces for the Same Kernel Database System

blocks for each language interface. The multi-lingual database system currently has interfaces for relational/SQL, hierarchical/DL/I, and network/CODASYL-DML. The KDS and its corresponding KDM and KDL are the same for all interfaces. The **attribute-based data model and language (ABDL)** have been chosen as the KDM and

KDL respectively and will be discussed more fully in the following sections. (Demurjian, 1986, pp. 5-7)

2. Multi-Model Database System

With the ability to experiment with different data models and languages in an integrated system realized through the MLDS, a natural choice for enhancement was the ability to use transactions from any given UDL to query databases of any given UDM. The **Multi-Model Database System (MMDS)** allows a user who might be proficient in only one data model or language, to query any database using his chosen data manipulation language, regardless of the underlying data model. For example, a user can utilize SQL transactions to manipulate a hierarchical database. This greatly enhances the capacity for expansion, and extensibility by allowing existing databases and queries to be kept intact when transferring them to a new database system. This reduces workload and avoids the error prone task of translating existing queries into a new language.

The strategy chosen for this system is the *Mixed-processing Strategy* which is modeled in Figure 5. This figure shows how a user utilizing language interface$_i$ (LI_i) would access a database of LI_j. This strategy utilizes two components, the schema transformer and a second language interface LI_i' which is a language interface very similar to LI_i which the user is currently using.

When the user requests access to a database that is not in the current LI_i, a search is made of the other LIs to find the database. Once the database is found (assume the database was created using UDM_j), the schema transformer takes the UDM_j database of LI_j and creates a UDM_i database for the user of LI_i to access. If the user now attempts to query the database using transactions of his original data language, the new language interface, LI_i', is used to query the database. Because we have used the schema transformer, there is no need to translate the queries themselves into the new model's

language, thereby greatly simplifying and speeding the query process. (Coker, 1987, pp. 7, 8)

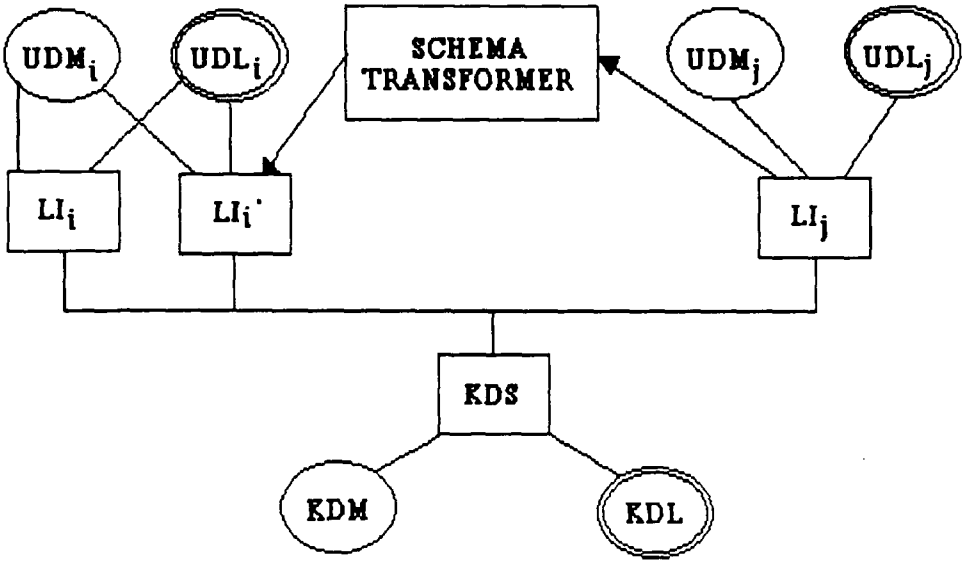


Figure 5. The Mixed-processing Strategy

3. Multi-Backend Database System

The **Multi-backend database system (MBDS)** is an innovative solution to the problems associated with mainframe-based database system design. Mainframe based systems are limited greatly in terms of performance, and are costly to replace. MBDS solves these problems by moving database functions to a separate dedicated mini or micro-computer based system. As seen in Figure 6, one computer, the controller, is used to interface to other computers or the user. Several other backend computers, with their own hard disk subsystems, do the dirty work of processing user queries, and are connected to the controller by a broadcast communications bus. Data from each user generated database is distributed over the backends in clusters so that each backend has approximately the same amount of data. When the controller receives a transaction to

work on, it transmits the query to all the backends. The backends search for the query results in parallel, realizing performance gains proportional to the number of backends being used.

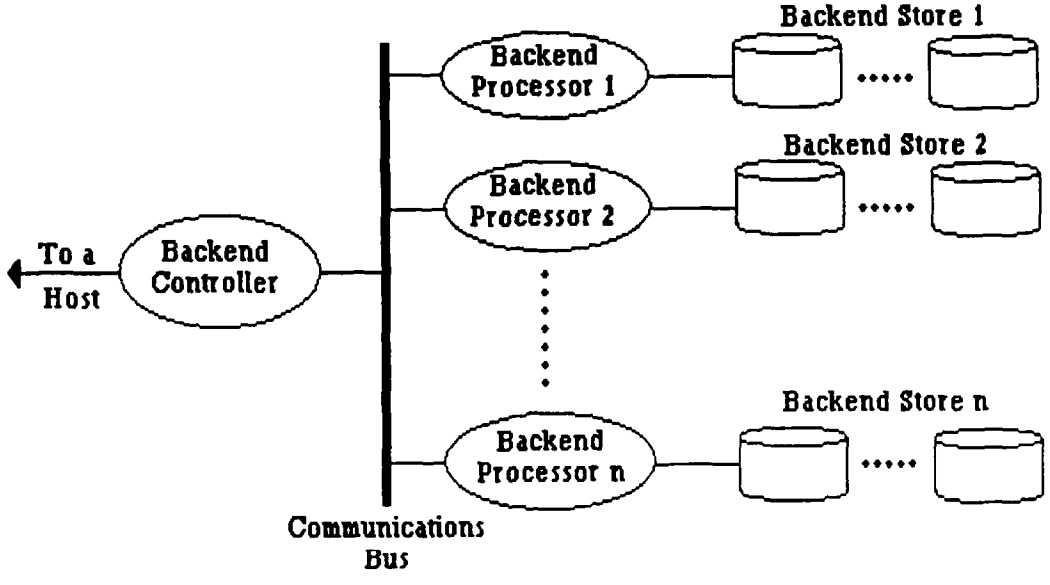


Figure 6. The Multi-Backend Database System

The MBDS system offers a high degree of extensibility, allowing the system to be upgraded with

- no modification to existing software
- no additional programming
- no modification to existing hardware
- no major disruption of system activity when additional hardware is being incorporated into the existing hardware. (Hsiao, 1983, p. 4)

Performance gains are easy to achieve by increasing the number of backends, which will produce a nearly proportional decrease in response time for a constant size database. As database size grows, adding backends results in nearly invariant response times.

4. Attribute-Based Data Model

As stated before, the Multi-Lingual, Multi-Model, Multi-Backend Database System (heretofore referred to as MBDS) requires the use of a kernel data model (KDM) and kernel data language (KDL) into which all databases are translated. The MBDS system uses the **Attribute-Based Data Model (ABDM) and Language (ABDL).**

a. Model Description

Any database consists of a collection of files, each of which consists of a number of records. In the ABDM each record consists of *attribute-value pairs* which are members of the Cartesian product of the set of attributes in the database, and the values in the domain set of these attributes. As an example, <NAME, Hogan> is an attribute-value pair in which Hogan is the value for the name attribute. Some records also contain an optional textual field which is referred to as the record body. A typical ABDM record is shown below.

(<FILE, Ships>, <NAME, Nimitz>, <TYPE, Carrier>)

The first attribute-value pair identifies the file which contains the records, and is followed by the rest of the attribute-value pairs which make up the record.

Keyword predicates, or simply *predicates* are used to search for specific records in a database. Predicates consist of an attribute name, relational operator, and attribute value. A typical query will be in disjunctive normal form such as

(FILE = Ships and TYPE = Carrier) or

(FILE = Ships and TYPE = Battleship)

which will locate all the records in the Ships file which have a value of either Carrier or Battleship for the TYPE attribute. (Hsiao, 1989, pp. 20-22)

b. The Attribute-Based Data Language (ABDL)

All database queries, whether they are in SQL, DL/I, or any other supported data manipulation language, are ultimately translated into ABDL which consists of the five basic operations insert, delete, update, retrieve and retrieve-common.

INSERT requests that a record be inserted into a previously defined database. A typical INSERT consists of all the attribute-value pairs necessary to fill a database record. A typical example would be

INSERT (<FILE, Ships>, <NAME, Saratoga>, <TYPE, Carrier>)

which inserts a record into the Ships file with the given name and type attribute values.

The DELETE request deletes one or more records from a previously defined database. As an example, the following query will remove all records in the Ships file in which the ship type is a destroyer.

DELETE ((FILE = Ships) and (TYPE = Destroyer))

The UPDATE request updates one or more records from a previously defined database. The request consists of the query to find the records to update, and the manner in which the records which satisfy the query should be modified. For example, the following UPDATE request

UPDATE ((FILE = Aircraft) and (TYPE = P3)) (WEIGHT = WEIGHT + 500)

will increment the weight of all P3s in the Aircraft file by 500.

The RETRIEVE request is used to retrieve records or specific attribute values of a previously defined database. Each RETRIEVE request consists of a query followed by the names of the attributes whose values you would like returned. The following query

RETRIEVE ((FILE = Ships) and (TYPE = Carrier)) (NAME)

will display the names of all ships of type carrier in the Ships file. It is also possible to use aggregate operations such as SUM, AVG, MIN, MAX, or to order the output by using the BY clause as in the following example

```
RETRIEVE (FILE = Aircraft) (TYPE) BY WEIGHT
```

which will display the types of aircraft in the Aircraft file in order of increasing weight.

Lastly, we have the RETRIEVE-COMMON request which merges two files who share common attribute values. The query

```
RETRIEVE (FILE = Naval_Aviators) (NAME)
COMMON (Aircraft_Type, Type_of_Aircraft)
RETRIEVE (FILE = Marine_Aviators) (NICKNAME)
```

will display the name of Naval aviators and the nickname of Marine aviators who fly the same type of aircraft. Note that the attribute names in the COMMON clause are not required to be the same.

The five operations presented for the Attribute-Based Data Language are extremely simple, and yet they are powerful enough to accomplish complex needs for update or retrieval of data. Therefore, any query that can be formulated in other data manipulation languages can be translated to corresponding ABDL queries. (Hsiao, 1989, pp. 32-35)

D. THE SOCKET ABSTRACTION

Network I/O in 4.3 BSD UNIX is accomplished by an abstraction know as the *socket.* A socket is a generalization of the UNIX file access mechanism that provides an endpoint for communication (Comer, 1988, p. 265). When inter or intra-network communication is desired the application program makes a request for the opening of a socket and all communication takes place through that socket. Although the procedure may vary slightly depending on the software used, (as it does on the PC side of our ethernet

connection) the following general procedures can be used to set up separate receiving and sending sockets. We will only describe socket operations that apply to this thesis specifically. For further information on sockets the reader is directed to the text by Comer.

1. Receiving Socket

The socket must be created using the socket system call as in the example below

```
sd1 = socket(AF_INET, SOCK_STREAM, 0)
```

which specifies that a socket of the DARPA Internet protocol family should be created with reliable stream delivery service. The socket call returns a small integer that will be used to reference the socket. The next step is to use the bind system call to bind the socket to a local address and port.

```
bind(sd1, &our_recv_addr, sizeof(our_recv_addr)
```

In this case we have bound our previously created socket to a port which has been specified in the structure our_recv_addr (of type sockaddr_in) containing the port number, address and protocol family. Next, we must use the listen system call to enable enqueing of requests for connection. A typical listen call is

```
listen(sd1, 5)
```

which puts the socket in a passive mode and specifies that it should listen for requests and enqueue up to five simultaneous requests. Any requests exceeding the maximum specified queue length will be discarded. Once we receive a request for connection, it must be accepted by an accept system call similar to

```
sd2 = accept(sd1, &their_addr, &len)
```

which blocks program execution while waiting for a connection. It creates a new socket and fills in the appropriate data in the structure their_recv_addr (also of type

sockaddr_in) based on the connection request. This socket is now ready to read data from the sending socket through a system call of the form

```
read(sd2, buffer, bytes_to_read)
```

which reads "bytes_to_read" bytes from socket sd2 and stores them in a buffer pointed to by the character pointer "buffer". If any of the above calls fail, they will return values that will flag the error.

2. Sending Socket

To set up our sending socket we make a socket system call as we did for the receiving socket saving the socket descriptor in the variable sd3.

```
sd3 = socket (AF_INET, SOCK_STREAM, 0)
```

Next, we attempt to connect the socket to a specified receiver as in

```
connect(sd3, &their_recv_addr, sizeof their_recv_addr)
```

where sd3 is the previously created socket's descriptor, and their_recv_addr is another structure of type sockaddr_in which specifies the receiver's address and port number. The socket is now ready to send data by using the write system call as in the example below.

```
write(sd3, buffer, bytes_to_write)
```

This will write "bytes_to_write" bytes from the buffer pointed to by "buffer" through the socket specified by sd3.

III. IMPLEMENTATION

A. MULTI-BACKEND DATABASE SYSTEM SERVER

The task of connecting GLAD to MBDS involved the integration of three distinct programming tasks, in three distinct programming environments. The first task to be implemented was to create the "server" version of the MBDS system which would allow the GLAD system to remotely interact with MBDS. This server is not a true language interface as described in Chapter II, since the GLAD object-oriented query language has not yet been implemented, and therefore could not be mapped into the kernel data language of MBDS. Instead, a modification of the ABDL interface to the single-user version of MBDS was implemented to give GLAD the capability to query MBDS databases, and to set the groundwork for future enhancements to GLAD which will be discussed in Chapter IV.

1. The MBDS Socket Interface

The first programming environment used was UNIX C, and the first step was to establish a means of communication between MBDS and GLAD. Since the PC and GLAD are connected by an Ethernet network, the natural choice for communications was through the use of the socket abstraction described in Chapter II. The implementation of the socket interface brought up several questions. How many sockets were necessary? Where in the MBDS system should the sockets be established?

Although two sockets which are connected to each other can form a full-duplex data stream, we chose to use separate sending and receiving sockets on both the GLAD and MBDS sides of the network. This choice was made to enhance the reliability of the system and to aid in debugging of the system. As Figure 7 shows, all commands and

queries from GLAD are sent via DDE messages to the GLAD socket interface, which transmits them through its sending socket to MBDS's receiving socket. Conversely, all MBDS data and error messages are sent through MBDS's sending socket to the GLAD socket interfaces' receiving socket, and are then sent via DDE messages to GLAD. Stream sockets were chosen to ensure sequenced, reliable connections rather than the less reliable datagram sockets.

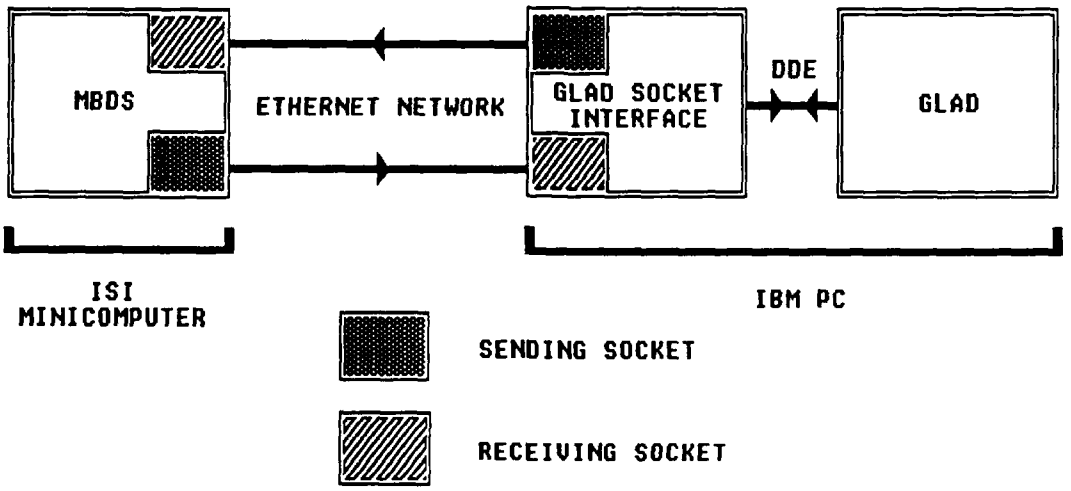


Figure 7. Overview of the GLAD/MBDS Interface

To ensure that the sockets would remain in service until exiting the MBDS system, they were created in the **Test Interface (TI)** portion of MBDS. TI is the user-interface for MBDS and presents the user with the following top-level menu upon starting MBDS.

The Multi-Lingual/Multi-Backend Database System

Select an operation:
- (a) - Execute the attribute-based/ABDL interface
- (r) - Execute the relational/SQL interface
- (h) - Execute the hierarchical/DL/I interface
- (n) - Execute the network/CODASYL interface
- (f) - Execute the functional/DAPLEX interface

(g) - Execute the object-oriented/GLAD interface
(x) - Exit to the operating system

Select-> _

To begin interfacing with the GLAD system, the user selects operation (g) which initiates the actions required to set up the socket interface. Note that this menu is presented on the ISI minicomputer, and must be initiated before attempting to open any databases from the GLAD system. The ability to place MBDS in the server mode from GLAD has not yet been implemented.

The receiving socket is set up first, a process which, using UNIX, is much more involved than setting up a sending socket. We were immediately faced with a problem which inhibited us from making the connection with the PC. UNIX provides the functions *gethostbyname* and *gethostbyaddr*, which normally take a host name or address, and return a pointer to a structure containing information for that host. This information is necessary to make a proper connection to the host. These two functions had been modified within MBDS and returned incorrect information when used as the UNIX manual describes. This change was left undocumented, and the original implementor had since left, making debugging the problem very difficult. It was finally decided to "hardwire" the host information into the server code. Although this is obviously not an ideal solution, we felt it was adequate given the experimental nature of the system. The code documentation includes code for a system in which the gethostbyname function works properly.

After the receiving socket is established, the sending socket is created and the MBDS system enters its server mode, awaiting messages from GLAD. If an error occurs in reading from a socket, MBDS sends an error message to GLAD, exits the server mode and the user is presented with the opening menu as previously discussed. Messages are

sent over the network by first sending the message length. Storage is dynamically allocated for the incoming message based on the message length. The first three bytes of any message is the message type. Currently, the three types of messages MBDS can receive are *open database*, *query database*, and *terminate session*. If an invalid message is received, an appropriate error message is sent to GLAD.

a. Open Database Option

The *open database* option allows GLAD users to prepare an MBDS database for use. Below is a typical open database message.

010SALES

The 010 is the code used to signify that this is an "open database" message and SALES is the name of the database to be opened. As an added safety feature, MBDS immediately checks to see if the database has already been opened even though GLAD will not let such a situation occur. If the database is already open, nothing more is done, and MBDS continues to wait for incoming messages. If the database has not been opened, a check is made to ensure it exists. If it does, the appropriate *template* and *descriptor* files are loaded. Template and descriptor files are required for all MBDS databases and contain the database's schema. Next, the file containing the database records is loaded. Although MBDS allows the user to have any number of record files for a particular database schema, the server code assumes that the records are located in a file using the database name with a ".r" extension. Since most databases in MBDS have only one record file, (which is normally named as discussed above) this was determined to be a viable solution. If the user desires to use different record files, he can copy the template and descriptor files (which are generally very small files) and give them another name corresponding to the desired record file. After the records have been loaded, the user and database IDs are broadcast to the 12 processes which make up the MBDS system. If any

one of the above steps fails, an appropriate error message is sent over the network to GLAD. At this point the database is ready for use and the MBDS is ready to process other incoming messages.

b. Query Database Option

The *query database* option allows the GLAD user to query any previously opened MBDS database. Below is a typical "query database" message.

020SALES@[RETRIEVE((FILE=Customer)and(CUSTID=C11))(NAME)]

The 020 is the code used to signify that this is a query. An "@" symbol is used to delimit the end of the database name, which in this particular query is SALES. Anything after the "@" is considered to be the query. MBDS parses the message and determines first if the requested database has been loaded. If not, an error message is generated and sent to GLAD. If the database has been loaded but is not the current MBDS database, a message is broadcast to the other MBDS processes making it the current database. The query is then sent to *Request Preparation*, one of the MBDS processes which handles the parsing and execution of the query. After the query has been processed, the results are gathered and translated to the GLAD's native data format, which consists of an "&" character following each attribute, and a carriage return and line feed (CR-LF) at the end of each record. MBDS uses an "@" symbol rather than the CR-LF at the end of each record in order to enable the results to be transmitted through the socket in one continuous stream. GLAD's socket interface saves these results in a text file, replacing the "@" symbols with CR-LF's. MBDS has a restriction on the length of the query results responses, breaking larger responses into more manageable "chunks" and placing an "end of results" marker at the end of the final chunk. These "chunks" are sent back to GLAD, followed by a special message indicating that all query results have been transmitted.

If a query will not parse correctly or some other error occurs, an appropriate error message is passed back to GLAD. In the case of UPDATE and DELETE queries where no real results are returned, MBDS returns the characters "&@." The "@" will be replaced with a CR-LF by the PC's socket interface prior to saving the results in a text file, and the "&" is the signal to GLAD that the update or deletion was successful. Results from aggregate operations such as MIN, MAX or SUM will be sent with the result of the aggregate operation followed by a "&@", with the "@" again being replaced by a CR-LF by GLAD's socket interface. After all query results have been transmitted, MBDS awaits the next message from GLAD.

c. Terminate Session Option

If GLAD wishes to terminate the session, a message length of zero is transmitted. Upon receiving this terminate session message, MBDS leaves the server mode, closes its sockets, and returns to the top-level menu, allowing the user to reinitiate a session or select another MBDS option. If GLAD's socket interface is purposely or inadvertently shutdown, a terminate session message is automatically sent to MBDS to ensure that the system will not lock up waiting for a message that will never come. This was accomplished by properly handling the messages that Windows sends to an application before shutting it down. There are two ways to shut down an application in Windows; by closing the application window itself, or shutting down the Windows system. In the first case, Windows sends the application a WM_CLOSE message before shutting it down, and in the second case it sends a WM_QUERYENDSESSION message. This gives the application an opportunity to tie up any loose ends before being shut down. In both cases, the socket interface first sends a *terminate session* message to MBDS, and then closes down the sending and receiving sockets.

B. THE GLAD SOCKET INTERFACE

1. Why a Separate Socket Interface?

The design of the socket interface proved to be one of the most interesting and difficult challenges faced in the implementation of this thesis. Ideally, we would make all calls to socket functions (i.e., read, write, etc.) from within the GLAD/Actor environment. The problem which arose was that making a call to a function which is not defined within the Actor environment requires that the function be placed in a Windows **Dynamic Linking Library (DLL)**. Windows DLLs are stored in a special executable (.EXE) format which is unlike normal libraries. Normal libraries contain code which is linked into a program at compile time. With DLLs however, the calling program's references to DLL routines are left unresolved until a call to a function within the DLL is made. At that time the connection is made dynamically between the calling program and the library routine, allowing many programs to share the same library code, greatly reducing the amount of memory required. Unfortunately, the network software that we had at our disposal, "The LAN WorkPlace" by Excelan, did not include a dynamic linking library of socket routines.

An attempt was made to create a DLL of socket functions, written in C, that made calls to the functions available within the Excelan socket library. The results were mixed, with some functions working correctly and others returning unexpected results. After some calls to Microsoft, Excelan and some further reading, the problem was discovered. The Intel 8086-family of microprocessors divide memory into 64K segments and programs reference these segments through the use of four segment registers. These registers are the code segment register (CS), the data segment register (DS), the stack segment register (SS), and the extra segment register (ES). In C, all variables defined outside of functions or declared as static are stored in static memory within the data

segment referenced by the DS register. All function parameters and variables within functions which are not declared as static variables are stored in the stack segment referenced by the SS. When a reference is made to a variable within a C program, no differentiation is made between pointers to stack variables and pointers to static variables. To put it simply, the DS is equivalent to the SS. This is where the problem occurs. When using Windows' DLLs, the data segment used is the library's own data segment, however the calling program's stack segment is used. Therefore, DS is not equivalent to SS and functions which assume that DS==SS will not work properly. This is the phenomenon which caused many of the functions within our DLL to fail. (Petzold, 1988, pp. 805-825)

The solution to this problem required us to implement a separate and distinct socket interface program written in C. Since we could use the Excelan socket library with a C program as long as it was not a DLL, we could make all the required calls to the socket functions. The socket interface was developed as a Windows application and provided us with yet another interesting problem. How could the socket interface communicate with the GLAD system? Thankfully, this was a much easier problem to solve since Windows provides the Dynamic Data Exchange protocol to enable Windows applications to communicate with each other. The socket interface basically acts as a communications relay between GLAD and MBDS. The interface normally runs as an icon in the shape of a wall socket, however by defining the debug_Flag in the sockets.h header file, it can be run as a window which will provide debugging information regarding the messages that is being processed.

2. Comparison of MBDS and GLAD Socket Interfaces

Although the GLAD Socket Interface is very similar to the one implemented in MBDS, the message-driven Windows environment required some unique twists to the

design. Separate sending and receiving sockets were set up to correspond with the MBDS sockets. The creation of these sockets using the Excelan software proved much less difficult than their UNIX counterparts. Since the GLAD socket interface would be receiving messages from two different sources, as opposed to one for the MBDS sockets, it was necessary to use sockets with synchronous non-blocking I/O. Normally, when a socket is read from, program execution will halt until a message is received by the socket. Since any Windows program is effectively a loop which interprets and translates messages, if execution got hung up waiting for information from MBDS, (which might not be sending any information) no messages from GLAD could be processed. This would effectively lock up the whole system. Synchronous non-blocking I/O allows a program to check to see if there is any data available from the socket, and if not, immediately return from the socket call and continue execution at the next statement following the call. In this case, the socket function returns a value of zero indicating that the data was not available.

The main portion of the socket interface program is a loop which gets messages, translates them, dispatches them to the proper handler routine, and then checks the socket for data from MBDS. If no Windows messages are received, the execution effectively halts at the "get message" portion of the loop and the socket will never be checked for data. If GLAD is awaiting data from MBDS, this could also cause a situation which would effectively lock up the system. A way was needed to ensure that some kind of message would be sent to the socket at regular intervals so that the socket would be checked for data regularly. Windows provides the WM_TIMER message for just such a situation. The Windows Software Development Kit provides a SetTimer function which will send WM_TIMER messages to an application at any user-defined interval. This routine was used in the socket interface to ensure such a lock-up would not occur.

3. The Socket Interface and Dynamic Data Exchange

As discussed in Chapter II, GLAD sends messages with MBDS requests to the socket interface using Windows' Dynamic Data Exchange protocol. The socket interface receives these messages in the form of WM_DDE_REQUEST messages in which the string containing GLAD's request is referenced by an atom that is passed in the message. The message is simply forwarded to MBDS without any local error checking, by sending the message length, followed by the message itself. If an error occurs in MBDS while translating or processing a request from GLAD, MBDS generates the error message and passes it back to the socket interface. If any errors occur in the creation or use of the sockets, memory allocation, or file I/O within the socket interface, an error message is generated by the socket interface. The socket interface passes error messages using a WM_DDE_DATA message in which the atom in the message references the string containing the error message. If MBDS dies for any reason, this will be detected by the socket interface, and an appropriate message will be sent to GLAD.

When MBDS returns results from a query, they are stored in a text file named "qresults.fil" on the PC. The "@" character at the end of each record returned by MBDS is replaced by a CR-LF combination prior to storing them in the text file. Each record appears on a different line of the text file which is stored in GLAD's native data format. As soon as all the results of the query are gathered in the file, a WM_DDE_DATA message is sent to indicate that the data is available. Currently, only one query can be processed at once, so using the same file name for storing the results does not present any conflict. GLAD renames the file to a unique name designated in the schema file after receiving the "end of results" message.

C. GRAPHICS LANGUAGE FOR DATABASE

1. Background

Although many excellent query languages have been developed and implemented in recent years, they all suffer from one basic fault. None of these languages are truly easy for the typical database user to learn or use . Ideally a system should be intuitive enough for the novice end-user to be able to perform the majority of database operations without reference to documentation or on-line help. We believe the key to such a system is the graphical user interface implemented in **Graphics Language for Database (GLAD)**. GLAD was developed by Professor C. Thomas Wu at the Naval Postgraduate School in Monterey, California. GLAD will provide a consistent and coherent interface for data manipulation and program development, regardless of the type of database system (e.g., hierarchical, network, or relational) being used. The key to this system is a simple visual representation of the database schema (Wu, 1988, pg. 2). Currently a prototype system has been developed in which the data definition, data manipulation, and on-line help system have been implemented, including the ability to store and manipulate graphic images as a part of the databases.

As discussed in Chapter II, the environment chosen to implement GLAD was Microsoft Windows utilizing the Actor object-oriented programming language. This environment was ideal for the rapid prototyping desired for the GLAD system since Actor provides and abundance of predefined objects which could easily be modified to meet our requirements. The Windows environment also provided the consistent and coherent user interface that we desired.

2. Hardware and Software Requirements

The GLAD system, including the network support implemented in this thesis (assuming one has access to an Ethernet computer network), has the following requirements:

- IBM compatible computer (80286 or higher)
- Minimum of 640 Kbytes memory (One to three megabytes recommended)
- Hard disk
- Graphics display and adapter
- Mouse (or other pointing device)
- Excelan EXOS 205T Model 4 Intelligent Ethernet Controller Board
- Excelan LAN WorkPlace Network Software for PC DOS TCP/IP Transport System
- Excelan LAN WorkPlace Network Software for PC DOS Socket Library Application Program Interface
- Microsoft Windows
- MS-DOS version 2.0 or higher

The Microsoft Windows environment is a very memory and CPU intensive environment, so hard disk access time, amount of random access memory, and CPU speed are important factors in providing satisfactory user response times.

3. Comparison of the MBDS and GLAD User Interfaces

Perhaps the most graphic way to show the benefits of the GLAD interface over the present MBDS interface is to run through a typical, multi-database session using the current MBDS system, and follow it by the same session using GLAD. One of the biggest problems with the MBDS interface is that new or casual users find it very

difficult to navigate through layer upon layer of an often cryptic menu structure. In many cases, the same prompt is given to obtain the name of a file, when in each case completely different files are being requested. The GLAD system shields the user from these confusing choices and makes the system a pleasure to use, even for first time users. We will only describe portions of the GLAD system which were changed by the work done on this thesis.

a. The MBDS User Interface

The opening MBDS screen allows the user to select the language interface that will be used during that particular portion of the session.

```
        The Multi-Lingual/Multi-Backend Database System

Select an operation:
    (a) - Execute the attribute-based/ABDL interface
    (r) - Execute the relational/SQL interface
    (h) - Execute the hierarchical/DL/I interface
    (n) - Execute the network/CODASYL interface
    (f) - Execute the functional/DAPLEX interface
    (g) - Execute the object-oriented/GLAD interface
    (x) - Exit to the operating system

Select-> _
```

The user selects the desired operation, in this case (a) for the attribute-based/ABDL interface. Next, the user is confronted with a menu which offers the following options for generating, loading or requesting interface with (querying) a database.

```
The attribute-based/ABDL interface:

    (g) - Generate a database
    (l) - Load a database
    (r) - Request interface
    (x) - Exit to the previous menu

Select-> _
```

The user would now select option (l) to load a new database and the system will then prompt the user with a similar menu.

Select an operation:

(u) - Use a database
(r) - Mass load a file of records
(x) - Exit, return to previous menu

Select-> _

The user should select the (u) option to load the template and descriptor files of the database. The user then will be prompted for the name of the database, which corresponds to the name of the template and descriptor files for the desired database.

Enter the name of the database:

For this sample session we will assume the user selected the FIRST database. In this case the ".t" and ".d" extensions of the template and descriptor files should not be included in the database name. After the template and descriptor files are loaded, the user is again presented with the following menu:

Select an operation:

(u) - Use a database
(r) - Mass load a file of records
(x) - Exit, return to previous menu

Select-> _

This time the user selects the (r) option to load the database records from a file. The user will be prompted for the name of the record file which is not required to be the same as the template and descriptor files.

Enter the record file name: _

In this case, the user must enter the record file name, including the ".r" extension. After the records have been loaded, the user will again be presented with the same menu.

Select an operation:

(u) - Use a database
(r) - Mass load a file of records
(x) - Exit, return to previous menu

Select-> _

This time the user must select (x) to return to the previous menu in order to query the database, a selection which is not in the least bit intuitive.

The attribute-based/ABDL interface:

(g) - Generate a database
(l) - Load a database
(r) - Request interface
(x) - Exit to the previous menu

Select-> _

This time the user should choose the (r) option to query the database. This menu selection brings the user to yet another menu.

Select a subsession:
(s) SELECT: select traffic units from an existing list
(or give new traffic units) for execution
(n) NEW LIST: create a new list of traffic units
(d) NEW DATABASE: choose a new database
(p) * PERFORMANCE TESTING
(r) * REDIRECT OUTPUT: select output for answers
(m) * MODIFY: modify an existing list of traffic units
(o) * OLD LIST: execute all the traffic units in an
existing list
(x) EXIT: return to previous menu

Refer to the MLDS/MBDS user manual before choosing
subsessions marked with an asterisk (*)

Select-> _

To query the database, assuming a file has not previously been created containing the required queries, the user must create a request file using the (n) option,

even if he only has one query. The process of creating a query is quite lengthy, and the ability to simply type in a query at the command line does not exist. The following lines show the complex set of interactions required simply to display all the members of the Employee relation. User input is in boldface italics.

Enter the name for the traffic unit file
It may be up to 40 characters long including the .ext.
Filenames may include only one '#' character
as the first character before the version number.
FILE NAME-> ***FIRST#3***

Enter the character for the desired Traffic Unit type.

(r) Request
(t) Transaction (multiple requests)
(f) Finished entering traffic units.

Select-> ***r***

Enter the character for the desired next step.

(i) INSERT
(r) RETRIEVE
(u) UPDATE
(d) DELETE
(c) RETRIEVE COMMON

Select-> ***r***

RETRIEVE Request
Enter responses as you are prompted. You will be prompted first for the predicates of the query, then attributes for the target-list, next for an attribute for the optional BY clause and finally for a pointer for the optional WITH clause.
When you have finished entering predicates for the query, respond to the ATTRIBUTE> prompt with a <return>.

ATTRIBUTE (<cr> to finish)-> ***TEMP***

Enter the character for the desired relational operator

(a) = EQUAL
(b) /= NOT EQUAL
(c) > GREATER THAN
(d) >= GREATER THAN or EQUAL

(e) < LESS THAN
(f) <= LESS THAN or EQUAL
Select-> ***a***

VALUE-> ***Employee***

So far your conjunction is
(TEMP=Employee).
Do you wish to 'and' additional predicates to this conjunction? (y/n) > ***n***
Do you wish to append more conjunctions to the query? (y/n) > ***n***

Begin entering attributes for the Target-List. When you are through entering attributes respond to the ATTRIBUTE> prompt with <return>.
Do you wish to be prompted for aggregation (Y/N)? ***n***

ATTRIBUTE (<cr> to finish)-> ***NAME***
ATTRIBUTE (<cr> to finish)-> ***AGE***
ATTRIBUTE (<cr> to finish)-> ***SALARY***
ATTRIBUTE (<cr> to finish)-> ***PHONE***
ATTRIBUTE (<cr> to finish)->

Do you wish to use a BY clause (Y/N)? ***y***

ATTRIBUTE (<cr> to finish)-> ***NAME***

Enter the character for the desired Traffic Unit type.

(r) Request
(t) Transaction (multiple requests)
(f) Finished entering traffic units.
Select-> ***f***

Select a subsession:
(s) SELECT: select traffic units from an existing list
(or give new traffic units) for execution
(n) NEW LIST: create a new list of traffic units
(d) NEW DATABASE: choose a new database
(p) * PERFORMANCE TESTING
(r) * REDIRECT OUTPUT: select output for answers
(m) * MODIFY: modify an existing list of traffic units
(o) * OLD LIST: execute all the traffic units in an
existing list
(x) EXIT: return to previous menu

Refer to the MLDS/MBDS user manual before choosing
subsessions marked with an asterisk (*)

Select-> ***s***

Will you use the current file (FIRST#3) (Y/N)? ***y***

List of executable traffic units:
(0) [RETRIEVE(TEMP=Employee)(NAME,AGE,SALARY,PHONE)BY
NAME]

Select Options:
(d) redisplay the traffic units in the list
(n) enter a new traffic unit to be executed
(num) execute the traffic unit at [num]
from the above list
(x) exit from this SELECT subsession
Option-> ***0***

At this point, all the members of the Employee relation will be displayed in the form

<Attribute Name, Attribute Value>,<Attribute Name, Attribute Value>...

Next, to load a new database, in this case the SALES database, we must backtrack through the menu structure to the following menu and repeat the entire database loading procedure for the new database.

The attribute-based/ABDL interface:

(g) - Generate a database
(l) - Load a database
(r) - Request interface
(x) - Exit to the previous menu

Select-> _

After database loading is complete, we must formulate another retrieve query in order to display members of any of the relations in the same complex manner as outlined above. To switch back to the database that we loaded first, we must again backtrack through the menu structure to the following menu and choose the use (u) option.

Select an operation:

(u) - Use a database
(r) - Mass load a file of records
(x) - Exit, return to previous menu

Select-> _

To query the database, we must again traverse the menu structure to the request interface portion of the system, and either formulate new queries, or use existing ones from a previously created file.

The preceding sample session shows only half of the interactions required to do something as simple as opening two databases and displaying the members of two of its relations. The menu structure is complex and often cryptic, and only the most experienced of users would have any chance of correctly using the system without extensive previous research. We will now show how the GLAD system solves these problems and allows even the first-time user to perform complex database operations with ease.

b. The GLAD User Interface

We will now present the same database interactions using the Graphics Language for Database's interconnection with the MBDS system. We will briefly mention the features available at each level of the interactions. Since we will mainly be describing the portions of GLAD changed by this thesis, we will leave quite a bit of the GLAD system unexplored. For a more detailed explanation of the GLAD system the reader is directed to the references by Wu, Fore and Williamson.

(1) GLAD Top-Level Window. Figure 8 shows the GLAD top-level window which is the gateway to all GLAD features. From this window the user can create a new database, or modify, open, or remove an existing database. Although many of the

features such as window resizing and menu selection are accessible through the keyboard, currently many portions of the program require the use of a mouse.

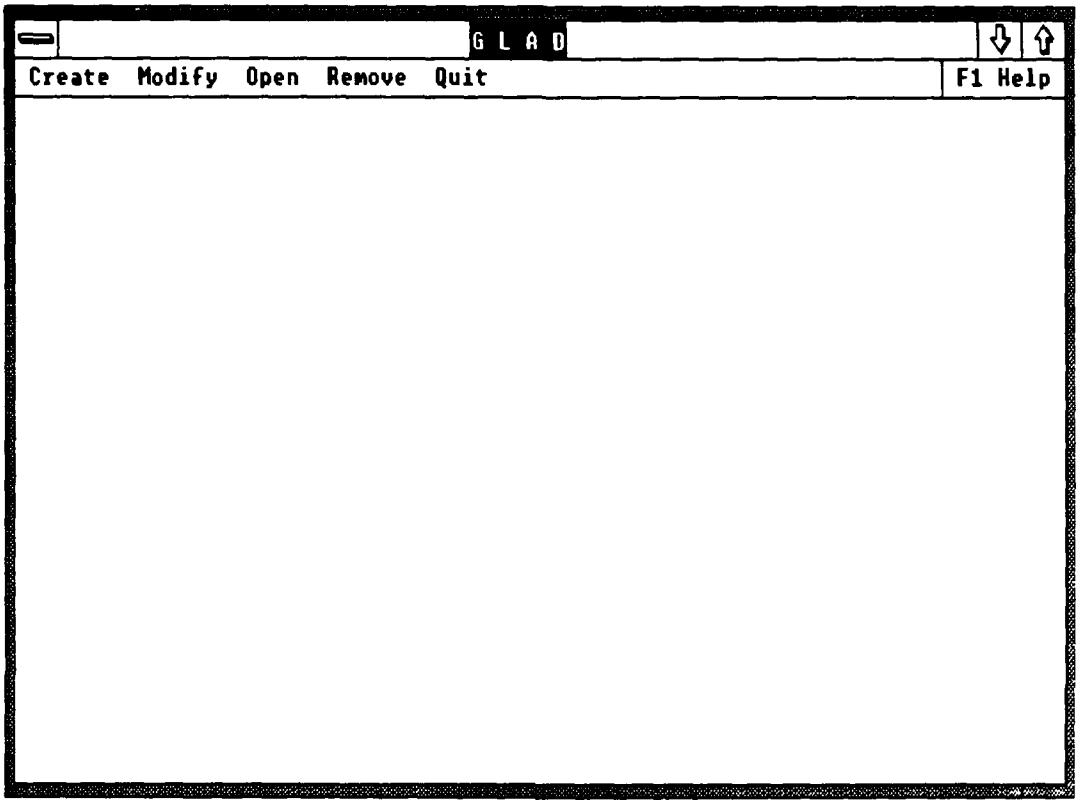


Figure 8. GLAD Top-Level Window

From the window's menu bar, the options available are

- **Create** - Allows the user to create a new database using GLAD's data definition facility.
- **Modify** - Allows the user to modify the schema of an existing database.
- **Open** - Enables the user to open a database for display and modification.
- **Remove** - Enables the user to remove an existing database from the system.

In future systems, only the Open option will be available to all users and an access control mechanism will be implemented to ensure only authorized users can gain access to possibly destructive database functions. Also available in every window, by clicking on **Help** with the mouse or selecting the F1 key, is a hypertext on-line help system. The system displays graphical as well as textual information regarding the GLAD system in popup windows. The **Quit** option is also available in all GLAD windows. The user can exit any window in the GLAD system by either selecting Quit, double clicking on the system box with the mouse, or single clicking on the system box and selecting **Close**.

In our particular case, we would like to open a database so we select the Open option and are presented with a dialog box as shown in Figure 9 which lists the databases to which we have access. The dialog box contains a scrollable list of databases since typically there will be more databases than can fit in the dialog box at any one time. The users options are presented in the form of buttons. To open a database, the user has two options. He may either double click on the desired database name, or single click on the database name, which will then be highlighted (selected), and then single click on the Open button. One of the most important features of GLAD is that it gives the user many ways to accomplish a given task, greatly improving the chance that he will remember at least one of them, and thereby increasing user productivity and satisfaction. We would like to open the MBDS database named FIRST so we select it and click on the Open button.

At this point, quite a bit goes on behind the scenes in the GLAD system that the user is unaware of, supporting the transparency of database selection that we desired. Selecting an MBDS database requires the exact same steps for the user as the opening of any other database in the GLAD system, however, the manner in which GLAD handles MBDS databases is an entirely different matter. When the user selects an MBDS

database, GLAD first checks to see if the GLAD socket interface is currently running. This would be obvious to the user (assuming none of the windows are covering the icon

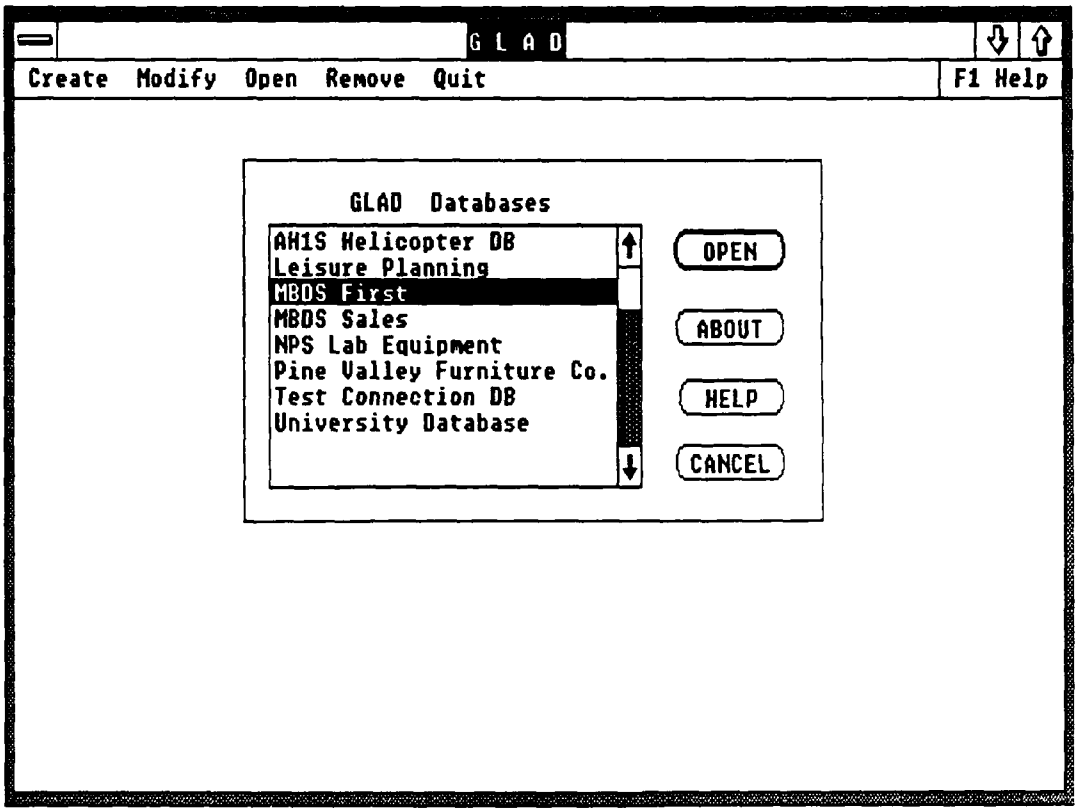


Figure 9. GLAD Top-Level Window with Database Selection Dialog Box

area of the screen) by the presence of the socket icon. If the socket interface has not been initiated, it will be executed, and the sockets will be set up between GLAD and MBDS as discussed previously in this chapter. Once GLAD has ensured that the socket interface is set up, it attempts to communicate with the interface by sending the WM_DDE_INITIATE message. Upon receiving a confirming acknowledgment from the socket interface, GLAD sends a request to MBDS via the socket interface to open the

database, in this case the FIRST database. GLAD then displays the data manipulation window for this database using a locally stored database schema file.

(2) Data Manipulation Window. Figure 10 shows the data manipulation window for the MBDS FIRST database with the socket interface running as an icon. As

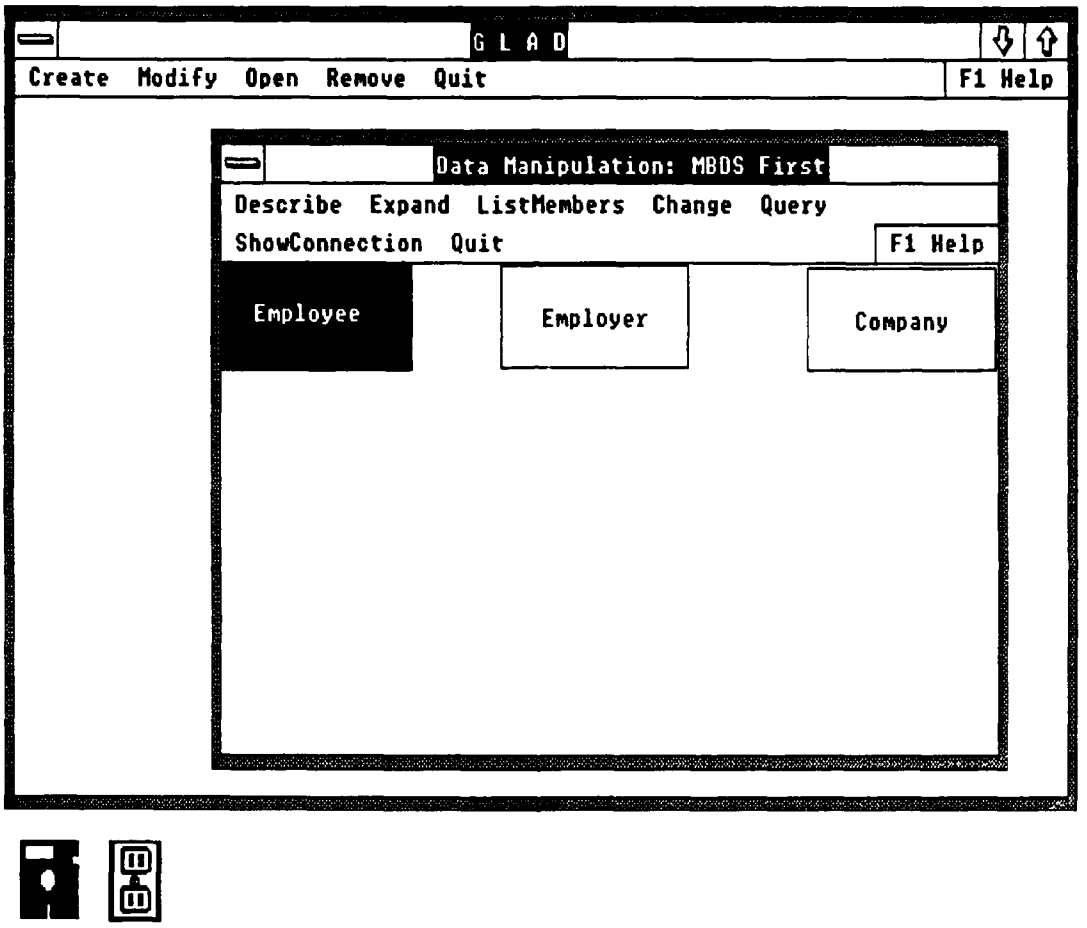


Figure 10. GLAD Data Manipulation Window and Socket Interface

before, the presentation of this window is no different from that of any other GLAD database. GLAD is based on an object-relationship model, and the objects (entities) of the database are represented by rectangular boxes with the name of the object in the center of the box. Each rectangle represents both the object type (i.e., database schema)

and a set of objects (i.e., database instance) currently in the database (Wu, 1988, pg. 5). Therefore, we can manipulate the objects both on the type level and the instance level. To select a particular object, the user clicks on the rectangle representing that object, and the color of the rectangle changes to confirm the selection. The options available in this window are

- **Describe** - This option displays the attributes of the selected object so the user can see the relationship between the selected object and other objects. GLAD allows both user and system defined attributes (See Figure 11).

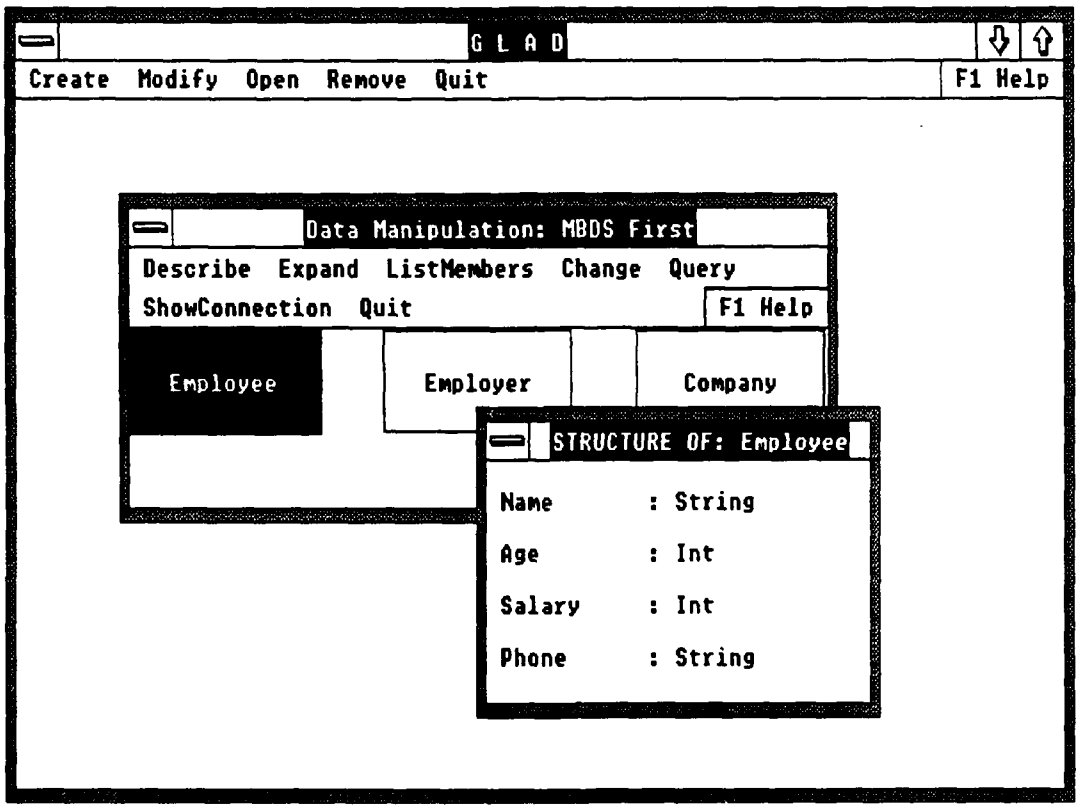


Figure 11. GLAD Describe Window

- **Expand** - This option displays the sub-classes of any object with a IS_A hierarchy (e.g., an Employee object might have have sub-classes of Faculty

and Staff). Objects that have sub-classes are displayed as nested rectangles in the data manipulation window.

- **ListMembers** - Allows the user to display and change the object's actual data using an all-at-once (browse) or one member at a time (display) presentation.

The Change and ShowConnection options are not implemented at this time, and the Query option is currently being implemented by other thesis students.

As before, we want to list all the employees, so we select the Employee object by clicking on it with the mouse. It changes to green, and from now on, all windows related to the Employee object will have green borders further enhancing the users awareness of where he is in the system. As mentioned before, there are two ways to look at an object, all at once or one member at a time. We will select the AllAtOnce menu option to view the entire database. When this option is selected, GLAD formulates an ABDL RETRIEVE request based on the locally stored schema. This request is sent to MBDS via DDE and the socket interface for processing. The results are returned in a similar manner, and stored in a file locally for display by GLAD.

(3) List Members Window. Figure 12 shows the List Members or Browse window which displays all the members of a database object. It uses scroll bars to enable the user to access portions of the database which will not fit in the window. To enable the user to view as much information as possible, only the first 20 characters of each field is displayed. If the user wants to view the complete information regarding any particular member he may select the **More** option after selecting a particular record by clicking on it, which will bring up the Display One window.

(4) Display One Window. Figure 13 shows the Display One window which enables the user to see all the information contained in a particular record. From this window, the user has the following options:

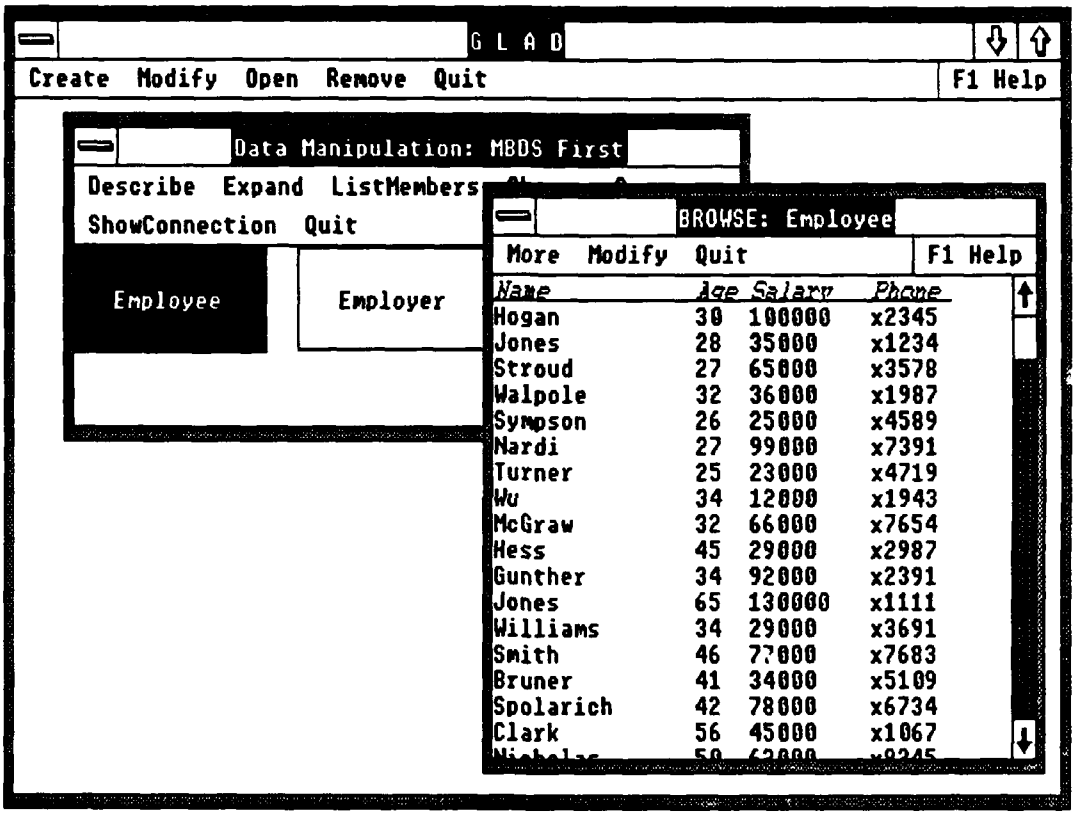


Figure 12. GLAD List Members Window

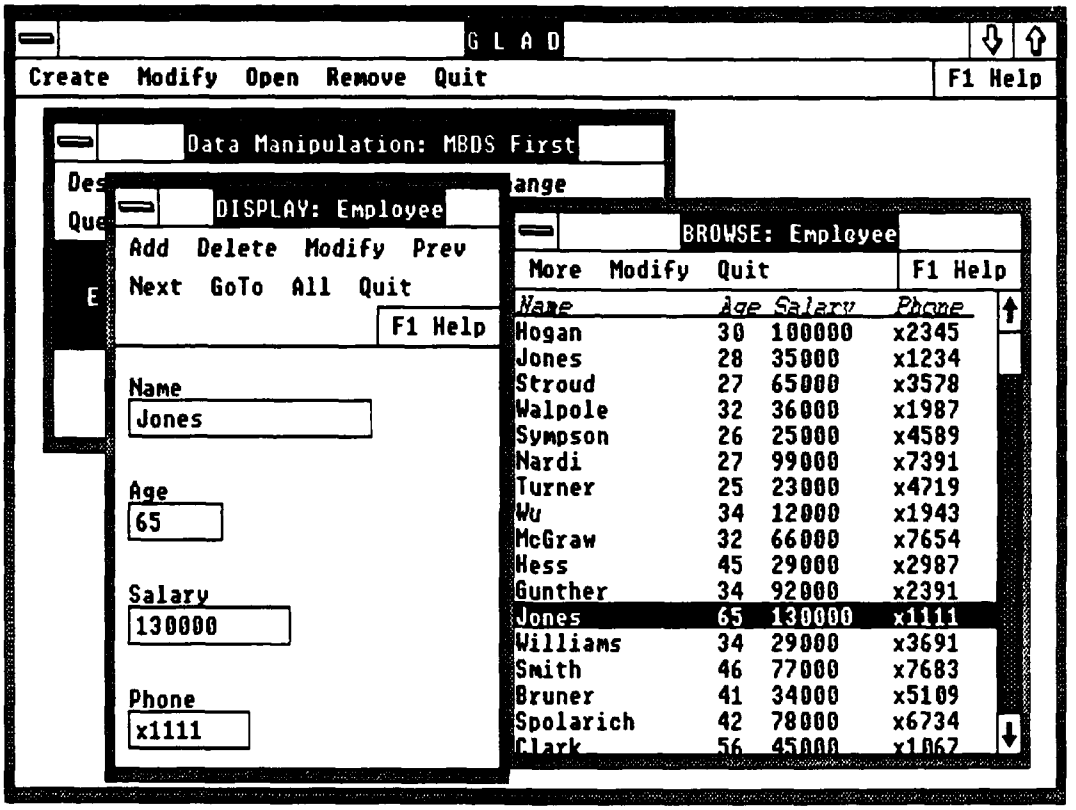


Figure 13. GLAD Display One Window

- **Add** - Add a new record to the members of the selected object.
- **Delete** - Delete the currently selected record.
- **Modify** - Modify the data contained in the currently selected record.
- **Prev and Next** - Switch to the record which precedes or follows this record in the database (if there is one). The Display and List Members windows are linked so that if both are on the screen at any one time, selecting a new record by clicking on it in the List Members window, or selecting the Prev or Next options in a Display One window, will also change the selected record in the other window.
- **GoTo** - Allows the user to view the first, last or Ith record, as selected by the user.
- **All** - Opens a List Members Window for the object if one has not already been opened. This is the same as selecting the AllAtOnce sub-option in the data manipulation window.

The ease of use of the GLAD system has allowed us to open the FIRST database and view the data with only five clicks of the mouse, and no file names or query language to remember or possibly mistype. When compared to the confusing and error-prone process normally required to open and view an MBDS database, the advantages of the GLAD system become readily, if not blatantly apparent. However, the convenience and ease of use of the GLAD system does not end here. GLAD allows you to view and manipulate several databases on-screen at the same time, switching between databases with ease. This is a feat which cannot be duplicated by MBDS. To open another database, in this case the MBDS database called SALES, we simply go back to the GLAD top-level window, and select the Open option. We are again presented with the dialog box containing the names of the databases available to us. By double-clicking on MBDS SALES we get another data manipulation window as shown in Figure 14. We can then select the ListMembers option to display the members of the Orders object of

this new database as shown in Figure 15. Any number of databases can be opened at any time, and the colored window borders make it easy to distinguish between the various

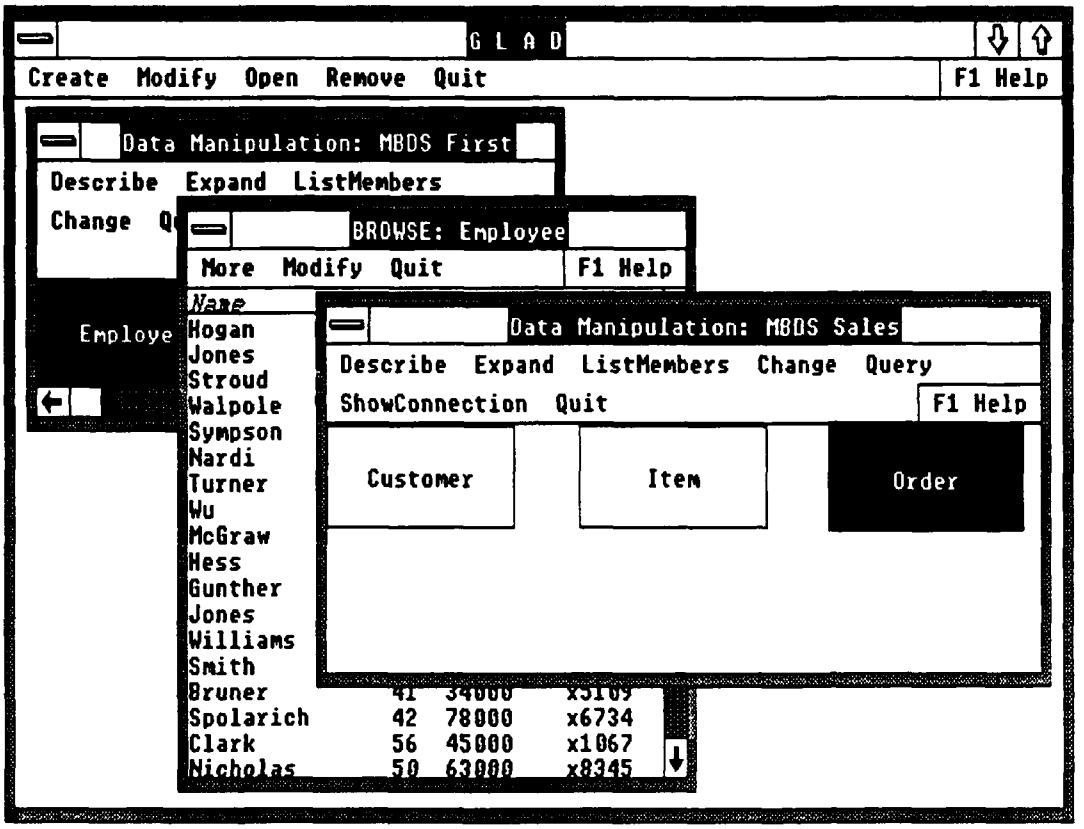


Figure 14. GLAD with Two MBDS Databases Open Simultaneously

database objects displayed. Switching between databases is as easy as clicking within the new database's window, and secting a new object to display. One look at the pages of interactions required to perform the above operations using the current MBDS interface clearly shows the simplicity and superiority of the GLAD interface.

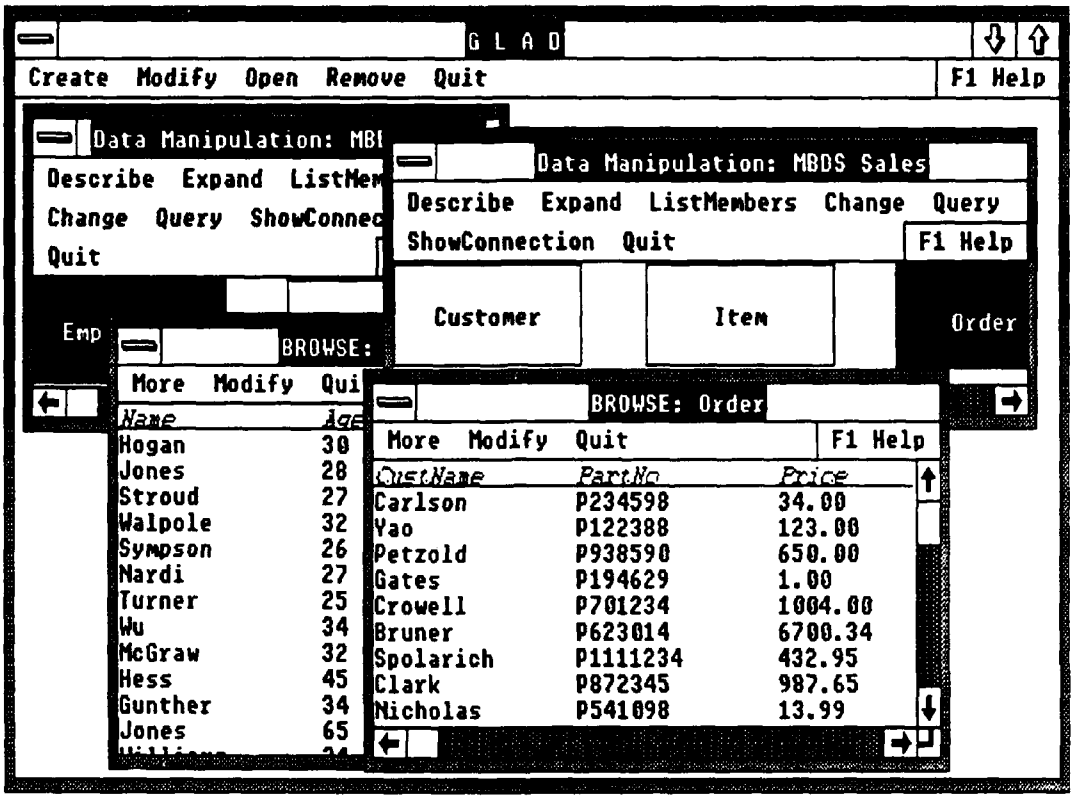


Figure 15. GLAD Displaying Information from Two MBDS Databases

IV. CONCLUSIONS

A. A REVIEW OF THE RESEARCH

The goal of the research documented in this thesis was to connect two database systems, running under different computer architectures and operating systems, to form a single coherent interface for all database interactions. We presented the strategy used, and the problems encountered in designing this system, including the reasoning behind the many difficult design decisions required.

In order to design the interconnection between GLAD and MBDS, the research took three distinct paths. Design of an MBDS Server and socket interface, design of a GLAD socket interface, and lastly the design of the GLAD system itself. The MBDS server required the implementation of a communications interface as well as new command driven version of the ABDL interface to MBDS. The GLAD socket interface provided the biggest challenge, requiring the coding of a stand-alone program that communicated with GLAD via Windows Dynamic Data Exchange. Due mainly to the ease of modification of object-oriented code, the changes to the GLAD system were the easiest to accomplish. Small changes were necessary to ensure the socket interface was activated for all MBDS databases. Facilities for communicating with the socket interface through the use of DDE were also implemented.

B. FUTURE ENHANCEMENTS

Since the GLAD system is still in its infancy, there are many possible enhancements to the present system which include, but are certainly not limited to the following:

- **Full Query Facility** - Although this thesis has given GLAD a limited ability to query MBDS databases, a much richer facility is required, and is presently being implemented. This includes the ability to query both MBDS databases

as well as GLAD's resident databases. Work is just beginning on implementing professor Wu's object-oriented query language which will later be fully integrated into the GLAD and MBDS systems.

- **Automatic Form Generation and Fill-in** - With the Navy's recent push towards the "paperless ship" concept, the need for the ability to generate and process the multitude of forms required to maintain a large navy becomes imperative. The graphical environment of GLAD makes adding such a facility to the system relatively easy.

- **Access Control** - To make GLAD a viable system in the eyes of the military and civilian sectors, it must have the ability to control the access of users to the system.

- **On-line Access to Maintenance and Reference Manuals** - One of the major strengths of computers is the ability to access and assimilate data from a variety of sources in quickly. Incorporation of a sophisticated hypertext facility within GLAD could help to expedite maintenance and administrative efforts by reducing the time wasted searching through unnecessary information. It would also aid in the paperless ship effort by eliminating the need for manuals, yet still providing a facility for hard-copy information when necessary.

- ***Incorporation of the Socket Interface Within* GLAD** - Although the present system works well, ideally we would like the socket interface to be an integral part of the GLAD system. The system is already plagued by memory constraints, and a separate interface does little to aid this problem. This can easily be accomplished when software becomes available whose socket functions are contained within a Dynamic Linking Library.

C. DISCUSSION AND BENEFITS OF THE RESEARCH

With the wide availability of computers to people of various skill levels and educational backgrounds, the need for a coherent and consistent user interface to all applications has become evident. This is particularly true in the area of databases, where the proliferation of data models and languages has caused much confusion in recent years. The GLAD project, and in particular this thesis, has shown that it is possible to provide a simple, yet sophisticated interface which can be used to interact with a database designed using any available model. The benefits of such a system to both the military

and civilian communities are obvious. The influx of computers into the Navy, to a relatively naive group of users, makes the need for such a simple and intuitive system almost imperative. The savings in training costs alone would make this system extremely attractive, but the ability to query diverse databases at remote locations displayed by this thesis is the key to future growth in the role of computers in the military.

Although a sister project named ARGOS is also being developed which is quite impressive, it unfortunately relies on Macintosh computers which are not readily available to the fleet today. GLAD is an attempt to mirror the progress made in the ARGOS project, and in this particular case we believe we have surpassed ARGOS, by providing a viable solution to the Navy's varied database needs using systems currently available in the fleet.

APPENDIX A. MBDS SERVER SOURCE CODE LISTING

The following listings are the MBDS code that was either created or modified for the implementation of this thesis.

A. TEST INTERFACE CODE (MODIFIED)

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <errno.h>
#include <ndbm.h>
#include <sys/time.h>
#include <fcntl.h>
#include "/usr/work/glad/VerE.6/CNTRL/TI/LangIF/include/glad_lil.h"

#include "flags.def"
#include "beno.def"
#include "msg.def"
#include "commdata.def"
#include "tstint.def"
#include "tstint.dcl"
#include "dblocal.def"
#include "tmpl.def"
#include "tmpl.dcl"

main(argc,argv)
int     argc;
char   *argv[];
{
  static char               answer;
  static int                StopSys = FALSE;
  char                      NOBE[2];
  int                       i, sd1, sd2, sd3, len;
  struct sockaddr_in        their_addr, their_recv_addr;
  struct sockaddr_in        our_recv_addr, our_send_addr;
  struct hostent            *hp, *gethostbyname();
```

```
/* initialize the test interface */
next_traf_id = 1;
initsr(TI);
reqs_left_count = 0;
/* determine output format: terminal and/or response file */
CRT_flg = TRUE;
f_flg = FALSE;

/* Determine the number of backends and transmit to all tasks. */
if (argc == 1)
{
  printf("Usage: ti.out #, where # is number of backends.\n");
  exit(0);
}
else
{
  strcpy(NOBE,argv[1]);
  if (NOBE[0] > '0' && NOBE[0] < '9')
  {
    NoBackends = str_to_num(NOBE);
    printf("\t\tNext stop... THE TWILIGHT ZONE\n");
    fflush(stdout);
  }
  else
  {
    printf("ERROR: Invalid number of backends arg on command line.\n");
    exit(0);
  }
}

/* send to all the appropriate tasks */
TI_S$SetNoBEs(NoBackends);

sleep(3);
system("clear");
printf("\n\n\n\n\n\n\n\n\n\n\n\n");
printf("\t\tSeconds remaining till psychotherapy: ");
for (i = 10; i > 0; i--)
{
    printf("%2d ", i);
    fflush(stdout);
    sleep(1);
    printf("\b\b\b");
```

```
}

Timer_on = 0;  /* init external timer flag to OFF */

/* Initialize the MLDS interfaces by loading the schemas */
creat_rel_db_list();
creat_net_db_list();
creat_hie_db_list();
/* creat_dap_db_list(); */

while (!StopSys)
{
  system("clear");
  printf("\tThe Multi-Lingual/Multi-Backend Database System\n\n");
  printf("Select an operation:\n\n");
  printf("\t(a)  -  Execute the attribute-based/ABDL interface\n");
  printf("\t(r)  -  Execute the relational/SQL interface\n");
  printf("\t(h)  -  Execute the hierarchical/DL/I interface\n");
  printf("\t(n)  -  Execute the network/CODASYL interface\n");
  printf("\t(f)  -  Execute the functional/DAPLEX interface\n");
  printf("\t(g)  -  Execute the object-oriented/GLAD interface\n");
  printf("\t(x)  -  Exit to the operating system\n");
  printf("\nSelect-> ");

  answer = getchar();
  if (answer != '\n')
    getchar();
  switch (answer)
  {
    case 'a': /*execute the attribute-based/ABDL interface*/
    case 'A':
      testint();
      break;

    case 'r': /*execute SQL interface*/
    case 'R':
      /* printf("\nThe SQL Interface is not operational.\n");*/
      sql_main();
      break;

    case 'h': /*execute DL/I interface*/
    case 'H':
      /* printf("\nThe DL/I Interface is not operational.\n");*/
      dli_main();
```

```
    break;

  case 'n': /*execute CODASYL interface*/
  case 'N':
    /* printf("\nThe CODASYL Interface is not operational.\n");*/
    dml_main();
    break;

  case 'f': /*execute DAPLEX interface*/
  case 'F':
    printf("\nThe DAPLEX Interface is not operational.\n");
    /*dap_main();*/
    break;

  case 'g':
  case 'G':

    /* Set up Receiving Socket */
    if ((sd1 = socket(AF_INET, SOCK_STREAM, 0)) < 0)
      perror("socket");

    our_recv_addr.sin_port = Glad_Rcv_Port;
    our_recv_addr.sin_family = AF_INET;
    our_recv_addr.sin_addr.s_addr = INADDR_ANY;

    if(bind(sd1, &our_recv_addr, sizeof(our_recv_addr), 0)<0)
      perror("bind");
    listen(sd1, 5);
    while (TRUE)
    {
      len=sizeof(their_addr);
      if((sd2 = accept(sd1, &their_addr, &len)) > 0)
        break;
      else
        perror("accept");
    }

    /* Set up Sending Socket */
    if ((sd3 = socket (AF_INET, SOCK_STREAM, 0)) < 0)
      perror("socket");
    /* Set up socket for synchronous non-blocking I/O */
    fcntl(sd3, F_SETFL, FNDELAY);

    their_recv_addr.sin_family = AF_INET;
```

```
        their_recv_addr.sin_port = Glad_Snd_Port;

        /* It was necessary to hardwire the PC307 address because        */
        /* the gethostbyname and gethostbyaddr functions have been       */
        /* altered within MBDS.  If this is fixed later, replace the     */
        /* following line with these two lines:                          */
        /*   hp = gethostbyname("pc307");                                */
        /*   bcopy(hp->h_addr,(char *)&their_recv_addr.sin_addr,hp->h_length); */

        their_recv_addr.sin_addr.s_addr = PC307_ADDR;

        if (connect(sd3, &their_recv_addr, sizeof their_recv_addr) < 0)
          perror("connect");

        glad_main(sd2,sd3);

        /* Close the sockets */
        close(sd1);
        close(sd2);
        close(sd3);
        break;

      case 'x': /* exit from MDBS */
      case 'X':
      case 'e':
      case 'E':
        StopSys = TRUE;
        TI_S$StopMsg();
        break;

      default:
        UserError(1);

    } /* end case */

  } /* end while */

  /* shutdown send/receive */
  finishsr(TI);

} /* end main */

TI_User(user_id)
  /* get the user id of this user - currently fixed */
```

```
  char  user_id[USERidLength + 1];
{
  strcpy(user_id,"user");
} /* end TI_user */
```

B. MBDS SOCKET INTERFACE AND SERVER

```
#include <stdio.h>
#include <ctype.h>
#include <glad.h>
#include <glad_lil.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <errno.h>
#include <ndbm.h>
#include <sys/time.h>
#include "/usr/work/glad/VerE.6/COMMON/commdata.def"
#include "flags.def"
#include "/usr/work/glad/VerE.6/CNTRL/TI/tstint.def"
#include "/usr/work/glad/VerE.6/CNTRL/TI/tstint.ext"
#include "/usr/work/glad/VerE.6/COMMON/msg.def"
#include "/usr/work/glad/VerE.6/COMMON/beno.dcl"
#include "/usr/work/glad/VerE.6/CNTRL/TI/dblocal.def"
#include "/usr/work/glad/VerE.6/COMMON/tmpl.def"

int     rcv_socket,          /* Socket ID for sending socket */
        snd_socket,          /* Socket ID for receiving socket */
        numDBs = 0;
extern char  current_dbid[DBIDLNTH + 1];
extern int   errno;
char  loadedDBs[MaxDBs][MFNLength + 1]; /* ID's of loaded databases */

/*****************************************************************************/
/*                                GLAD_MAIN                                  */
/*****************************************************************************/

glad_main(recv_socket, send_socket)
int     recv_socket, send_socket;
{
 char  user_id[USERidLength + 1],     /* ID of user, currently fixed to "user" */
       traf_unit[TULength],           /* Current database query */
```

```
        len_buf[MLBuffer + 1],      /* Buffer for length of incoming message */
        msg[MsgLength],             /* Incoming message */
        command_str[CmdLength + 1], /* Command embedded 1st 3 bytes of
                                       message */
        *readnet(),
        *index(),                   /* Std index function */
        *eon_marker,                /* End of database name marker */
        dbid[MFNLength + 1];        /* ID of database currently being used */

 int    msg_len = 0,                /* Length of incoming message */
        glad_command,               /* Command portion of message */
        dbid_length,                /* Length of database name */
        i,
        not_found;                  /* Have we found a match between loaded */
                                    /* databases and requested database?    */

#ifdef EnExFlag
  printf("Enter glad_main\n");
  fflush(stdout);
#endif

  TI_chk_reqs_left();
  rcv_socket = recv_socket;
  snd_socket = send_socket;
  strcpy(dbid,"");
  strcpy(command_str,"   ");
  TI_GUser(user_id);    /* Get the user's ID */

  /* Specify to system that the user always wants to wait for a response */
  /* from the system before proceeding to the next transaction.          */
  always_wait_flag = TRUE;

  while (1)
  {
    Glad_message = readnet(&msg_len);

    if (msg_len == 0)
      return (GLADDown);
    if (msg_len == -1)
      return (ReadError);
    if (msg_len == GLADTerminate)
      return(GLADTerminate);

    /* Parse the GLAD command from the message string */
```

```
strncpy(command_str,Glad_message,3);
glad_command = str_to_num(command_str);
printf("\nMBDS received command #%s ", command_str);

switch (glad_command)
{
 case OpenDatabase : /* Glad wants to open a database */
  printf("for the %s database.\n", &Glad_message[3]);
  /* Parse the database ID from Glad_message */
  strcpy(dbid, &Glad_message[3]);
  i = 0;
  not_found = 1;
  /* See if the database is already loaded */
  while (i < numDBs && (not_found = strcmp(dbid,loadedDBs[i])))
   i++;
  if (not_found)
  {
   /* Prepare requested database for queries */
   strcpy(current_dbid,dbid);
   glad_db_setup();
   reqs_left_count = 0;
  }
 break;

 case DBQuery : /* Glad wants to query the database */
  eon_marker = index(&Glad_message[3],'@');
  if (eon_marker == 0)
  {
   printf("with the following message:\n%s", &Glad_message[3]);
   glad_error(InvalidMsg,"");
  }
  else
  {
   dbid_length = eon_marker - &Glad_message[3];
   strncpy(dbid,&Glad_message[3], dbid_length);
   i = 0;
   not_found = 1;
   printf("for the %s database, with the following query:\n%s",
        dbid, eon_marker + 1);
   /* See if the query is for a database that is currently open */
   while (i < numDBs && (not_found = strcmp(dbid,loadedDBs[i])))
    i++;
   if (not_found)
    glad_error(DBNotLoaded,"");
```

```
        else
        {
          if (strcmp(dbid,current_dbid))
          {
            /* Not the current database, change it to current database */
            dbl_template(dbid);
            strcpy(current_dbid,dbid);
          }
          /* Parse the database query from the message */
          strcpy(traf_unit, eon_marker + 1);

          /* Query the database */
          glad_db_query(traf_unit);
        }
      }
      break;

    default : /* Glad has sent an invalid message */
      glad_error(InvalidMsg,"");

   } /* end switch */

   free(Glad_message);

  } /* end while */

#ifdef EnExFlag
  printf("Exit glad_main\n");
  fflush(stdout);
#endif

} /* end glad_main */

/*****************************************************************************************/
/*                                    GLAD_DB_SETUP                                      */
/*****************************************************************************************/

glad_db_setup()
{
  char *add_path(),
       filename[MFNLength+1],
       user_id[USERidLength + 1],      /* ID of user, currently fixed to "user" */
       dbid[MFNLength+1],              /* ID of database currently being used */
       dbid_cpy[MFNLength+1];
```

```
  FILE *fopen(), *fptr1;

#ifdef EnExFlag
  printf("Enter glad_db_setup\n");
  fflush(stdout);
#endif

  get_DB(dbid);
  if (!(confirm_database(dbid)))
  {
    glad_error(DBNotExist,"");
    strcpy(current_dbid,""); /* reset database ID */
  }
  else
  {
    dbl_template(dbid);    /* Loads the record templates for the new database */
    strcpy(dbid_cpy,dbid);
    strcpy(filename,strcat(dbid_cpy,".r"));
    fptr1 = fopen(add_path(filename), "r");
    if (fptr1)
    {
      dbl_records(fptr1,dbid);
      /* Brodcast the user and database ids */
      DBL_S$Use(dbid);
      TI_S$AssignDB(user_id, dbid);
      strcpy(loadedDBs[numDBs++],dbid);
    }
    else
    {
      glad_error(DBNoRecords,"");
      strcpy(current_dbid,"");
    }
  }

#ifdef EnExFlag
  printf("Exit glad_db_setup\n");
  fflush(stdout);
#endif

} /* end glad_db_setup */
```

```
/*****************************************************************************/
/*                              SEND RESULTS                                 */
/*****************************************************************************/

sendResults(rid, res, res_len)
/* Send the results (response) from MDBS for a request to GLAD */
struct  ReqId *rid;
char    res[];
int     res_len;
{
 char qres[RESLength + CmdLength];
 char first_attr[ANLength + 1], attr[ANLength + 1], val[AVLength + 1];
 int    first,
        first_rec,
        req_done,
        qres_len = 3,
        k = 1,
        j = 0;

#ifdef EnExFlag
 printf("Enter sendResults\n");
 fflush(stdout);
#endif

 strcpy(qres,"010");

 if (res_len >= 2)
 {
  if (res[res_len - 3] == CSignal)
  {
   /* Indicate that a request has finished execution */
   req_done = TRUE;
   res_len -= 2;
  }
  else
   req_done = FALSE;

  /* get the first attribute to see if it is a RETRIEVE COMMON */
  while (first_attr[j++] = res[k++])
   ;

#ifdef msgflag
 printf("\nreqid: (traf_id = %s, req_no = %s) \n", rid->traf_id, rid->req_no);
 fflush(stdout);
```

```
#endif

   first_rec = TRUE;
   k = 1;

   while (k < res_len - 1)
   {
    /* copy the attribute name */
    j = 0;
    while (attr[j++] = res[k++])
      ;
#ifdef pr_flag
 printf("attr = >%s<\n", attr);
 fflush(stdout);
#endif

    /* copy the attribute value */
    j = 0;
    while (val[j++] = res[k++])
      ;

#ifdef pr_flag
 printf("val = >%s<\n", val);
 fflush(stdout);
#endif

    first = strcmp(first_attr,attr);

    if (!first_rec && !first)
    {
     /* Insert end of attribute and record markers */
     qres[qres_len++] = '&';
     qres[qres_len++] = '@';
    }
    else if (first && attr[0] != '#')
     /* Insert end of attribute marker */
     qres[qres_len++] = '&';
    else
     first_rec = FALSE;

    if (attr[0] == '#')
    {
     /* For aggregate ops copy end of attribute and record marks */
     /* to the query response                                    */
```

```
      qres[qres_len++] = '&';
      qres[qres_len++] = '@';
    }
    else
    {
      j = 0;
      /* Copy the attribute value to the file */
      while (qres[qres_len++] = val[j++])
        ;
      /* Back up over the null character */
      qres_len--;

      if (k >= res_len - 1 || res[k] == '#')
      {
        /* Insert end of attribute and record markers */
        qres[qres_len++] = '&';
        qres[qres_len++] = '@';
      }
    }

  } /* end while */

  qres[qres_len++] = '\0';
  writenet(qres, qres_len);

#ifdef msgflag
  if (req_done)
  {
    printf("\nrequest with id: (traf_id = %s, req_no = %s)  ",
              rid->traf_id, rid->req_no);
    printf("is completed \n");
  }
  fflush(stdout);
#endif

#ifdef EnExFlag
  printf("Exit sendResults\n");
  fflush(stdout);
#endif

  } /* end if */
  else
    /* This is answer to request such as update, which returns */
    /* no attributes or values                                 */
```

```
    writenet("010&@", 0);

} /* end sendResults */

/*****************************************************************************************/
/*                                    GLAD_DB_QUERY                                      */
/*****************************************************************************************/

glad_db_query(trafunit)
char   trafunit[TULength];              /* Current database query */
{
 char  user_id[USERidLength + 1],       /* ID of user, currently fixed to "user" */
       dbid[MFNLength + 1],             /* ID of database currently being used */
       response[RESLength],
       err_msg[RESLength];
 int   MsgType,
       i,
       res_len;
 struct ReqId rid;

#ifdef EnExFlag
 printf("Enter glad_db_query\n");
 fflush(stdout);
#endif

 get_DB(dbid);

 /* Send the traffic unit to Request Preparation */
 TI_S$TrafUnit(dbid, trafunit);

 while (reqs_left_count)
 {
  /* Receive the next message from TI */
  TI_R$Message();
  MsgType = TI_R$Type();

  switch (MsgType)
  {
   case CH_ReqRes:  /* Results (possibly partial) for request available */
     /* Get the results */
     TI_R$ReqRes(&rid, response);

     /* Find out if done, and the response length */
     for (res_len = 0; response[res_len] != EOResult; ++res_len)
```

```
        ;
      ++res_len;

printf("\nSending query results back to GLAD\n");
sendResults(&rid, response, res_len);

      if (response[res_len - 3] == CSignal)   /* The request is done */
      {
        --reqs_left_count;
        writenet("020", 0);
      }
      break;

    case ReqsWithErr:  /* Traffic unit has errors in it */
      /* receive the error message */
      TI_R$ErrorMessage(trafunit, err_msg);

      /* send the error message back to GLAD */
      glad_error(QueryError,err_msg);
      break;

    case CH_TransDone:  /* The transaction is done */
      /* Don't need to worry about this since we will be doing only */
      /* one transaction at a time.                                 */
      break;

    default:
      SysError(2, "glad_db_query");
      sleep(ErrDelay);

   } /* end switch */

  } /* end while */

  /* process finishing touches for the request */
  if (f_flg)
    fclose(r_file_ptr);

#ifdef EnExFlag
  printf("Exit glad  db_query\n");
  fflush(stdout);
#endif

} /* end glad_db_query */
```

```
/*****************************************************************************/
/*                                  READNET                                  */
/*****************************************************************************/

char   *readnet(bytes_read)
int    *bytes_read;
{
 int    msg_len;
 char  buffer[MLBuffer + 1],
       *new_message,
       *var_str_alloc();

#ifdef EnExFlag
 printf("Enter readnet\n");
 fflush(stdout);
#endif

  while ((*bytes_read = read(rcv_socket, buffer, MLBuffer)) < MLBuffer)
  {
   /* Wait until full message is passed and check for GLAD going down */
   if (*bytes_read == 0)
     return (new_message);
   if (*bytes_read < 0)
   {
    glad_error(ReadError,"");
    *bytes_read = -1;
    return(new_message);
   }
  }
  buffer[MLBuffer] = '\0';
  msg_len = str_to_num(buffer);
  if (msg_len <= 0)
  {
   *bytes_read = -2;
   return (new_message);
  }
  new_message = var_str_alloc(msg_len);

  while ((*bytes_read = read(rcv_socket, new_message, msg_len)) < msg_len)
  {
   /* Wait until the full database message is passed from GLAD and */
   /* check to make sure GLAD has not gone down */
   if (*bytes_read == 0)
```

```
    return (new_message);
   if (*bytes_read < 0)
   {
    glad_error(ReadError,"");
    *bytes_read = -1;
    return (new_message);
   }
  }
  return (new_message);

#ifdef EnExFlag
 printf("Exit readnet\n");
 fflush(stdout);
#endif

}

/*****************************************************************************************/
/*                                    WRITENET                                           */
/*****************************************************************************************/

writenet(buffer, length)
char buffer[];
int  length;
{
 int    bytes_written;          /* Bytes written to GLAD */
 char  msg_len[MLBuffer + 1],   /* Message length to be passed to GLAD */
       len_build[MLBuffer + 1]; /* Used to build message length string */

#ifdef EnExFlag
 printf("Enter writenet\n");
 fflush(stdout);
#endif

 /* Figure and send message length to GLAD */
 if (!length)
   length = strlen(buffer);
 num_to_str(length, msg_len);


 /* Pad message length string with leading zeroes to produce 4 characters */
 strcpy(len_build,"");
 if (length/1000)
   ;
```

```
  else if (length/100)
    strcpy(msg_len,strcat(strcat(len_build,"0"),msg_len));
  else if (length/10)
    strcpy(msg_len,strcat(strcat(len_build,"00"),msg_len));
  else
    strcpy(msg_len,strcat(strcat(len_build,"000"),msg_len));


  /* Send message length to GLAD */
  while ((bytes_written = write(snd_socket, msg_len, MLBuffer)) < MLBuffer)
    if((bytes_written < 0) && (errno != EWOULDBLOCK))
    {
      /* Data not being written was not due to network overflow */
      return (WriteError);
    }
  /* Send message to GLAD */
  while ((bytes_written = write(snd_socket, buffer, length)) < length)
  {
    if ((bytes_written < 0) && (errno != EWOULDBLOCK))
    {
      /* Data not being written was not due to network overflow */
      return (WriteError);
    }
  }

#ifdef EnExFlag
  printf("Exit writenet\n");
  fflush(stdout);
#endif

}

/*****************************************************************************************/
/*                                        GLAD_ERROR                                     */
/*****************************************************************************************/

glad_error(error_no,error_msg)
int     error_no;
char    error_msg[RESLength];
{
#ifdef EnExFlag
  printf("Enter glad_error\n");
  fflush(stdout);
#endif
```

```
switch (error_no)
{
 case DBLoaded : /* Database is already loaded */
  writenet("100A Database has already been loaded.  Must exit and re-enter
     MBDS.", 0);
  break;

 case DBNotLoaded : /* Trying to query a database that has not been loaded */
  writenet("100Attempting to query a database that has not been loaded",
     0);
  break;

 case InvalidMsg : /* GLAD sent an unrecognized message */
  writenet("100Last message was not a valid message", 0);
  break;

 case DBNotExist : /* Attempted to open a database that doesn't exist */
  writenet("100Attempted to open a database that doesn't exist", 0);
  break;

 case DBNoRecords : /* There is no record file for the database */
  writenet("100No record file (.r) exists for the database", 0);
  break;

 case QueryError : /* The last message was an invalid query */
  writenet(strcat("100The last query was invalid.  ",error_msg), 0);
  reqs_left_count = 0;
  break;

 case ReadError : /* Error reading data from GLAD */
  writenet(strcat("100MBDS receive socket is down.  ",error_msg), 0);
  break;

 default : /* Invalid error message code received */
  printf("\nInvalid error message code received in glad_error.\n");
}
printf("\n***Error message sent to GLAD.  The error code is %d.\n", error_no);
if (strcmp(error_msg,""))
 printf("The error message is %s.\n",error_msg);
}
```

C. GLAD_LIL.H HEADER FILE

```
#define Glad_Rcv_Port     1705
#define Glad_Snd_Port     1710
#define PC307_ADDR        (u_long)0x83780130

#define MLBuffer    4
#define MaxDBs      10
#define MsgLength   2000
#define CmdLength   3

#define OpenDatabase      10
#define DBQuery           20
#define GladError         100

#define DBLoaded          10
#define DBNotLoaded       20
#define InvalidMsg        30
#define DBNotExist        40
#define DBNoRecords       50
#define QueryError        60
#define ReadError         70

#define GLADTerminate     -2
#define WriteError        -1
#define GLADDown          0
```

APPENDIX B. GLAD SOCKET INTERFACE SOURCE CODE LISTING

The listing that follows is the code generated for the GLAD socket interface implemented in this thesis.

A. SOCKET INTERFACE

```
#include <windows.h>
#include <dde.h>
#include <io.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stddef.h>
#include <ctype.h>
#include <fcntl.h>
#include <signal.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>

/* network defintions */
#include <sys/extypes.h>
#include <sys/exerrno.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/reply.h>
#include <sys/soioctl.h>
#include "sockets.h"

/*****************************************************************************/
/*                                  WIN_MAIN                                 */
/*****************************************************************************/

int PASCAL WinMain (hInstance, hPrevInstance, lpszCmdLine, nCmdShow)

HANDLE          hInstance, hPrevInstance;
LPSTR           lpszCmdLine;
int             nCmdShow;
```

```
{
 HWND          hWnd;
 MSG           msg;
 WNDCLASS      wndclass;

 if (!hPrevInstance)
 {
  wndclass.style          = CS_HREDRAW | CS_VREDRAW;
  wndclass.lpfnWndProc    = WndProc;
  wndclass.cbClsExtra     = 0;
  wndclass.cbWndExtra     = 0;
  wndclass.hInstance      = hInstance;
  wndclass.hIcon          = LoadIcon (hInstance, szAppName);
  wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW);
  wndclass.hbrBackground  = GetStockObject (WHITE_BRUSH);
  wndclass.lpszMenuName   = NULL;
  wndclass.lpszClassName  = szAppName;

  if (!RegisterClass (&wndclass))
    return FALSE;
  }

  hInst = hInstance;
  hWnd = CreateWindow (szAppName, "GLAD to MBDS Socket Interface",
                WS_OVERLAPPEDWINDOW,
                CW_USEDEFAULT, 0,
                CW_USEDEFAULT, 0,
                NULL, NULL, hInstance, NULL);

#ifdef debugFlag
  ShowWindow (hWnd, nCmdShow);
#endif

#ifndef debugFlag
  ShowWindow (hWnd, SW_SHOWMINNOACTIVE);
#endif

  UpdateWindow (hWnd);

  CreateSockets (hWnd);

  SetTimer (hWnd, TimerId, 2000, NULL);

  while (GetMessage (&msg, NULL, 0, 0))
```

```
  {
   TranslateMessage (&msg);
   DispatchMessage (&msg);
   if (mbds_up)
    ReadNet (hWnd);
  }
  return msg.wParam;
}
/*****************************************************************************/
/*                              HANDLE ERROR                                 */
/*****************************************************************************/

void HandleError (hWnd, errorType)

HWND        hWnd;
int         errorType;
{
 HDC        hDC;
 char       errorMessage[80];

 switch (errorType)
 {
  case MBDS_DIED :
   mbds_up = FALSE;
   strcpy(errorMessage,"130MBDS has shut down due to an error");
   break;

  case SEND_SOCKET :
   strcpy(errorMessage,"030Unable to create sending socket");
   break;

  case RECV_SOCKET :
   strcpy(errorMessage,"040Unable to create receiving socket");
   break;

  case NON_BLOCK :
   strcpy(errorMessage,"050Unable to initiate synchronous non-blocking I/O");
   break;

  case SOCK_CONN :
   strcpy(errorMessage,"060Unable to connect send socket to MBDS");
   break;

  case SOCK_ACCEPT :
```

```
            strcpy(errorMessage,"070Unable to accept connection from MBDS");
            break;

          case SOCK_READ :
            strcpy(errorMessage,"080Error in reading from receive socket");
            break;

          case SOCK_WRITE :
            strcpy(errorMessage,"090Error in writing to send socket");
            break;

          case MEM_ALLOC :
            strcpy(errorMessage,"100Unable to allocate memory for incoming
               message");
            break;

          case MEM_LOCK :
            strcpy(errorMessage,"110Unable to lock memory for incoming message");
            break;

          case FILE_OPEN:
            strcpy(errorMessage,"120Unable to open query results file");
            break;

          default :
            strcpy(errorMessage,"140Undefined error occured in socket interface");
            break;
        }
        /* Send Socket Interface Error message to GLAD */
        wDDEData = GlobalAddAtom(errorMessage);
        /* Send the message to all windows */
        PostMessage ((LONG)0xFFFF,
                WM_DDE_DATA,
                hWnd,
                MAKELONG (CF_TEXT, wDDEData));

      #ifdef debugFlag
        hDC = GetDC (hWnd);
        TextOut (hDC, xChar, yChar * (1 + line_no), szBuffer,
              sprintf (szBuffer, "Error Type = %d, Error Message = %s.",
              errorType, &errorMessage[3]));
        ReleaseDC (hWnd, hDC);
        line_no++;
        if (line_no > 20)
```

```
   line_no = 0;
#endif

}

/*****************************************************************************/
/*                              CREATE SOCKETS                               */
/*****************************************************************************/

void CreateSockets(hWnd)

HWND        hWnd;
{
  int   timelimit = 30 ,
        on = 1;

  hostname = "db1";
  addr = rhost (&hostname);

  my_send_socket.sin_family = AF_INET;
  my_send_socket.sin_port = htons (0);
  my_send_socket.sin_addr.s_addr = 0;

  my_recv_socket.sin_family = AF_INET;
  my_recv_socket.sin_port = htons (GLAD_RECV_PORT);
  my_recv_socket.sin_addr.s_addr = 0;

  their_recv_socket.sin_family = AF_INET;
  their_recv_socket.sin_port = htons (GLAD_SEND_PORT);
  their_recv_socket.sin_addr.s_addr = addr;

  /* Create network sockets */
  send_socket = socket (SOCK_STREAM, (struct sockproto *)0,
                  &my_send_socket, SO_KEEPALIVE);
  if (send_socket < 0)
    HandleError (hWnd, SEND_SOCKET);

  recv_socket = socket (SOCK_STREAM, (struct sockproto *)0,
                 &my_recv_socket, (SO_ACCEPTCONN|SO_KEEPALIVE));
  if (recv_socket < 0)
    HandleError (hWnd, RECV_SOCKET);

  /* Limit time for other end to take all data before closing */
  soioctl (send_socket, SIOCSLINGER, &timelimit);
```

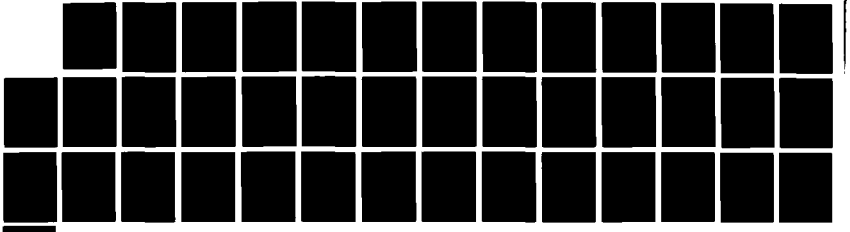
```
  soioctl (recv_socket, SIOCSLINGER, &timelimit);

  /* Specify synchronous-nonblocking I/O so we don't lose control */
  rc = soioctl (send_socket, FIONBIO, &on);
  if (rc < 0)
    HandleError (hWnd, NON_BLOCK);

  rc = soioctl (recv_socket, FIONBIO, &on);
  if (rc < 0)
    HandleError (hWnd, NON_BLOCK);

  /* Connect it to the backend */
  while (((rc = connect (send_socket, &their_recv_socket)) < 0)
         && ((errno == EWOULDBLOCK) || (errno == EINPROGRESS)))
           ;
  if ((rc < 0) && (errno != EISCONN))
  {
    /* Compensate for known possible anomaly */
    if ((repetitions > 1) && (errno == EINVAL))
      errno = ETIMEDOUT;
    HandleError (hWnd, SOCK_CONN);
  }

  /* Accept connection from backend */
  while ((rc = accept (recv_socket,&their_send_socket)) < 0 &&
         (errno == EWOULDBLOCK));
   ; /* Wait until the backend initiates a connection */
  if (rc < 0)
    HandleError (hWnd, SOCK_ACCEPT);

  socket_open = TRUE;
}
/*****************************************************************************/
/*                              RESULTS TO FILE                              */
/*****************************************************************************/

void ResultsToFile (hWnd, pMem, length)

HWND        hWnd;
PSTR        pMem;
int         length;
{
  HANDLE    hResultsFile;
  int       last_eoa_mark,
```

```
                count;

  if (newFile)
  /* New query - Delete existing query results file and start a new one */
  {
    hResultsFile = OpenFile (szFileName, &ofDataFile,
                        OF_CREATE | OF_READWRITE);
    newFile = FALSE;
  }
  else
  /* More results from same query - Append to the existing query results file */
  {
    hResultsFile = OpenFile (szFileName, &ofDataFile,
                        OF_REOPEN | OF_READWRITE);
    if (hResultsFile == (HANDLE)-1)
      HandleError (hWnd, FILE_OPEN);
  }
  /* Remove '@' delimiting end of record and replace with CR-LF */
  last_eoa_mark = -1;
  for (count = 0; count <= length; count++)
  {
    if (pMem[count] == '@')
    {
      write (hResultsFile, &pMem[last_eoa_mark + 1], count - last_eoa_mark - 1);
      last_eoa_mark = count;
      write (hResultsFile, "\x0d\x0a", 2);
    }
  }
  close (hResultsFile);
}

/*****************************************************************************/
/*                                  READNET                                  */
/*****************************************************************************/

void ReadNet (hWnd)

HWND        hWnd;
{
  char      buffer[MLBUFFER + 1];
  int       rc,
            msg_len;
  HANDLE    hMem;
  PSTR      pMem;
```

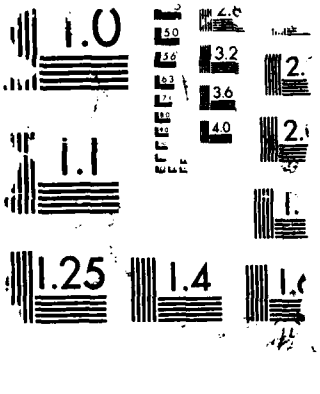
```
void        ParseResponse();

if (socket_open)
{
 /* Is there an incoming message? */
 if ((rc = soread (recv_socket, buffer, MLBUFFER)) > 0)
 {
  /* Is it a message length? */
  if (rc == MLBUFFER)
  {
   buffer[MLBUFFER] = '\0';
   msg_len = atoi(buffer);

   /* Allocate and lock local memory for the incoming message */
   hMem = LocalAlloc (LMEM_FIXED, (WORD) msg_len + 1);
   /* Was Allocation successful? */
   if (hMem == NULL)
    HandleError (hWnd, MEM_ALLOC);

   pMem = LocalLock (hMem);
   /* Was lock successful? */
   if (pMem == NULL)
   {
    LocalFree (hMem);
    HandleError (hWnd, MEM_LOCK);
   }
   /* Wait until we receive the full message from MBDS */
   while ((((rc = soread (recv_socket, pMem, msg_len)) < 0) &&
           errno == EWOULDBLOCK) || rc < msg_len)
    /* Has MBDS died? */
    if (rc == 0)
    {
     LocalUnlock (hMem);
     LocalFree (hMem);
     HandleError (hWnd, MBDS_DIED);
    }
   /* Did an error occur during the socket read? */
   if (rc < 0)
   {
    LocalUnlock (hMem);
    LocalFree (hMem);
    HandleError (hWnd, SOCK_READ);
   }
   pMem[rc] = '\0';
```

AD-A220 081 INTERCONNECTION OF THE GRAPHICS LANGUAGE FOR DATABASE ... 2/2
SYSTEM TO THE MULTI (U) NAVAL POSTGRADUATE SCHOOL
MONTEREY CA T R HOGAN DEC 89

UNCLASSIFIED F/G 12/5 NL



END
FILMED
DTIC


1.0
1.1
1.25
1.4
3.2
3.6
4.0

```
      /* Parse the command and send to GLAD */
      ParseResponse(hWnd, pMem, rc);

      /* Free the memory used for the message */
      LocalUnlock (hMem);
      LocalFree (hMem);
    }
   }
   /* Check if read error occured or if MBDS died */
   else if ((rc < 0) && (errno != EWOULDBLOCK))
    HandleError (hWnd, SOCK_READ);
   else if (rc == 0)
    HandleError (hWnd, MBDS_DIED);
  }
}

/*****************************************************************************/
/*                          PARSE RESPONSE                                   */
/*****************************************************************************/

void ParseResponse(hWnd, pMem, Length)

HWND        hWnd;
PSTR        pMem;
int         Length;
{
 char       command_str[CMDLENGTH + 1];
 int        mbds_command;
 HDC        hDC;

 /* Parse the response from MBDS message */
 strncpy(command_str, pMem, CMDLENGTH);
 command_str[CMDLENGTH] = '\0';
 mbds_command = atoi(command_str);

 switch (mbds_command)
 {
  case QUERY_RESULTS : /* MBDS is returning the results of a query */
   ResultsToFile (hWnd, &pMem[3], Length - CMDLENGTH);
   break;

  case END_OF_RESULTS: /* MBDS has finished processing the query */
   newFile = TRUE;
```

```
    default : /* Send End of Result or Error message to GLAD */
      wDDEData = GlobalAddAtom(pMem);
      /* Send the message to GLAD */
      PostMessage (hClientWindowHandle,
                   WM_DDE_DATA,
                   hWnd,
                   MAKELONG (CF_TEXT, wDDEData));

   } /* End of switch */

#ifdef debugFlag
   hDC = GetDC (hWnd);
   TextOut (hDC, xChar, yChar * (1 + line_no), szBuffer,
          sprintf (szBuffer, "Command = %d, Message = %s",
          mbds_command, &pMem[3]));
   ReleaseDC (hWnd, hDC);
   line_no++;
   if (line_no > 20)
     line_no = 0;
#endif

}

/*****************************************************************************************/
/*                                        WRITENET                                       */
/*****************************************************************************************/

int WriteNet(hWnd, buffer)

HWND        hWnd;
char        *buffer;
{
   HDC      hDC;              /* Handle to device context */
   int      bytes_written,    /* Bytes written to MBDS */
            bytes_to_write;   /* Length of message to be sent */
   char  msg_len[MLBUFFER + 1], /* Message length to be passed to MBDS */
         len_build[MLBUFFER + 1];/* Used to build message length string */

   strcpy (len_build,"");
   bytes_to_write = strlen (buffer);
   /* Convert message length to a string */
   itoa (bytes_to_write, msg_len, 10);
```

```
#ifdef debugFlag
 hDC = GetDC (hWnd);
 TextOut (hDC, xChar, yChar * (1 + line_no), szBuffer,
        sprintf (szBuffer, "buffer = %s, bytes_to_write = %d, msg_len = %s",
        buffer, bytes_to_write, msg_len));
 line_no++;
 if (line_no > 20)
   line_no = 0;
#endif

 /* Pad message length string with leading zeroes to produce 4 characters */
 if (bytes_to_write/1000)
  ;
 else if (bytes_to_write/100)
   strcpy (msg_len, strcat (strcat (len_build,"0"), msg_len));
 else if (bytes_to_write/10)
   strcpy (msg_len, strcat (strcat (len_build,"00"), msg_len));
 else
   strcpy (msg_len, strcat (strcat (len_build,"000"), msg_len));

#ifdef debugFlag
 TextOut (hDC, xChar, yChar * (1 + line_no), szBuffer,
        sprintf (szBuffer, "msg_len = %s",
        msg_len));
 ReleaseDC (hWnd, hDC);
 line_no++;
 if (line_no > 20)
   line_no = 0;
#endif

 /* Send message length to MBDS. Ensure message is sent in its entirety */
 while ((bytes_written = sowrite (send_socket, msg_len,
                MLBUFFER)) < MLBUFFER)
 {
  if ((bytes_written < 0) && (errno != EWOULDBLOCK))
    /* Data not being written was not due to network overflow */
    HandleError (hWnd, SOCK_WRITE);
 }

 /* If message length is zero, this is a quit command */
 if (bytes_to_write)
 {
  /* Send message to MBDS. Ensure message is sent in its entirety */
  while ((bytes_written = sowrite (send_socket, buffer, bytes_to_write)) <
```

```
            bytes_to_write)
    {
     if ((bytes_written < 0) && (errno != EWOULDBLOCK))
       /* Data not being written was not due to network overflow */
       HandleError (hWnd, SOCK_WRITE);
    }
   }
  return (bytes_written);
}
/*****************************************************************************************/
/*              CREATE PROCEDURE FOR SOCKET INTERFACE                    */
/*****************************************************************************************/

/* Process WM_CREATE messages here */

void ProcessSocketCreate (hWnd)

HWND        hWnd;
{
  staticHICON               hIcon;
        HDC                 hDC;
        TEXTMETRIC          tm;

  /* Load Socket Icon and get Text Metrics */
  hIcon = LoadIcon (hInst, szAppName);
  hDC = GetDC (hWnd) ;
  GetTextMetrics (hDC, &tm) ;
  xChar = tm.tmAveCharWidth ;
  yChar = tm.tmHeight + tm.tmExternalLeading ;
  ReleaseDC (hWnd, hDC) ;
}
/*****************************************************************************************/
/*              SOCKET INTERFACE INITIATE DDE PROCEDURE                  */
/*****************************************************************************************/

/* Process WM_DDE_INITIATE messages here */

void ProcessInitiateDDE (hWnd, wClient, lParam)

HWND        hWnd;
WORD        wClient;
LONG        lParam;
{
  WORD      wServer,      /* Requested server application */
```

```
            wTopic;       /* Requested server topic */

 wServer = LOWORD (lParam);
 wTopic  = HIWORD (lParam);

 /* Get the name of the desired server */
 GlobalGetAtomName (wServer, szBuffer, SZBUFFER);
 wDDEServer = GlobalFindAtom (szBuffer);

 if (strcmpi ("Sockets", szBuffer) != 0)  /* If not for us, ignore */
   return;

 /* Get the name of the desired topic */
 GlobalGetAtomName (wTopic, szBuffer, SZBUFFER);
 wDDETopic = GlobalFindAtom (szBuffer);

 if (strcmpi ("Database", szBuffer) != 0) /* If not our topic, ignore */
   return;

 SendMessage     (wClient,           /* Address message to client */
                  WM_DDE_ACK,        /* Acknowledge initiation */
                  hWnd,              /* Let client know our window handle */
                  MAKELONG (wDDEServer, wDDETopic));
                                     /* Let client know the server  name and topic */
}

/*****************************************************************************/
/*                          HANDLE GLAD'S REQUESTS                           */
/*****************************************************************************/

/* Handle WM_DDE_REQUEST messages here */

void ProcessRequestDDE (hWnd, hClient, lParam)

HWND        hWnd;
HANDLE      hClient;
LONG        lParam;
{
 GLOBALHANDLE hMem;
 LPSTR        lpMem;

 /* Get atom of requested data item */
 wDDEData = HIWORD (lParam);
 /* Save the requesting window's handle */
```

```
  hClientWindowHandle = hClient;

  GlobalGetAtomName (wDDEData, szInBuffer, SZBUFFER);
  GlobalDeleteAtom (wDDEData);
  WriteNet(hWnd, szInBuffer);
}
/*****************************************************************************/
/*                          WINDOW PROCEDURE                                 */
/*****************************************************************************/

long FAR PASCAL WndProc (hWnd, iMessage, wParam, lParam)

HWND        hWnd;
unsigned    iMessage;
WORD        wParam;
LONG        lParam;
{
  HDC               hDC;
  PAINTSTRUCT       ps;

  switch (iMessage)
  {
    case WM_CREATE :
      ProcessSocketCreate (hWnd, lParam);
      break;

    case WM_DDE_INITIATE :  /* GLAD is ready to initiate communications */
      ProcessInitiateDDE (hWnd, wParam, lParam);
      break;

    case WM_DDE_REQUEST :   /* GLAD has a message for us or MBDS */
      ProcessRequestDDE (hWnd, wParam, lParam);
      break;

#ifndef debugFlag
    case WM_QUERYOPEN :
      break;
#endif

    case WM_QUERYENDSESSION :
      /* Send MBDS a termination signal and close sockets before closing */
      WriteNet (hWnd, "");
      socket_open = FALSE;
      soclose (send_socket);
```

```
      soclose (recv_socket);
      return (TRUE);
      break;

    case WM_CLOSE :
      /* Send MBDS a termination signal and close sockets before closing */
      WriteNet (hWnd, "");
      socket_open = FALSE;
      soclose (send_socket);
      soclose (recv_socket);
      DestroyWindow (hWnd);
      break;

    case WM_DESTROY :
      PostQuitMessage (0);
      break;

    default:
      return DefWindowProc (hWnd, iMessage, wParam, lParam);
  }
  return 0L;
}
```

B. SOCKET INTERFACE HEADER FILE

```
#define EOFCHAR             '\032' /* ^Z */
#define BUFFERSIZE          1000
#define SZBUFFER            256
#define MLBUFFER            4
#define CMDLENGTH           3
#define GLAD_SEND_PORT      1705
#define GLAD_RECV_PORT      1710

/* Error Codes */
#define SEND_SOCKET   30
#define RECV_SOCKET   40
#define NON_BLOCK     50
#define SOCK_CONN     60
#define SOCK_ACCEPT   70
#define SOCK_READ     80
#define SOCK_WRITE    90
#define MEM_ALLOC     100
#define MEM_LOCK      110
```

```
#define FILE_OPEN       120
#define MBDS_DIED       130

/* Message Return Types from MBDS */
#define QUERY_RESULTS       10
#define END_OF_RESULTS      20

/* DDE Definitions */
#define DF_ACK_REQUIRED     0xffff
#define DF_CANNOT_SEND      0x8000
#define DF_CAN_SEND         0x0000
#define DF_CLIENT_RELEASE   0x2000

/* Forward declarations for program functions */
void    CreateSockets(),
        ReadNet(),
        testMBDS();

long FAR PASCAL WndProc (HWND, unsigned, WORD, LONG) ;

int     rc,
        line_no = 0,                /* Current line number to print on */
        send_socket = -1,           /* Sending socket identifier */
        recv_socket = -1;           /* Receiving socket identifier */

long   addr;   /* Internet address */

extern long rhost();                /* Returns host address */

char    szBuffer [SZBUFFER],        /* String buffer */
        szInBuffer [SZBUFFER],      /* Buffer for received DDE messages */
        *hostname = "",             /* Hostname of computer to connect with */
        szAppName[] = "Sockets"; /* Application name */

unsigned long repetitions = 0;

/* Excelan Socket Library Functions */
int     soread(),
        sowrite(),
        accept(),
        soioctl(),
        socket(),
        soclose(),
        connect();
```

```
short  htons();

struct sockaddr_in  my_send_socket,   /* PC's send socket address info */
                    their_recv_socket, /* Backend's recv socket address info */
                    my_recv_socket,   /* PC's recv socket address info */
                    their_send_socket; /* Backend's send socket address info */

static short  xChar,       /* Width of character */
              yChar;       /* Height of character */

HANDLE hInst;              /* Handle to this instance of server */

/* Dynamic Data Exchange Variables */
HWND      hClientWindowHandle;    /* Handle of the Client Window */
WORD      wDDETopic;        /* Word identifying topic of conversation */
WORD      wDDEServer;       /* Word identifying Socket (server) program */
WORD      wDDEData;         /* Word identifying data to be passed */

struct        /* Structure for working with DDE Header data */
{
  WORD Word1;
  WORD Word2;
} Header;

/* File I/O Variables */
OFSTRUCT ofDataFile;   /* Structure to hold values returned by Open File
                          routine */

BOOL socket_open,      /* Is the socket open to read from */
     newFile = TRUE,   /* Do we need to create or append  */
                       /* to the query results file      */
     mbds_up = TRUE;   /* Has MBDS died? */

char  szFileName[] = "QRESULTS.FIL"; /* File for saving query results */
int   TimerId = 1;
```

C. SOCKET INTERFACE RESOURCE FILE

```
sockets ICON sockets.ico
```

APPENDIX C. GLAD SOURCE CODE LISTING

The listing that follows includes those classes that were modified for the implementation of this thesis.

A. DMWINDOW CLASS (MODIFIED)

```
/* GLAD Window for data manipulation interaction */!!

inherit(MyWindow, #DMWindow,
#(dbSchema          /*meta data of opened db*/
prevObj             /*previously selected object if any */
selObj              /*currently selected object if any*/
colorTable          /*available colors for shading*/
rectSize            /*width & height of object rectangle expressed in Point*/
objMoved            /*true if object is dragged*/
prevCurPt           /*previous cursor point while object is dragged*/
boundRect           /*bounding box that surrounds all objects. Used for scroll
                    bars & setting windoworg*/
logOrg              /*origin of logical coordinate, mapped to device coord
              (0,0)*/
hScroll             /*true if there is HScrollBar*/
vScroll             /*true if there is VscrollBar*/
bDDEAcknowledge /* has DDE initiation been acknowledged? */
mbdsDB              /* Is the current DB an MBDS database? */
hSockets            /* Handle of Socket interface window */
haveData            /* have the results of the retrieve come back? */
awaitingData        /* Are we waiting for data from MBDS */
waitDialog          /* Dialog box which shows while waiting for data */
dbName              /* Name of the database */), 2, nil)!!

now(DMWindowClass)!!

now(DMWindow)!!

/* Formulate a retrieve request to get selected MBDS object's data */
Def  formulateRetrieve(self | aStr, attribs)
{
  aStr := new(String, 100);
  aStr := "020" + asUpperCase(dbName) + "@" +
```

```
                "[RETRIEVE(TEMP=" + name(selObj) + ")(";
 attribs := new(String, 50);
 attribs := "";
 do(attributes(selObj),
    {using(attr)
      attribs := attribs + asUpperCase(attr[NAME]) + ",";
    }
  );
 aStr := aStr + attribs + "&";
 aStr := subString(aStr, 0, indexOf(aStr, '&', 0) - 1);
 aStr := aStr + ")BY ";
 aStr := aStr + subString(attribs, 0, indexOf(attribs, ',', 0)) + "]";
 ^asciiz(aStr);
}!!

/* Tell MBDS to load the appropriate database */
Def loadMBDSDatabase(self, dbName | aStr, aCommand)
{
 aStr := delete(dbName, 0, 4);
 aStr := leftJustify(aStr);
 aStr := asUpperCase(rightJustify(aStr));
 aStr := "010" + aStr;
 aCommand := addAtom(self, lP(asciiz(aStr)));
  Call PostMessage(hSockets, WM_DDE_REQUEST, hWnd, pack(CF_TEXT,
aCommand));
}!!

/* Handle Dynamic Data Exchange acknowledgments */
Def WM_DDE_ACK(self, wp, lp | aServer, aTopic)
{
 aServer := Call GlobalFindAtom(lP(asciiz("Sockets")));
 aTopic := Call GlobalFindAtom(lP(asciiz("Database")));
 deleteAtom(self, aServer);
 deleteAtom(self, aTopic);
 hSockets := wp;
 bDDEAcknowledge := true;
}!!

/* Get the latest data for a MBDS object */
Def getMembers(self | aStr aCommand)
{
 aStr := formulateRetrieve(self);
 aCommand := addAtom(self, lP(aStr));
```

```
  Call PostMessage(hSockets, WM_DDE_REQUEST, hWnd, pack(CF_TEXT,
aCommand));
}!!

/* Handle Socket interfaces's WM_DDE_DATA messages here */
Def WM_DDE_DATA(self, wp, lp | mbdsCommand, aStr, objFile, delFile)
{
 aStr := getAtom(self, high(lp));
 deleteAtom(self, high(lp));
 mbdsCommand := subString(aStr, 0, 3);
 select

  case mbdsCommand = "020"
   objFile := new(File);
   delFile := new(File);
   setName(delFile, memberFile(selObj));
   delete(delFile);
   setName(objFile, "qresults.fil");
   reName(objFile, memberFile(selObj));
   setHaveMembers(selObj, true);
   haveData := "YES";
   if awaitingData
    end(waitDialog, 0);
   endif;
  endCase;

  default
   haveData := "ERROR";
   if awaitingData
    end(waitDialog, 0);
   endif;
   errorBox("ERROR", delete(aStr, 0, 3));

 endSelect;
}!!

/* Return the string that the atom references */
Def getAtom(self, atom | bufStr, aStr)
{
 bufStr := new(String, 100);
 Call GlobalGetAtomName(atom, lP(bufStr), 100);
 aStr := removeNulls(getText(bufStr));
 freeHandle(bufStr);
 ^aStr
```

```
}!!

/* Delete or decrease the count on an atom */
Def deleteAtom(self, param)
{
 if ((param >= 0xC000) and (param <= 0xFFFF))
  ^Call GlobalDeleteAtom(param);
 endif;
 ^nil
}!!

/* Create a new atom or increment the count of one that already exists */
Def addAtom(self, param)
{
 ^Call GlobalAddAtom(param);
}!!

/* Initiate Dynamic Data Exchange with the Socket interface */
Def initDDE(self, reason | aServer, aTopic)
{
 bDDEAcknowledge := false;
 aServer := addAtom(self, IP(asciiz("Sockets")));
 aTopic := addAtom(self, IP(asciiz("Database")));
   Call SendMessage(0xFFFFL, WM_DDE_INITIATE, hWnd, pack(aServer,
aTopic));
 if not(bDDEAcknowledge)
  if reason = "initDDE"
   errorBox("ERROR","Unable to initiate DDE with the Socket Interface");
  endif;
  ^false;
 else
  ^true;
 endif;
}!!

Def reDraw(self, obj | hdc)
{
 hdc := getContext(self);
 setLPtoDP(self,hdc);
 display(self,obj,hdc);
 releaseContext(self,hdc)
}!!

/*gets the filename from the name listed in the listbox*/
```

```
Def getGLADfilename(self ,selDb| tmpStr)
{
 tmpStr := new(String,30);
 tmpStr := "";
 do(selDb, {using(elem)
          if elem <> ' '
           tmpStr := tmpStr + asString(elem)
          endif });
 ^subString(tmpStr,0,7) + ".sch"
}!!

/*returns the currently selected object*/
Def selObj(self)
{
 ^selObj
}!!

/* draws an object on the window using
  the hdc display context */
Def display(self,obj,hdc | objName, objRect,
                    hBrush, hPen, hOldBrush, hOldPen)
{
 eraseRect(self,obj,hdc); /*first erase it*/
 /*select the color brush for filling
  used with Rectangle (via draw) */
 hBrush := Call CreateSolidBrush(color(obj));
 /*set bkcolor for shading with DrawText*/
 Call SetBkColor(hdc,color(obj));
 hOldBrush := Call SelectObject(hdc,hBrush);
 objRect := rect(obj);
 if obj.thickBorder /*draw it with a thick border*/
  hPen   := Call CreatePen(0,5,Call GetTextColor(hdc));
  hOldPen:= Call SelectObject(hdc,hPen);
  draw(objRect,hdc);
  Call SelectObject(hdc,hOldPen);/*restore the dc*/
  Call DeleteObject(hPen)
 else
  draw(objRect,hdc) /*with a reg. border*/
 endif;
 if nesting(obj) /*draw the inner box if it is a nested object*/
  draw(nestedRect(obj),hdc)
 endif;

 objName := name(obj);
```

```
  Call DrawText(hdc,lP(objName),-1,objRect,
            DT_CENTER bitOr DT_VCENTER
            bitOr DT_SINGLELINE);
  Call SelectObject(hdc,hOldBrush);
  Call DeleteObject(hBrush);
  freeHandle(objName)
}!!

/*erase the region a little larger than object
  rectangle in case it is displayed with a thick
  border*/
Def eraseRect(self,obj,hdc | hBrush tmpRect)
{
  tmpRect := copy(rect(obj));
  hBrush := Call CreateSolidBrush(WHITE_COLOR);
  Call FillRect(hdc,inflate(tmpRect,5,5),hBrush);
  Call DeleteObject(hBrush)
}!!

/*clientRect for DMWindow ignores the scroll bars
  if present*/
Def clientRect(self | cRect, incr)
{
  cRect := clientRect(self:WindowsObject);
  if hScroll
    incr := Call GetSystemMetrics(3);/*SM_CYHSCROLL*/
    setBottom(cRect, bottom(cRect)+incr-1)
  endif;
  if vScroll
    incr := Call GetSystemMetrics(2);/*SM_CXVSCROLL*/
    setRight(cRect, right(cRect)+incr-1)
  endif;
  ^cRect
}!!

/*move vert scroll bar down for incr amount*/
Def moveDownVScroll(self, incr | newy)
{
  newy := min(y(logOrg) + incr,
          bottom(boundRect) - height(clientRect(self)));
  /*adjust newy so it won't go beyond boundRect*/
  setLogOrg(self,x(logOrg),newy);
  setScrollPos(self,SB_VERT,newy);
  repaint(self)
```

```
}!!

/*move vert scroll bar up for incr amount*/
Def moveUpVScroll(self, incr | newy)
{
/*adjust newy so it won't go beyond boundRect*/
 newy := max(y(logOrg) - incr,top(boundRect));
 setLogOrg(self,x(logOrg),newy);
 setScrollPos(self,SB_VERT,newy);
 repaint(self)
}!!

/*check if any of four constraints is violated
 if so adust accordingly */
Def checkForViolation(self | cRect)
{
 cRect := clientRect(self);
 if (x(logOrg)+right(cRect)) > right(boundRect)
  logOrg.x := max(left(boundRect),
              right(boundRect)-right(cRect));
  setRight(boundRect, max(x(logOrg)+right(cRect),
                  right(boundRect)))
 endif;

 if (y(logOrg)+bottom(cRect)) > bottom(boundRect)
  logOrg.y := max(top(boundRect),
              bottom(boundRect)-bottom(cRect));
  setBottom(boundRect,max(y(logOrg)+bottom(cRect),
                  bottom(boundRect)))
 endif;

 if x(logOrg) < left(boundRect)
  setLeft(boundRect, min(left(boundRect),
                  right(boundRect)-right(cRect)));
  logOrg.x := left(boundRect)
 endif;

 if y(logOrg) < top(boundRect)
  setTop(boundRect, min(top(boundRect),
                 bottom(boundRect)-bottom(cRect)));
  logOrg.y := top(boundRect)
 endif
}!!
```

```
/*clientRect has changed. process only if the
  change is not by the changes in scroll
  bars, i.e. window size really changed */
Def reSize(self,wp,lp)
{
  adjBoundRect(self);/*always be adjusted*/
  setScrollRanges(self);
  repaint(self)

}!!

/*move horz scroll bar right for incr amount*/
Def moveRightHScroll(self, incr | newx)
{
  newx := min(x(logOrg) + incr,
          right(boundRect)-width(clientRect(self)));
  /*adjust newx so it won't go beyond boundRect*/
  setLogOrg(self,newx,y(logOrg));
  setScrollPos(self,SB_HORZ,newx);
  repaint(self)
}!!

/*move horz scroll bar left for incr amount*/
Def moveLeftHScroll(self, incr | newx)
{
  newx := max(x(logOrg) - incr,left(boundRect));
  /*adjust newx so it won't go beyond boundRect*/
  setLogOrg(self,newx,y(logOrg));
  setScrollPos(self,SB_HORZ,newx);
  repaint(self)
}!!

Def hThumbPos(self, lp | newx)
{
  newx := asInt(lp);
  setLogOrg(self,newx,y(logOrg));
  setScrollPos(self,SB_HORZ,newx);
  repaint(self)
}!!

Def vThumbPos(self, lp | newy)
{
  newy := asInt(lp);
  setLogOrg(self,x(logOrg),newy);
```

```
  setScrollPos(self,SB_VERT,newy);
  repaint(self)
}!!

Def downPage(self,lp)
{
  moveDownVScroll(self,height(clientRect(self)))
}!!

Def upPage(self,lp)
{
  moveUpVScroll(self,height(clientRect(self)))
}!!

Def rightPage(self,lp)
{
  moveRightHScroll(self,width(clientRect(self)))
}!!

Def leftPage(self,lp)
{
  moveLeftHScroll(self,width(clientRect(self)))
}!!

Def rightArrow(self,lp)
{
   moveRightHScroll(self,asInt(0.25*x(rectSize)))
}!!

Def downArrow(self,lp)
{
  moveDownVScroll(self,asInt(0.5*y(rectSize)))
}!!

Def upArrow(self,lp)
{
  moveUpVScroll(self,asInt(0.5*y(rectSize)))
}!!

Def leftArrow(self,lp)
{
  moveLeftHScroll(self,asInt(0.25*x(rectSize)))
}!!
```

```
/*convert device pt (DP) to logical pt (LP)*/
Def dPtoLP(self,aPt)
{
 ^point(aPt.x + logOrg.x, aPt.y + logOrg.y)
}!!

/*set the logical coord. origin*/
Def setLogOrg(self,x,y)
{
 logOrg.x := x;
 logOrg.y := y
}!!

/*logOrg is now mapped to device coord. (0,0)*/
/*need to pass hdc since there could be two disp context
allocated to this window at one time */
Def setLPtoDP(self, hdc)
{
 Call SetWindowOrg(hdc,x(logOrg),y(logOrg))
}!!

Def setScrollRanges(self | xmin,ymin,xmax,ymax)
{
 xmin:=left(boundRect);
 ymin:=top(boundRect);
 cRect := clientRect(self);
 if (xmax:=right(boundRect)-width(cRect)) < xmin
  xmax := xmin
 endif;
 if (ymax:=bottom(boundRect)-height(cRect)) < ymin
  ymax := ymin
 endif;
 hScroll := xmin < xmax;
 vScroll := ymin < ymax;
 setScrollRange(self,SB_HORZ,xmin,xmax);
 setScrollRange(self,SB_VERT,ymin,ymax)
}!!

Def adjBoundRect(self |tmpRect)
{
 tmpRect:=copy(boundRect);
 boundRect:=copy(rect(first(dbSchema)));
 do(dbSchema,
    {using(obj | objRect)
```

```
      objRect := rect(obj);
      setTop   (boundRect,min(top(objRect),top(boundRect)));
      setLeft  (boundRect,min(left(objRect),left(boundRect)));
      setBottom(boundRect,max(bottom(objRect),bottom(boundRect)));
      setRight (boundRect,max(right(objRect),right(boundRect)));
     } );
  /*changes boundRect accordingly per violation*/
  checkForViolation(self);
  if tmpRect = boundRect
    ^nil
  endif
}!!

Def start(self, databaseName | dosFilename, DDEInitiated)
{
  dbName := databaseName;
  mbdsDB := false;
  if subString(dbName, 0, 4) = "MBDS"
    mbdsDB := true;
    if not(initDDE(self, "socketActive?"))
      exec("sockets.exe");
    endif;
    if (DDEInitiated := initDDE(self, "initDDE"))
      loadMBDSDatabase(self, dbName);
      dbName := delete(dbName, 0, 5);
    endif;
  endif;
  if not(mbdsDB) or DDEInitiated
    dosFilename := getGLADfilename(self,dbName);
    loadSchema(self, dosFilename);
    adjBoundRect(self);
    setLogOrg(self, /*center of bRect to center of cRect*/
      left(boundRect)+asInt(0.5*(width(boundRect)-width(clientRect(self)))),
       top(boundRect)+asInt(0.5*(height(boundRect)-height(clientRect(self)))));
    setScrollRanges(self);
    setScrollPos(self,SB_HORZ,x(logOrg));
    setScrollPos(self,SB_VERT,y(logOrg));
    show(self,1);
  endif;
}!!

/*set the width and height of object rectangle*/
Def setObjRectSize(self | tm, wd, ht)
{
```

```
 tm := new(Struct,32);
 Call GetTextMetrics(hDC:=Call GetDC(hWnd),tm);
 wd:= 14*asInt(wordAt(tm,10));
 ht:= 4*asInt(wordAt(tm, 0));
 rectSize := point(wd,ht);
 Call ReleaseDC(hWnd,hDC)
}!!

/*mouse is dragged while left button is pressed.
 move obj if mouse is in it */
Def mouseMoveWithLBDn(self,wp,point | aLPt)
{
 if selObj
   objMoved := true;
   eraseRect(self,selObj,hDC);
   aLPt:= dPtoLP(self,point); /*convert to DP to LP*/
   setNewRect(selObj,aLPt,prevCurPt);
   prevCurPt:= aLPt;
   display(self,selObj,hDC)
 endif
}!!

Def initMenuID(self)
{
 menuID := %Dictionary( 1->#describe
                        2->#expand
                        3->#listMembers
                        4->#oneMember
                        5->#addMember
                        6->#deleteMember
                        7->#modifyMember
                        8->#query
                        9->#showConnection
                        950->#help
                        11->#close )
}!!

Def showConnection(self | aConnWin)
{
 errorBox("Show Connection","")
}!!

Def expand(self |win )
{
```

```
 if not(selObj)
  errorBox("ERROR!","No Object is selected")
 else
  if not(nesting(selObj))
   errorBox("ERROR!","Selected Object is not a nested object")
  else
   win := new(NestDMWindow,self,"GladDMIMenu",
          "SubClasses of: "+name(selObj),nil);
   addWindow(selObj,win);
   start(win,selObj,colorTable)
  endif
 endif
}!!

/*open oneMemWin for the selected object*/
Def oneMember(self | oneMemWin, gotMembers)
{
 if not(selObj)
  errorBox("ERROR", "No Object Selected")
 else
  if mbdsDB and not(haveMembers(selObj))
   getMembers(self);
   awaitingData := true;
   waitDialog := new(Dialog);
   runModal(waitDialog, DATAWAIT, self);
  else haveData := "YES";
  endif;
  if haveData = "YES"
   oneMemWin := new(DisplayOneWindow,self,"GladOMMenu",
          "DISPLAY: "+name(selObj),nil);
   addWindow(selObj,oneMemWin);
   start(oneMemWin,selObj,0);
  endif;
 endif
}!!

/*count the number of the describe window opened*/
Def countOpnDscrbWin(self)
{
 ^size(extract(dbSchema,{using(obj) obj.aDscrbWin}))
}!!

/*initialize the color table.  this method
 is called from the new method*/
```

```
Def init(self)
{
 colorTable := new(ColorTable,10);
 set(colorTable);
 logOrg :=0@0; /*need dummy assigment,so initial call
            to checkForViolation via reSize works*/
 boundRect:=new(Rect);
 setObjRectSize(self);
 init(self:MyWindow)
}!!

Def rButtonRelease(self,wp,point | tmpObj)
{
 if (tmpObj := objSelected(self,dPtoLP(self,point)))
  /*an object is clicked with rbutton*/
   if tmpObj <> selObj
    if color(tmpObj) == WHITE_COLOR
     errorBox("Wrong Button??",
           "Use LEFT button to select an object")
    else
     errorBox("E R R O R",
           "RIGHT button clicked object is not"+CR_LF+
           "the selected (bold-lined) object")
    endif
   else /* = selObj */
    closeOpenWindows(selObj);
    if not(referenced(selObj))
     /*unshade it if not referenced by other objects*/
     avail(colorTable,color(selObj));
     setColor(selObj,WHITE_COLOR)
    endif;
    /*now unselect it*/
    regBorder(selObj);
    hDC := getContext(self);
    setLPtoDP(self,hDC);
    display(self,selObj,hDC);
    releaseContext(self,hDC);
    selObj := nil
   endif
 endif
}!!

/*list the members (ie instances) of the
  selected object*/
```

```
Def listMembers(self | win, gotMembers)
{
 if not(selObj)
  errorBox("ERROR!","No object is selected")
 else
  if mbdsDB and not(haveMembers(selObj))
   getMembers(self);
   awaitingData := true;
   waitDialog := new(Dialog);
   runModal(waitDialog, DATAWAIT, self);
  else haveData := "YES";
  endif;
  if haveData = "YES"
   win := new(ListMemWindow,self,"GladLMMenu",
      "BROWSE: "+name(selObj),nil);
   addWindow(selObj,win);
   start(win,selObj);
  endif;
 endif
}       !!

/*queries the database*/
Def query(self)
{
 errorBox("query","")
}!!

Def help(self|aStr)
{aStr :=asciiz("Data Manipulation Window");
 pcall(Lib.procs[#GUIDANCESETCONTEXT],HGuide,
 IP(aStr),1);
 freeHandle(aStr);
}!!

/*describe the structure of the selected object*/
Def describe(self | describeWin)
{
 if not(selObj)
  errorBox("ERROR", "No Object Selected")
 else
  describeWin := new(DescribeWindow,self,nil,
            "STRUCTURE OF: "+name(selObj),nil);
  addWindow(selObj,describeWin);
  start(describeWin,selObj)
```

```
  endif;
}!!

/*left button is released*/
Def lButtonRelease(self,wp,point| aLPt)
{
  select
    case selObj and not(objMoved)
    /*an object was not moved, so select it*/
    is
      if prevObj /*unbold the bolded border*/
        regBorder(prevObj);
        /*unshade it if has no opened windows
          and not referenced by other objects*/
        if not( anyOpenWindow(prevObj) or referenced(prevObj) )
          avail(colorTable,color(prevObj));
          setColor(prevObj,WHITE_COLOR)
        endif;
        display(self,prevObj,hDC)
      endif;
      if color(selObj) = WHITE_COLOR
        /*not referenced in another's describe window,
          so assign it a color*/
        setColor(selObj,nextBrushColor(colorTable))
      endif;
      thickBorder(selObj);
      display(self,selObj,hDC)
    endCase

    case selObj and objMoved
     /*an object was just moved, so don't select it*/
     /*adjust boundRect and scroll bars accordingly*/
     is
      display(self,selObj,hDC);
      selObj := prevObj;
      if adjBoundRect(self)
        setScrollRanges(self);
        setScrollPos(self,SB_HORZ,x(logOrg));/*need these when*/
        setScrollPos(self,SB_VERT,y(logOrg));/*bars reappear*/
        repaint(self)
      endif
    endCase
  endSelect;
  releaseContext(self,hDC)
```

```
}!!

/*left button is pressed; check if the cursor is within
  the object rectangle. If yes get ready to move or
  select it*/
Def lButtonDown(self,wp,point | aLPt)
{
  objMoved := nil;
  if selObj
    /*remember it if some object is currently selected*/
    prevObj := selObj
  endif;
  aLPt:= dPtoLP(self,point);
  if (selObj := objSelected(self,aLPt))
    prevCurPt := aLPt
  endif;
  hDC := getContext(self);
  setLPtoDP(self,hDC)
}!!

/*detects whether the cursor is in the object rect*/
Def objSelected(self,cursorPt)
{
  do (dbSchema,{using(obj)
            if containedIn(obj,cursorPt)
              ^obj /*return the selected obj*/
            endif });
  ^nil
}!!

/*draws the diagram. called by the show method
  via update method which sends WM_PAINT */
Def paint(self,hdc)
{
  setLPtoDP(self,hdc);
  do (dbSchema, {using(obj) display(self,obj,hdc)})
}!!

/*gets the meta data of db to be opened,
  initialize other instance variables*/
Def loadSchema(self, aSchemaFile | aFile, anObj)
{
  aFile := new(TextFile);
  setName(aFile,aSchemaFile);
```

```
 open(aFile,0); /*read-only*/
 dbSchema := new(OrderedCollection,10);
 anObj := new(GladObj);
 loop
 while get(anObj,aFile,rectSize)
  add(dbSchema,anObj);
  anObj := new(GladObj)
 endLoop;
 close(aFile)
}!!

Def addMember(self)
{
 errorBox("add member","")
}!!

Def deleteMember(self)
{
 errorBox("delete member","")
}!!

Def modifyMember(self)
{
 errorBox("modify member","")
}!!
```

B. GLADWINDOW CLASS (MODIFIED)

```
/* top-level display window for GLAD */!!

inherit(MyWindow, #GladWindow,
#(dbList            /* DBDialog*/
openedDBs           /*orderedcollection of dbnames*/
mbdsOpen            /* true if an MBDS database has been opened */), 2, nil)!!

now(GladWindowClass)!!

now(GladWindow)!!

Def close(self)
{ if dbList /*something is loaded*/
   updateDbsFile(dbList)
 endif;
```

```
 close(self:Window)
}!!

/* Initialise a call to Guidance */
Def  initGuidance(self|aStr,aString)
{Lib := new(Library);
 Lib.name := "Gydance.exe";
 add(Lib, #GUIDANCEINITIALISE, 0, #(0 0 1 1 0 0));
 add(Lib, #GUIDANCESETCONTEXT, 0, #(0 1 0));
 add(Lib, #GUIDANCETERMINATE, 0, #(0));
 load(Lib);
 aString := "GLAD";
 aStr := "index.gui";
 HGuide :=pcall(Lib.procs[#GUIDANCEINITIALISE],
 HInstance, handle(self), lP(aString),
 lP(aStr), 1,1);
}!!

/*initialize the variables; remove the scroll bar*/
Def init(self)
{
 dbList := new(DBDialog);
 mbdsOpen := false;
 openedDBs := new(OrderedCollection,2);
 setScrollRange(self,SB_VERT,0,0);
 setScrollRange(self,SB_HORZ,0,0);
 initMenuID(self)
}!!

/*adds a new database name to a DBDialog object*/
Def addDb(self, dbName)
{
 addDb(dbList,dbName)
}!!

Def modifyDb(self | dbName, dDWin)
{
 setState(dbList, OPEN_DB);
 if runModal(dbList,OPNDBLIST,self) == OPEN_DB

   dbName := getSelDb(dbList);
   if subString(dbName, 0, 4) = "MBDS"
     errorBox("ERROR","MBDS Databases cannot be modified from within" +
                "GLAD at this time.");
```

```
   else if size(extract(openedDBs,{using(db) db = dbName})) > 0
    errorBox("ERROR","Database '"+dbName+"' is already opened. "+
                "Cannot open more than one data definition or "+
                "data manipulation window for a single database.")
    else /*okay, open DD window*/
     add(openedDBs,dbName);
     dDWin := new(DDWindow,self,"GladDdlMenu",
                "Modify Database: "+dbName,nil);
     startWithExistingDb(dDWin,dbName)
    endif
   endif
  endif
}!!

Def initMenuID(self)
{
  menuID := %Dictionary ( 1->#makeNewDb
                    2->#modifyDb
                    3->#openDb
                    4->#removeDb
                    950->#topHelp
                    6->#close )
}!!

/*create it as overlapped window; need for stand-alone appl*/
Def create(self,par,wName,rect,style)
{
  ^create(self:MyWindow,nil,wName,rect,WS_OVERLAPPEDWINDOW)
}!!

Def removeDb(self)
{
  setState(dbList,REMOVE_DB);
  runModal(dbList,RMVDBLIST,self)
}!!

Def openDb(self | dMWin , dbName)
 {
  setState(dbList,OPEN_DB);
  if runModal(dbList,OPNDBLIST,self) == OPEN_DB

    dbName := getSelDb(dbList);
    select
```

```
    case size(extract(openedDBs,{using(db) db = dbName})) > 0
     is errorBox("ERROR","Database '"+dbName+"' is already opened. "+
                 "Cannot open more than one data definition or "+
                 "data manipulation window for a single database.")
    endCase

    default /*okay, open DM window*/
     if subString(dbName, 0, 4) = "MBDS"
      mbdsOpen := true;
     endif;
     add(openedDBs,dbName);
     dMWin := new(DMWindow, self, "GladDmlMenu",
              "Data Manipulation: "+dbName,nil);

     start(dMWin,dbName);
    endSelect
  endif
}!!

Def topHelp(self|aStr)
{aStr :=asciiz("GLAD WINDOW");
 pcall(Lib.procs[#GUIDANCESETCONTEXT],HGuide,
 IP(aStr),1);
 freeHandle(aStr);
}!!

Def makeNewDb(self | inpDbNameDlg, dbName, dDWin)
{
 inpDbNameDlg := new(InputDialog, "Create Database",
            "Enter the name for a new database ", "");

 if runModal(inpDbNameDlg, INPUT_BOX, self) == IDOK

   dbName := getText(inpDbNameDlg);
   if nameExist(dbList,dbName)  /* Database name already exists */
    errorBox("ERROR!", "Database '" + dbName + "' already exists!")
   else    /*  okay, can use this dbName */
         /*  start the DDwindow */
    dDWin := new(DDWindow,self,"GladDdlMenu",
              "Define Database: "+dbName+" (*NEW*)",nil);
    startWithNoDb(dDWin,dbName)
   endif
  endif;
```

```
}!!
```

C. GLADOBJ CLASS (MODIFIED)

```
/* for storing Glad objects such as emp, dept, etc. */!!

inherit(Object, #GladObj,
#(name
rect          /*drawing rect for the ojbect */
color         /*to fill the box when selected*/
nesting       /*true if it is a generalized object*/
attributes    /*collection of name, class, and type(U or S or C)*/
refCnt        /*reference count*/
thickBorder   /*true if most recently selected object*/
memberFile    /*contains tuple*/
assocWindows      /*collection of its opened windows*/
templateFile      /*file containing display format*/
haveMembers       /* Do we have the latest data file for the object? */), 2, nil)!!

now(GladObjClass)!!

now(GladObj)!!

/* Sets haveMembers variable (whether we have latest MBDS data) to correct
state */
Def  setHaveMembers(self, state)
{
 haveMembers := state;
}!!

/* Have we got the current data from an MBDS file? */
Def  haveMembers(self)
{
 ^haveMembers
}!!

/*check if windowType window is opened for this glad object*/
Def hasSibling(self,windowType)
{
 do(assocWindows,{using(win) if class(win) = windowType ^win endif});
 /*not found*/
 ^nil
}!!
```

```
Def templateFile(self)
{
 if templateFile = ""
  ^nil
 else
  ^templateFile
 endif
}!!

/*is self nested?*/
Def nesting(self)
{
 ^nesting
}!!

/*reset the attributes of self to attrList from DDWindow*/
Def setAttr(self,attrList)
{
 attributes := attrList
}!!

Def reDefine(self,newName,newNesting)
{
 name := newName;
 nesting := newNesting
}!!

/*check whether the self's name is in the OrderedCollection of GladObjects*/
Def nameAlreadyUsed(self,dbSchema)
{
 do(dbSchema,{using(obj) if name(obj) = name /*of self*/ ^true endif});

 ^nil
}!!

/*set the values for newly created object. this is called from DDWindow*/
Def initialize(self,newName,nest,newRect)
{
 name := newName;
 if nest=NESTED then nesting := new(OrderedCollection,10) endif;
 rect := newRect;
 refCnt := 0;
 color := WHITE_COLOR
```

```
}!!

/*save the self's attributes to the file*/
Def saveAttr(self, aSchemaFile)
{
 do(attributes,{using(attr) write(aSchemaFile,attr[NAME]+"&"+
                                  attr[CLASS]+"&"+
                                  attr[LENGTH]+"&"+
                                  attr[USER_DEF]+"&"+CR_LF)});
 write(aSchemaFile,"&&"+CR_LF)
}!!

/*save the self's nested objects to the file*/
Def saveNesting(self, aSchemaFile)
{
 do(nesting, {using(aGladObj) save(aGladObj,aSchemaFile)});
 write(aSchemaFile,"@@"+CR_LF)
}!!

/*save the self to the file*/
Def save(self, aSchemaFile)
{
 write(aSchemaFile,name+CR_LF);  /*obj name*/
 write(aSchemaFile,asString(left(rect))+"@"+
           asString( top(rect))+CR_LF); /*rect location*/

 saveAttr(self,aSchemaFile);

 memberFile := memberFile cor " "; /*if undefined, save blank*/
 write(aSchemaFile,memberFile+CR_LF);

 templateFile := templateFile cor " ";
 write(aSchemaFile,templateFile+CR_LF);

 if nesting /*save the nested objects*/
  write(aSchemaFile,"G"+CR_LF);
  saveNesting(self,aSchemaFile)
 else
  write(aSchemaFile,"N"+CR_LF)
 endif
}!!

Def removeWindow(self,win)
{
```

```
  remove(assocWindows,find(assocWindows,win))
}!!

Def name(self)
{
  ^name
}!!

Def rect(self)
{
  ^rect
}!!

Def memberFile(self)
{
  ^memberFile
}
!!

Def addWindow(self,win)
{
  add(assocWindows,win)
}!!

Def attributes(self)
{
  ^attributes
}!!

Def closeOpenWindows(self)
{
  do(assocWindows, {using(win) close(win)})
}!!

/*obj is moved, adjust its rect*/
Def setNewRect(self, point, prevPt)
{
  offset(rect,x(point)-x(prevPt),
         y(point)-y(prevPt))
}!!

Def thickBorder(self)
{
  thickBorder := true
```

```
}!!

Def regBorder(self)
{
 thickBorder := nil
}!!

Def setColor(self, aColor)
{
 color := aColor
}!!

Def color(self)
{
 ^color
}!!

/*see if self is referenced by others
 describe windows */
Def referenced(self)
{
 ^(refCnt > 0)
}!!

/*see if self has any opened window*/
Def anyOpenWindow(self)
{
 ^size(assocWindows) <> 0
}!!

Def containedIn(self,point)
{
 ^(left(rect) <= x(point) and
    x(point) <= right(rect) and
   top(rect) <= y(point) and
    y(point) <= bottom(rect))
}!!

/*returns an inner box for a generalized object*/
Def nestedRect(self | tmpRect)
{
 tmpRect:=new(Rect);
 init(tmpRect,left(rect),top(rect),right(rect),bottom(rect));
 ^inflate(tmpRect,-5,-5)
```

```
}!!

/*gets the nested objects and assign them to
  self's nesting */
Def getNesting(self, aSchemaFile, rectSize | anObj)
{
  nesting := new(OrderedCollection,5);
  anObj := new(GladObj);
  loop
  while get(anObj,aSchemaFile,rectSize)
    add(nesting,anObj);
    anObj := new(GladObj)
  endLoop
}!!

/*reads in the next object from the schema file.
  rectSize is Point with width@height */
Def get(self, aSchemaFile, rectSize  | pt)
{
  if ( (name := readLine(aSchemaFile)) <> "@@")
    /*there's more*/
    haveMembers := false;
    pt := asPoint(readLine(aSchemaFile));
    rect:= new(Rect);
    init(rect,x(pt),y(pt),x(pt)+x(rectSize),y(pt)+y(rectSize));
    attributes:= getAttr(self,aSchemaFile);
    memberFile:= readLine(aSchemaFile);
    templateFile:=readLine(aSchemaFile);
    if (at(readLine(aSchemaFile),0)=='G')
      getNesting(self,aSchemaFile,rectSize)
    endif;
    refCnt := 0;
    assocWindows := new(OrderedCollection,4);
    color := WHITE_COLOR; /*unselected color*/
    /*returns self*/
  else
    ^nil
  endif
}!!

/*get the object attributes */
Def getAttr(self, aSchemaFile | aColl, anAttr, aStr)
{
  aColl := new(OrderedCollection,10);
```

```
 loop
 while ( (aStr := readLine(aSchemaFile)) <> "&&" )
  anAttr := new(Array,4);
  do (over(NAME,USER_DEF+1),
     {using(idx)
       anAttr[idx] :=
          subString(aStr,0,indexOf(aStr,'&',0));
       aStr := delete(aStr,0,size(anAttr[idx])+1)});
    add(aColl,anAttr)
 endLoop;
 ^aColl
}!!
```

LIST OF REFERENCES

Comer, D. E., *Internetworking With TCP/IP Principles, Protocols, and Architecture*, Prentice Hall, Inc., 1988.

Duff, C., and others, *Actor Language Manual*, The Whitewater Group, Inc., 1989.

Durant, D., Carlson, G., and Yao, P., *Programmer's Guide to Windows*, SYBEX, Inc., 1988.

Floyd, M., "A Class Act," *Dr. Dobb's Journal*, v. 14, April 1989.

Fore, H. R., *Prototyping Visual Interface for Maintenance and Supply Databases*, Masters Thesis, Naval Postgraduate School, Monterey, California, June 1989.

Hsiao, D. K., *Modern Database System Architectures*, unpublished manuscript, 1989.

Leffler, S. J., and others, *The Design and Implementation of the 4.3BSD UNIX Operating System*, Addison-Wesley Publishing Co., 1989.

Naval Postgraduate School Technical Report NPS52-83-006, *Design and Analysis of a Multi-Backend Database System for Performance Improvement, Functionality Expansion and Capacity Growth (Part I)*, by D. K. Hsiao and M. J. Menon, June 1983.

Naval Postgraduate School Technical Report NPS52-86-011, *The Multi-Lingual Database System*, by S. A. Demurjian and D. K. Hsiao, February 1986.

Naval Postgraduate School Technical Report NPS52-87-026, *The Multi-Model Database System*, by H. Coker and others, June 1987.

Naval Postgraduate School Technical Report NPS52-88-050, *Implementation of Visual Database Interface Using an Object-Oriented Language*, by C. T. Wu and D. K. Hsiao, June 1988.

Pascoe, G. A., "Elements of Object-Oriented Programming," *BYTE*, v. 11, August 1986.

Petzold, C., *Programming Windows*, Microsoft Press, 1988.

Shneiderman, B., *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, Addison-Wesley Publishing Co., 1987.

Wegner, P. "Dimensions of Object-Based Language Design," *OOPSLA '87 Proceedings*, October 1987.

Williamson, M. L., *An Implementation of a Data Definition Facility for the Graphics Language for Database*, Masters Thesis, Naval Postgraduate School, Monterey, California, December 1988.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Director, Project BASIS Naval Data Automation Command Washington Navy Yard Washington, D.C. 20374	1
4.	Curriculum Officer Computer Technology Program, Code 37 Naval Postgraduate School Monterey, California 93943-5000	1
5.	Professor C. Thomas Wu, Code 52Wu Computer Science Department Naval Postgraduate School Monterey, California 93943-5000	10
6.	Marciano Pitargue Vitalink Communication Corporation 6607 Kaiser Drive Fremont, California 94555	1
7.	Tom Chu Computer Science Department Naval Postgraduate School Monterey, California 93943-5000	1
8.	Debbie Gaiser 2739 Eaton Street New Orleans, Louisiana 70131	1
9.	LT Thomas R. Hogan 821 170th Place N.E. Bellevue, Washington 98008	2