NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

THE INSTRUMENTATION OF A KERNEL DBMS FOR THE EXECUTION OF KERNEL TRANSACTIONS EQUIVALENT TO THEIR OBJECT-ORIENTED TRANSACTIONS

by

Robert Eugene Clark Jr.
Necmi Yildirim

September 1995

Thesis Advisor: David K. Hsiao

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1995	Master's Thesis

4. TITLE AND SUBTITLE
The Instrumentation of a Kernel DBMS for the Execution of Kernel Transactions Equivalent to their Object-Oriented Transactions

5. FUNDING NUMBERS

6. AUTHOR(S)
Robert Clark and Necmi Yildirim

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the authors and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The issues addressed in this thesis are to examine whether the data manipulation operations of the kernel database system are capable of supporting the new Object-Oriented Data Model and Language Interface (OODM&L Interface). The data manipulation operations of the kernel database system consist of Retrieve, Delete, Update, and Retrieve-Common. To examine these issues, it is necessary to review the adequacy and inadequacy of these four operations in their ability to carry out object-oriented data manipulation operations in the OODM&L Interface.

A code review of the four operations is needed in order to determine what modifications are required for the kernel to execute object-oriented operations in the object-oriented transaction. Additionally, the code for the communications between the kernel system and the OODM&L Interface is designed and implemented.

The result of this thesis implements the modified kernel operations and documents how the object-oriented data manipulation is carried out in the newly modified kernel database system. These modifications range from changing certain variables in the kernel database system to rewriting lines of C code in modules of the kernel database system. Secondly, this thesis implements the required communication capability between the kernel database system and the object-oriented data model and language interface. The communication implementation is accomplished with four new functions comprised of 172 lines of C code written into the kernel system code. This additional code enables data to be passed between the object-oriented interface and the kernel database system.

14. SUBJECT TERMS
Kernel Database System
Mutimodel and Multilingual Database System
Object-Oriented Data Model and Language (OODM&L)

15. NUMBER OF PAGES
123

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Approved for public release; distribution is unlimited

THE INSTRUMENTATION OF A KERNEL DBMS FOR THE EXECUTION OF KERNEL TRANSACTIONS EQUIVALENT TO THEIR OBJECT-ORIENTED TRANSACTIONS

Robert Eugene Clark Jr.
Lieutenant, United States Navy
BS, Old Dominion University, 1989
and
Necmi Yildirim
Lieutenant Junior Grade, Turkish Navy
BS, Turkish Naval Academy, 1989

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1995

Authors:

Robert Eugene Clark Jr.

Necmi Yildirim

Approved by:

David K. Hsiao, Thesis Advisor

C. Thomas Wu, Co-Advisor

Ted Lewis, Chairman,
Department of Computer Science

ABSTRACT

The issues addressed in this thesis are to examine whether the data manipulation operations of the kernel database system are capable of supporting the new Object-Oriented Data Model and Language Interface (OODM&L Interface). The data manipulation operations of the kernel database system consist of Retrieve, Delete, Update, and Retrieve-Common. To examine these issues, it is necessary to review the adequacy and inadequacy of these four operations in their ability to carry out object-oriented data manipulation operations in the OODM&L Interface.

A code review of the four operations is needed in order to determine what modifications are required for the kernel to execute object-oriented operations in the object-oriented transaction. Additionally, the code for the communications between the kernel system and the OODM&L Interface is designed and implemented.

The result of this thesis implements the modified kernel operations and documents how the object-oriented data manipulation is carried out in the newly modified kernel database system. These modifications range from changing certain variables in the kernel database system to rewriting lines of C code in modules of the kernel database system. Secondly, this thesis implements the required communication capability between the kernel database system and the object-oriented data model and language interface. The communication implementation is accomplished with four new functions comprised of 172 lines of C code written into the kernel system code. This additional code enables data to be passed between the object-oriented interface and the kernel database system.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

The completion of this thesis would not have possible without the guidance, support and understanding of many individuals. We would like to take this opportunity to express our sincerest thanks to those who helped us prepare this thesis. We thank Dr. David K. Hsiao for his expertise and insight. Secondly, we thank Dr. C. Thomas Wu for his assistance with the coding and debugging of this project. A special thanks goes out to our families Jennifer, Robert, and Hulya for their patience, encouragement and loving support throughout this process. Finally, we would like to thank our parents Robert and Betty Clark and Ali and Solmaz Yildirim for their love, inspiration and leadership. Without the help and moral support of the above people and others, this work would not have been possible.

I. INTRODUCTION

For years databases and database systems have grown in size and complexity. Now, various heterogeneous databases and database systems are required to communicate with one another.

With the development of the object-oriented programming language, there is the evolution of the object-oriented database and the Object-Oriented Database System (O-ODBMS). Our goal is to design and implement a database system that supports the object-oriented data language (O-ODL). This is the Kernel Database System. It is the underlying database, software, and hardware on which all the model/language interfaces must operate. One such model/language interface is the object-oriented/ O-ODL interface. Thus, the user can access the object-oriented database with the object-oriented transactions via the interface. On the other hand, the kernel database system is organized in an attribute-based format where all the data are stored as attribute-value pairs.

In addition to the object-oriented model/O-ODL interface, there are the functional/ Daplex, the relational/SQL, the network/CODASY-DML, and the hierarchical/DL/I interfaces. All the interfaces use the same attribute-based data and transactions for the stored database.

The utilization of a particular model/language interface yields two important advantages. The first advantage allows a user to access the database using a particular language preferred by the user. For example, if a user is comfortable using Daplex (a functional data language), then the user may use the same language for processing the user's functional database. If the object-oriented user desires to access data in an object-oriented way, then the user should be allowed to do so. This is achieved through the use of the object-oriented interface on the same kernel database system. There is no need to utilize a different database system. This is efficient, since there is only one (kernel) database system, instead of a multitude of heterogeneous database systems. Further, it allows all the heterogenous databases and system software supported on the same kernel database system to communicate with one another.

This thesis demonstrates how the primary kernel functions are used by the object-oriented interface in order to manipulate data. It also documents modifications that have been made in order for the interface to work properly and reliably. The kernel DBMS is responsible for maintaining the entire database. The model/language interface merely works with and through the kernel so a user can continue to remain in the user's familiar data model and data language. In our case, they are object-oriented.

For implementing the object-oriented interface, there is the creation of a Real-Time Monitor. This monitor is the subject of a thesis by Erhan Senocak [SenocakSep95]. The Real-Time Monitor is used for program execution of object-oriented transactions. The transactions are received by the object-oriented interface and sent from the Real-Time Monitor of the interface to the Kernel System in executable format. Communications and interactions of the Kernel Database System and the Real-Time Monitor are discussed in detail in subsequent chapters.

This thesis also outlines the specific problems found within the primary functions in the kernel database system. The objectives are to review the existing code and modules for the primary system operations to ensure that they support the real-time monitor's demand correctly. Second, the development of the communications and interactions of the Real-Time Monitor and the Kernel System is also discussed. Lastly, the documentation is provided for all those modifications that are required for the kernel to work properly with the new interface via the monitor in order to complete a given object-oriented transaction.

In the remaining chapters of this thesis, we first describe the Kernel Database System in Chapter II. This includes the kernel data model, the kernel data language and kernel DBMS. In addition, we describe the architecture of the kernel system and how it interacts with all the interfaces. In Chapter III, we review the existing Kernel Database Primary Functions. In Chapter IV, we discuss all the communications with the Real-Time Monitor. The discussion also includes the relationship of the compiler to the monitor and the relationship of the monitor to the Kernel system. Chapter V is used to discuss modifications made to the Kernel functions in order to support the monitor properly. In

Chapter VI, we discuss our accomplishments and our recommendations for future research. There is also a summary of them.

II. THE KERNEL DATABASE SYSTEM

A. THE KERNEL DATA MODEL, THE KERNEL DATA LANGUAGE, AND THE KERNEL DBMS

The kernel database system used for the thesis research is the Multi-Model/Multi-Lingual Database System (M^2DBMS). This kernel system supports all the data model/ language interfaces. The existing interfaces include hierarchical, relational, network, functional, and object-oriented. These interfaces interact with the kernel data systems thereby allowing a user to access data in the user's database while remaining within the user's own model and language environment. The kernel and interfaces are shown in Figure 1.

The underlying kernel data model and kernel data language are the attribute-based data model and attribute-based data language. With this data model, data are stored as attribute-value pairs. Each attribute-value pair consists of an attribute name and an attribute value. The *attribute* is the name of the domain of values. A *record* is a set of attribute-value pairs which have the following constraints: (1) No attribute is repeated in a record. (2) An attribute cannot have more than one value in a record. (3) Every record must have at least one key. See [HSIao Mar95].

The justification for not repeating an attribute in a record is twofold. First, if an attribute is repeated then we have duplicate information about that record in the database. This duplication is wasteful and can lead to inconsistency problems. Second, if an attribute is repeated and it is assigned with different values, then the system cannot determine what is the correct value to return or operate on. For example, assuming the record contained two attributes such as <FNAME, Robert> and <FNAME, David>. If a retrieval request for the first name (FNAME) is made of the database, the system will not know which one of the two values to return. Similarly, if a user wishes to update an attribute in the record, again the system will not know which attribute-value pair to act on.

The justification for the second constraint is the following. If an attribute is allowed to have more than one value in it, then the same problem arises when attempting to query

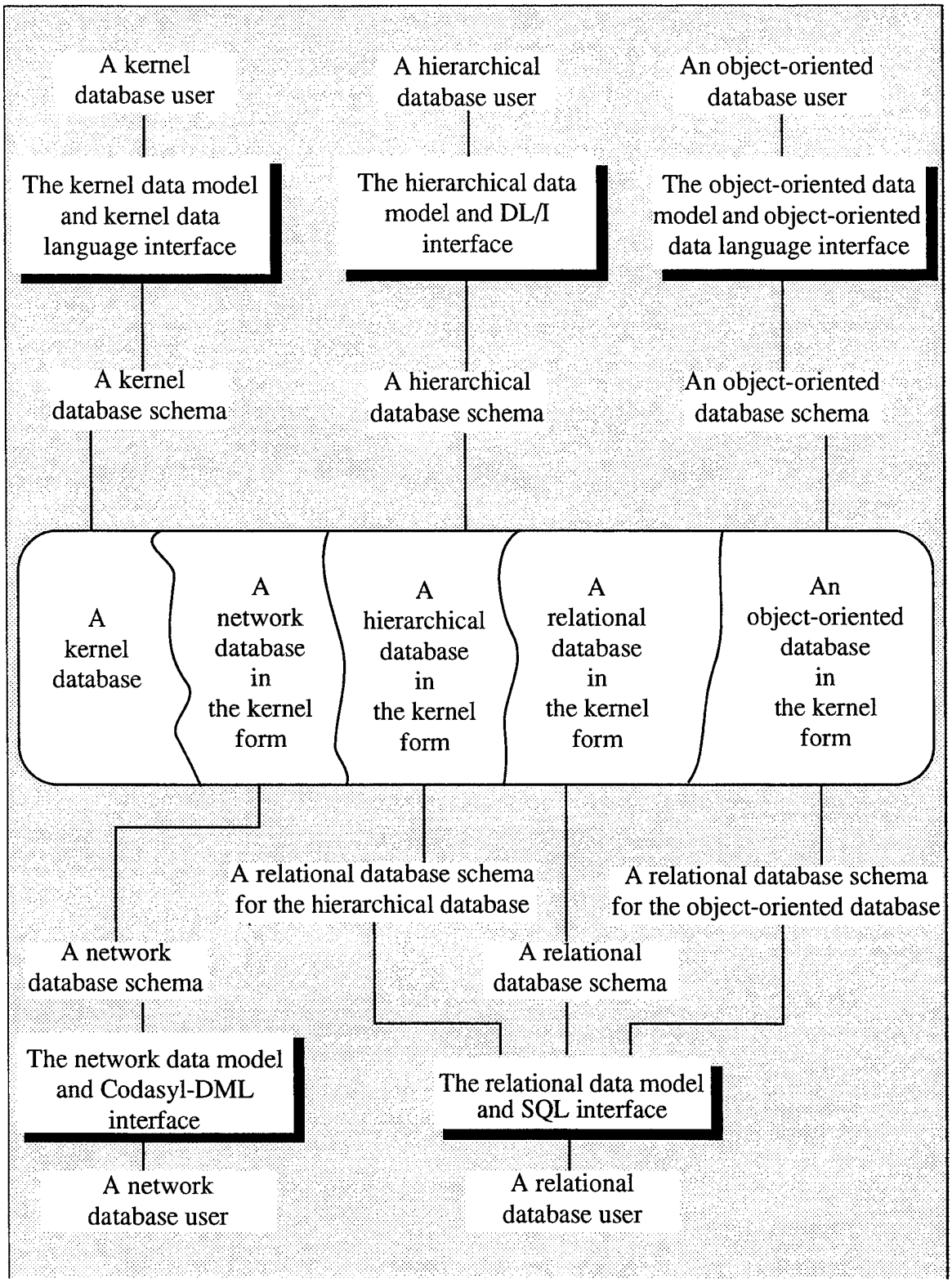


Figure 1. Multimodel, Multilingual, and Cross-Model Access Capabilities of MDBS

the system for a particular attribute value. The system cannot determine which value to return or modify for a given query. For example, if the system contained a record with the following attribute-value pair, <ZIPCODE, 93940> and <ZIPCODE, 93950>, the system would not be able to discern which value to act on for any given transaction.

The third constraint states that each record will have at least one key. This key is used to identify a record from all other records in the database. This key value allows a user to conduct searches of the database more efficiently by limiting the number of records searched for a transaction. The kernel database system allows for a record to have more than one key, if they are required to *uniquely* identify that record.

The attribute domain can be an integer or a string data type. For example, a user can describe the record using the following attribute-value pairs enclosed in parentheses: (<FNAME, Robert>, <MI, E>, <LNAME, Clark>). In this example, all of the attributes are declared as the string data type.

Within the kernel system, there exists several files that describe the data stored in the database, commonly referred to as *metadata*. These files are the template file, or .t files and the descriptor file, or .d files. The *template file* lists the name of the database and all the template names within that database. The template is a subdatabase. The template name is the name of a subdatabase *name* which is the value of TEMP. The template file further describes the database. This description is more detailed and lists the data types for each attribute within the database. For example, *FNAME* is of the string type and *ZIPCODE* is of the integer type. In addition to the data-type declarations, the system recognizes different attribute types. The attribute types are type A, type B, and type C.

A type-*A* attribute partitions its values into value ranges. For example, the value for *ZIPCODE* is defined as an integer between 10000 and 99999. If a user enters a value outside this range, then the system will not accept it.

A type-*B* attribute is one of distinct values. For example, the attribute *MI* (middle initial) has to be one of the letters in the alphabet.

A type-*C* attribute is like a type-*B* attribute in that it is distinct. However, the type-*C* attribute is entered by the user in real time. By utilizing these attribute types we have three different kinds of keys [Hsiao Dec91].

The kernel data language or attribute-based data language (ABDL) consists of five primary operations. These operations are Insert, Retrieve, Update, Delete, and Retrieve-Common. These primary operations are, although primitive, powerful. The ABDL primary operations are discussed in detail in Chapter III.

In addition to the primary operations, several other operators exist to manipulate sets of records. These operators allow a user to further define a transaction or perform functions on a set of records. These are the relational and aggregate operators. The relational operators are as follows; = for EQUAL, /= for NOT-EQUAL, > for GREATER-THAN, >= for GREATER-THAN-OR-EQUAL, < for LESS-THAN, and <= for LESS-THAN-OR-EQUAL.

An example of a relational operation would be;

RETRIEVE((TEMP=Address)and(ZIPCODE>=93940))(State)

This query results in finding all the states where the zipcode is greater than or equal to 93940. The names of the states that meet this criteria would be displayed to the screen. Thus, the pair of parentheses enclosed an attribute name is another operator of ABDL, known as the *target list.*

The aggregate operations are utilized to perform certain operations on set of records. For example, if a user wishes to retrieve the maximum value for a zipcode, then the following transaction may be used;

RETRIEVE(TEMP=Address)(MAX(ZIPCODE)).

The same type of transaction may be used to find the minimum value of all the zipcodes by replacing MAX with MIN.

The kernel database system is unique in that it supports several different model/language interfaces. To support these different interfaces, the kernel uses only one data model and data language. This is the attribute-based data model and data language. In utilizing this type of approach, we provide to the user different databases logically and to

facilitate communications among them. In addition, the user can remain in their familiar model/language environment while accessing and processing data from other environments. For example, an object-oriented user can access object-oriented data while a relational user can access relational data.

A database system of this design leads to several advantages. One such advantage is the fact that the database and its data do not need to be duplicated when a new data model/ language interface is added to the system. We only need to create a new interface to interact with the kernel. This approach allows for different database systems to be supported on the same kernel without duplicating the system software and data.

B. THE ARCHITECTURE

The computer architecture for the kernel database system is a multibackend database supercomputer, or MDBS. The design of the MDBS supercomputer is depicted in Figure 2. The MDBS consists of many identical database backends. Each database backend is a combination of a database processor and a database store [Hsiao Dec91]. As shown in Figure 2, the database backends are controlled by a controller. The controller is a combination of hardware and software processes. The controller is a single microprocessor-based computer. The controller interfaces with the user directly or through a general-purpose computer. The controller broadcasts a transaction from the user to all of the backends for parallel execution. It then collects results from all the back-ends and routes them to the user. The controller does little database operation, since we do not want the controller to become a bottleneck of the supercomputer [Hsiao Mar95]. The controller, being the vital link between the user and the backends, requires two basic processes to operate. These processes are known as the Transaction Processing and Post processing. The transaction processing is used for receiving a transaction from the frontend computer, and relaying it to the backends. The postprocessing is used for returning results for the transaction to the user.

When a user executes a transaction on the database, the transaction is sent to the backend through the transaction processor. The transaction can be built with different

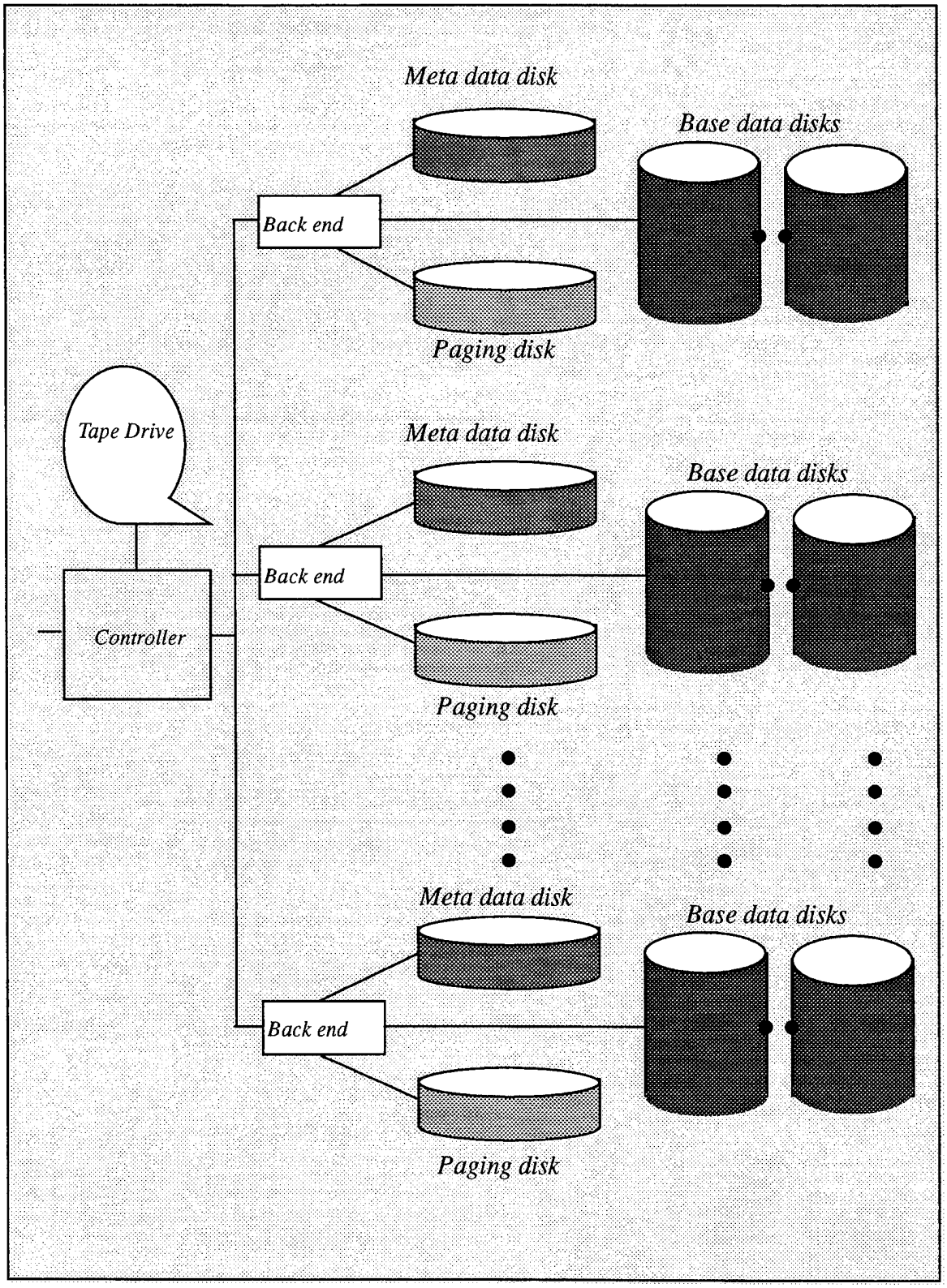


Figure 2. The Multibackend Database Supercomputer

means. One method is to execute a file of previously written transactions (i.e., canned queries). For example, after the database is loaded, the user can chose to execute canned queries that are stored in a file. The file may contain those queries of the database relevant to the user or database. Another method is for a user to develop the on-line query through the model/language interface.

In either case, the transaction or query is sent to the backend as a TrafficUnit. The TrafficUnit is made up of two pointers. The first pointer in the TrafficUnit is the database identifier, or dbid. The dbid is used by the backend to identify which database is to be accessed by the user from the frontend. The second pointer in the TrafficUnit is the transaction or query itself, which is identified with its id, Trafficid. Trafficid consists of a message queue and the ABDL transaction. The message queue is used to identify (1) the user on the frontend called the header.sender and (2) the process that is receiving the transaction, called the header.receiver. A small amount of preprocessing must be accomplished within the TP to correctly translate the transaction into the attribute-based primary database operations.

On the other hand, PP postprocesses database records. Using the example of retrieving the maximum value of zipcodes in the address file (i.e., Template=Address), each backend returns the value for the zipcode. PP then determines which of these values is the largest and then return this result to the user. In Figure 3, we depict the steps involved by the software processes to manipulate the data between the interfaces and the backends. Each process is described in detail in Appendix A.

As shown in Figure 3, an object-oriented model/language interface exists. It's existence owes to our new design and implementation of the kernel database system. This design and implementation effort involves the use of a data-definition-language compiler, a data-manipulation-language compiler, the real-time monitor, and the newly modified kernel database system. All of these components work together in order to execute an object-oriented transaction. In Figure 4, we depict the newly implemented object-oriented interface.

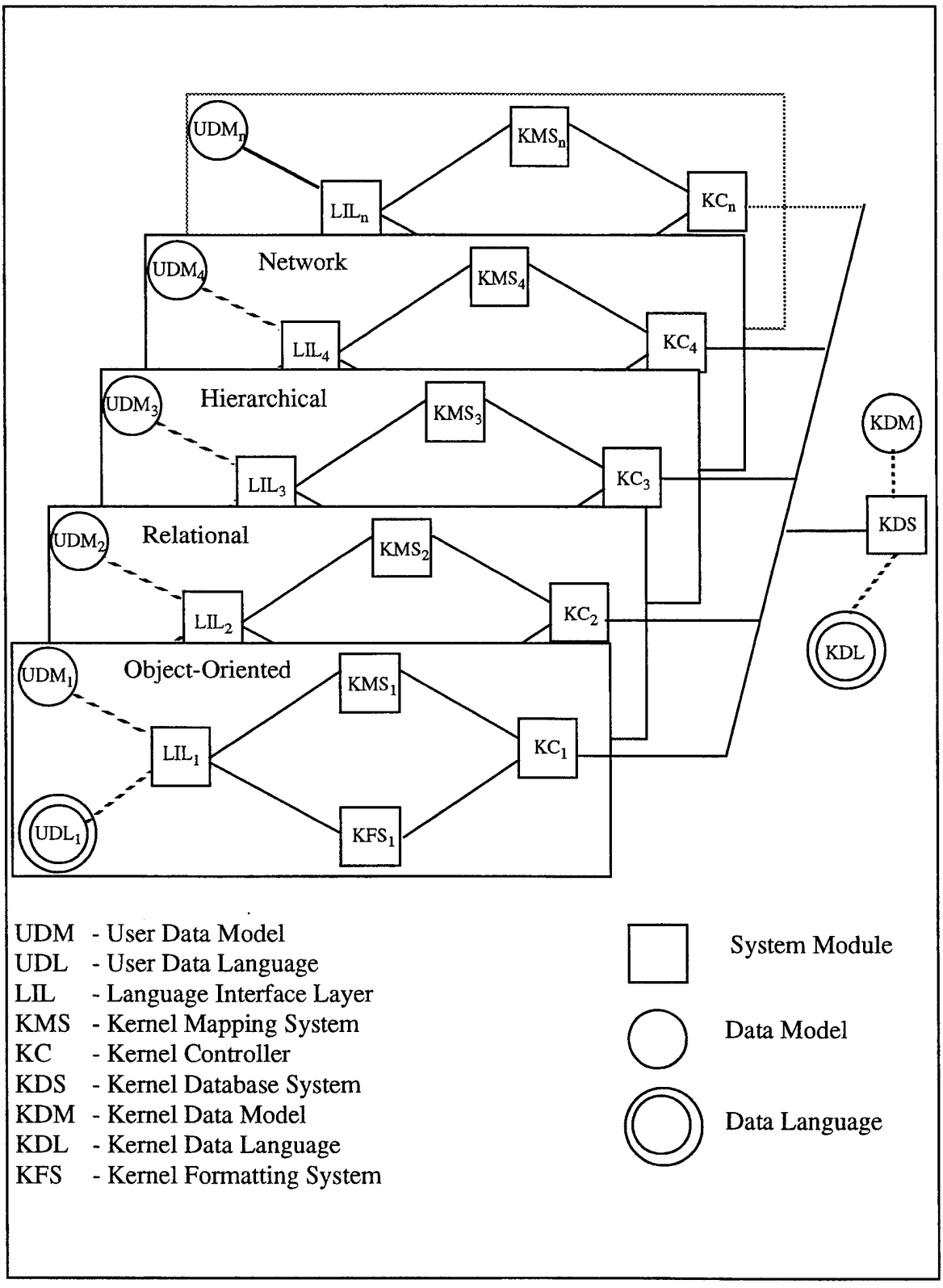


Figure 3. The Multi-model/Multi-lingual Database System

The data-definition-language compiler is the thesis work of Luis Ramirez and Recep Tan [RamTanSep95]. The implementation of the data-manipulation-language compiler is the work of Carlos Barbosa and Aykut Kutlusan [BarKutSep95]. The real-time monitor is the work of Erhan Senocak [Sen Sep95]. The modification of the kernel database system to support these compilers and monitor is the work of this thesis.

As shown in Figure 4, a user enters an object-oriented transaction via the Model/ Language interface layer (Lil). The transaction is then passed on to the data-definition-language and data-manipulation-language compilers. These compilers then take the transaction or query and translate it into an attribute-based query along with the psuedocode that is used by the real-time monitor for transaction or query execution. The transaction or query is then passed to the kernel database system from the real-time monitor. The transaction or query is now in a strictly attribute-based format that the backend can accept and process. Once the kernel receives the translated transaction or query, it executes the proper primary operations to complete the request or transaction. After the data has been retrieved or manipulated, the response is sent back to the real-time monitor for routing to the user.

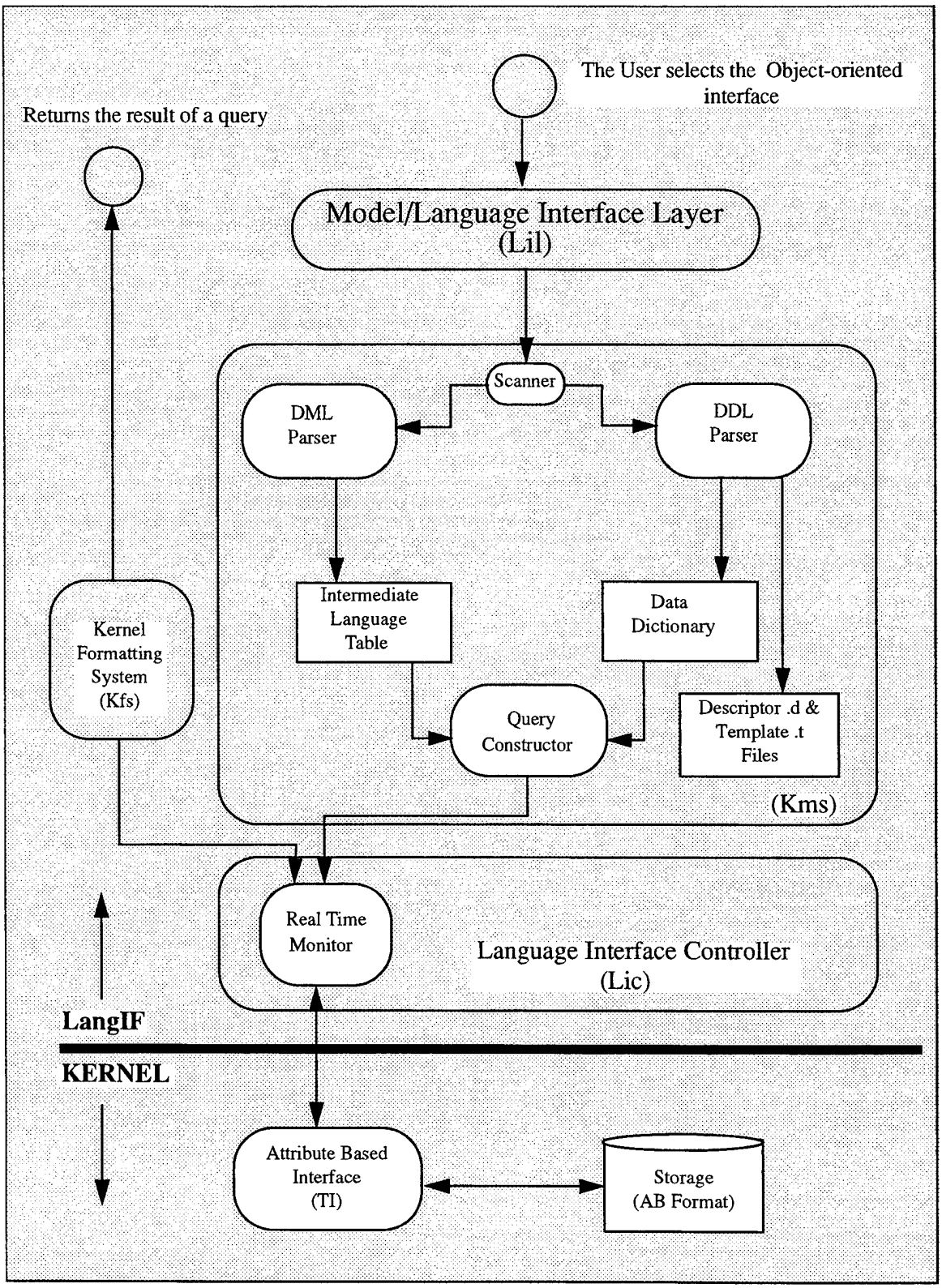


Figure 4. The Object-Oriented Interface

III. THE EXISTING KERNEL DATABASE OPERATIONS

A. RETRIEVE

Created in the Attribute-Based Kernel Database System, templates establish key attributes for each record. The position of the first attribute in the template is by definition the key attribute. If another attribute is to be used as the key attribute, then the use of the BY-clause with the other attribute in the query can overlook the first positional key. The name of the template (e.g., TEMP = *name*) and a set of attribute (e.g., OID, FNAME, MI, LNAME) known as the target list are required in the Retrieve operation. The BY-clause is optional. Together, they are called the query.

Thus, the Retrieve operation takes two arguments: a query and a target list. The query specifies the set of records to be retrieved from the base data and the target list specifies the values to be displayed from the retrieved data. A simple target list simply lists the values of attribute-value pairs whose attribute has been targeted. A complex target list may specify an aggregate function such as AVERAGE over a specific attribute Grade. The output is the average of Grades found in a set of records.

An example of a common Retrieve transaction follows:

RETRIEVE (TEMP = Name) (OID, LNAME, FNAME, MI)

In this particular example, the Object Identifier, or OID is the key attribute. The OID attribute is the first attribute declared after the name of the template in the .t file. This transaction will display the values for the attributes given in the target list. The attributes are sorted by the increasing values of the OID attribute. The remaining attributes follow in the order in which they are listed in the target list. A sample output is depicted below:

<(OID, N1) (LNAME, Clark) (FNAME, Robert) (MI, E)>

<(OID, N2) (LNAME, Hsiao) (FNAME, David) (MI, K)>

<(OID, N3) (LNAME, Yildirim) (FNAME, Necmi) (MI, L)>

It is important to note that the key attribute, as defined by the **.t** file, must be present in the target list in order for it to execute properly. If the key attribute is not listed in the target list of the transaction, the transaction will fail and the system may crash.

However, the order in which it resides in the target list makes no difference. The following example displays values of attributes in the target list and sort the displayed records by values of the OID attribute.

RETRIEVE (TEMP = Name) (LNAME, FNAME, OID, MI)

A sample output is displayed like this:

<(LNAME, Clark) (FNAME, Robert) (OID, N1) (MI, E)>
<(LNAME, Hsiao) (FNAME, David) (OID, N2) (MI, K)>
<(LNAME, Yildirim) (FNAME, Necmi) (OID, N3) (MI, L)>

We note that the record with the lowest value N1 for OID appears first.

On the other hand, if a user desires the displayed records to be ordered on the basis of a different attribute, then the user uses the BY-clause. By appending onto the query with the BY-clause, the user informs the system by which attribute the output records must be sorted. An example of a Retrieve with a BY-clause is as follows:

RETRIEVE (TEMP = Name) (OID, FNAME, MI, LNAME) BY FNAME

This query retrieves records with OID, FNAME, MI, LNAME. However, for display, it lists them in the order of values of FNAME. A sample output is displayed like this:

<(OID, N2) (FNAME, David) (MI, K) (LNAME, Hsiao)>
<(OID, N3) (FNAME, Necmi) (MI, L) (LNAME, Yildirim)>
<(OID, N1) (FNAME, Robert) (MI, E) (LNAME, Clark)>

We notice that the records are listed on the basis of the FNAME attribute and appear in alphabetical order of first names. This ordering is due to the BY-clause. The attribute

of the BY-clause allows the system to sort records in the order of the attribute values. If no attribute is given, i.e., the BY-clause is not used, then the system uses the key attribute which is the first attribute identified in the template.

Whenever submitting a query to the system, the key value must be part of the query. If no BY-clause is used, the key value, by default, is the first attribute listed in the **.t** file after the declaration of the template name. In all cases where the key attribute is not present to in the query, the BY-clause must be present.

Consider a query without OID.

RETRIEVE (TEMP = Name) (LNAME) BY LNAME

This example produces a listing of values (i.e., last names) of LNAME only. The list is sorted alphabetically of based on leading letters of each value (last name) of the LNAME attribute. A sample output is displayed below:

<(LNAME, Clark)>

<(LNAME, Hsiao)>

<(LNAME, Yildirim)>

The records are sorted and displayed in the alphabetical order. This query executes normally, despite the absence of the key attribute, OID. Because the BY-clause is used, the system now has an attribute to sort on. The system does not assume that the first attribute in the query is a key attribute. The system uses the template to determine the key attribute. The portion of the code which implements the BY-clause has been reviewed, and has been determined to work in this way. This code is located in the BE/RECP directory and is to be explained in detail in Chapter V. We also discuss the BE code for Retrieve operation in Chapter V.

In addition to conducting retrievals on single attributes and equality predicates, we have the ability to use relational operations and aggregate functions within transactions. These types of transactions allow a user to further define a set of records that the user

wishes to retrieve. We recall, these relational operations and aggregate functions have been described in Chapter II.

B. DELETE

The Delete operation in the ABDBMS does not remove records from the database. This operation is merely used to *tag* those records that the user wishes to remove from the database. In other words it does not physically remove any record from the database. The Delete operation takes only one argument, a query. The following is a sample Delete transaction for the ABDBMS.

DELETE ((TEMP = Address) and (NUMBER = 144))

After this transaction is executed, the user is prompted for the next transaction. The system does not inform the user that the transaction was completed successfully or not.

In the ABDBMS, the Delete operation is carried out in three steps. Step one - the records which satisfy the query are retrieved from the database data. This step is like the Retrieve operation with the target listing of all the records retrieved. Using the above example, the only records that will be tagged for deletion in the Template Address are those which have the attribute value, 144, for the attribute, NUMBER.

Step two - each retrieved record is tagged for later removal. This step is also known as writing the *deletion tag* into a record. In the database, each record that is active, or has not been previously tagged for deletion has a value of "1" in the deletion-tag field. Once the record has been tagged for deletion, this value changes to "2". This value will be used later by the garbage collection routine to remove all the records tagged with the value "2".

Step three - the record with the deletion tag is placed on the secondary storage where the record originally came from. This step is like the Insertion operation. As stated, no record has been physically removed by this operation. The removal of records with deletion tags is the function of the garbage-collection routine of the system which is carried out in a non-prime-time periodicity.

Using this particular convention for deletion, the ABDBMS allows a user to delete several records swiftly, since the garbage-collection of deleted records is time-consuming. The user only needs to enter one argument, the Template name, and the transaction executes normally. This can be dangerous, if the user has not properly refined the deletion query. The user can easily and unknowingly delete incorrect records from the database. The way to avoid the pitfall is to try the query with the Retrieve operation for verification prior to any use of the query for the deletion. After a record has been tagged for deletion, then the record is no longer accessible to the user, even though the record still exists in the database data.

The user also has the ability to utilize the relational operator, AND, in the Delete query. Again, this allows the user to refine the query and reduce the number of records to be deleted from the database. For example, if a user desires to delete all the records where the value for the ZIPCODE is greater than or equal to 93940, the user may use the following transaction with the relational operator and.

DELETE ((TEMP = Address) and (ZIPCODE >= 93940))

C. UPDATE

In the Attribute-Based Kernel Database System, the Update operation takes two arguments: a query and a modifier. The Update is used to modify records of a database. The query specifies which records of the database to be modified. The modifier specifies how these records to be modified.

The operation is carried out in four steps: Steps one through three are the same as the steps of the Delete operation. However, there is an additional forth step. For each record to be tagged for deletion, this operation makes a copy of the record. The copy is changed by the modifier specified by the user. The modified copy is then entered into the database by the Insert operation as a new record [Hsiao Mar95].

In the ABDBMS, there are basically two types of Updates. One type of Update is simply modifying the value of a particular attribute to a new value. The second type of

Update is to perform some type of arithmetic operation on a particular attribute and store the new value in that record.

For example, if a user wishes to change the value of the FNAME attribute of a record from "Recep" to "Tony" the following transaction would be used.

UPDATE ((TEMP = Name) and (FNAME = Recep)) <FNAME = Tony>

This transaction results in all the records that have the FNAME attribute-value of "Recep" to be changed to "Tony". Again, this modification is made to all the records with no further interaction from the user. The system does not inform the user that the change has been made. The only way the user knows that the change has been made is to query the database for the new information.

A second example of the same type of transaction is the following:

UPDATE ((TEMP = Address) and (ZIPCODE = 93940)) <ZIPCODE = 93950>

This transaction will change all the records in the *Address* template that have 93940 as the ZIPCODE attribute-value pair to the new value of 93950.

As stated, the user has the option to modify attribute using arithmetic operations. These operations consist of addition (+), subtraction (-), multiplication (*), and division (/). These operations are used to modify those attribute-value pairs that consist of integer values. The following is an example of the addition operation. All other arithmetic operations work in the same fashion and are not demonstrated here.

UPDATE (TEMP = Address) <ZIPCODE = ZIPCODE + 10>

This transaction results in the ZIPCODE attribute-value to be incremented by 10 for all the records in the *Address* template. The query portion of the transaction is the TEMP=Address part and the modifier is the ZIPCODE=ZIPCODE+10 part. The system will retrieve all the records in the *Address* template, determine the current value of ZIPCODE for each record and then increase that value by 10. The new value of ZIPCODE is then stored in the record.

The reader should note that the subtraction and division operations require that these operators be proceeded and followed by a space in order for the query to execute properly. The justification for this convention is to allow for hyphens (-) and slashes (/) within attribute-values. This was implemented earlier by other thesis students who designed a database for various military equipment which needed these characters to define certain attributes. This is discussed further in Chapter V.

D. RETRIEVE-COMMON

The Retrieve-Common operation consists of two Retrieve operations with a common clause. The common clause specifies an attribute of the record set determined by the first Retrieve operation and an attribute of the record set determined by the second Retrieve operation. The clause requires that the output of the operation is a set of records each of which is composed of two records- one from the first record set and the other from the second record set such that these two records have common attribute values for the attributes specified in the common clause. Of course, each output record can be reduced in size if a target list is used in either Retrieve operation[Hsiao Mar95]. A sample Retrieve-Common transaction follows.

RETRIEVE ((TEMP = Name) and (OID = N4)) (OID, FNAME, LNAME)
COMMON (OID, OID_S)
RETRIEVE (TEMP = Course_stu) (OID, OID_S)

This Retrieve-Common transaction searches through the records in the *Name* and *Course_stu* templates and displays the OID of the Courses that the student is taking. The first step is to retrieve the record which has an OID attribute-value of "N4" in the *Name* template. The next step is to retrieve the records from the *Course_stu* template which have the same value (N4) for the OID_S attribute. This is the "Common" attribute between the two templates and records.

Despite the usefulness of this type of transaction in the attribute-based system, our design does not require that we use this query. With the introduction of the real-time monitor, we can simply use multiple Retrieve queries to accomplish the same thing. For

example, if a user wants to know what classes a student is taking, they simply retrieve the *Name* record and the *Course_stu* record then the real-time monitor will do the equi-join or cross multiplication on the two sets of records. The common values between the two sets would then be displayed on the screen.

IV. THE COMMUNICATIONS WITH THE REAL-TIME MONITOR

A. THE RELATIONSHIP OF THE COMPILER AND THE REAL-TIME MONITOR

The implementation of an Object-Oriented interface with the kernel database system requires the utilization of a real-time monitor. The real-time monitor is used for program execution of an object-oriented transaction or query from the object-oriented interface. As stated, the compiler is the work of Carlos Barbosa and Aykut Kutlusan [BarKut Sep 95]. The compiler receives the object-oriented transaction or query from the user. This transaction is then compiled and formatted into the attribute-based format. The formatting function is accomplished through the use of a *query constructor*. The query constructor reads in the transaction or query from the user and converts it into an attribute-based transaction or query. This format is required in order for the kernel system to understand the transaction or query.

The query constructor accesses an intermediate-language table, the data dictionary, and the symbol table which enables the actual attribute-based transaction or query to be developed. This final part of the compiler constitutes the bulk of the work. Since data being queried or manipulated are not accessible to the compiler, a method of translating individual lines and even parts of each line, must be developed. This process must be done to accommodate for the differences between object-oriented and attribute-based syntaxes and data structures. Once the transaction or query is compiled and constructed, it is then passed to the real time monitor for execution.

In addition, the compiler also provides additional *pseudocode* to the transactions. The pseudocode is comprised of symbols the real-time monitor uses to complete the query execution. This pseudocode is required so the real-time monitor can read the transaction and queries properly, and understand what the user wishes to accomplish (i.e. what the transaction or query is to do). This pseudocode will be used by the real-time monitor to

execute the transaction in the proper order. See [Sen Sep95] for a complete description of all pseudocode symbols and a detailed explanation of the real-time monitor.

Once the transaction and pseudocode are combined, the queries are then stored in a simple text file, *query_f*. The next step is for the Language Interface Layer to make a function call to the real-time monitor with *rtm_exec()*. The *rtm_exec()* function takes *path_name* for the text file as an argument.

The next step is for the real-time monitor to read the input text file, *query_f*. The input text file contains the data stated above. The real-time monitor is responsible for reading this text file and sending these transactions to the backend for further processing. The transactions are sent to the backend one at a time.

B. THE RELATIONSHIP OF THE MONITOR AND THE KERNEL DATABASE SYSTEM

The Real-Time Monitor (RTM) is responsible for the inter-communications between the kernel database system controller and the object-oriented language interface module. The code for the real-time monitor resides in the Language Interface Controller (Lic) directory. Communications between the real-time monitor and the backend are accomplished via the Test Interface (TI) directory. All procedure calls to TI are characterized by *TI_* at the start of the procedure name, followed by an *R* or an *S* for *receive* or *send* functions, respectively. For example, TI_S$TrafUnit is a process which *sends* message traffic to the backend. Whereas TI_R$TrafUnit is a process which *receives* message traffic from the backend.

The RTM does not develop transactions or queries. It simply receives the text file containing the transactions and pseudocode written by the compiler. It assumes that the transaction and queries are correct as received from the DML compiler. In other words, the RTM submits the translated attribute-based data language transactions or queries to the kernel database system for processing. The RTM determines what order the transactions will be passed to the kernel database. Transactions and queries are executed one at a time

by forwarding them to the backend system, while requesting reports of any errors between each transaction. This process is continued until the last of ABDL transactions or queries are passed to the backend.

The results of the transactions sent to the backend are loaded into a temporary file called, *response_f*. This process is made possible by some new functions which were added into the file *tisubs.c*, located in the CNTRL/TI directory. Once the results are loaded into the file, the RTM loads these result(s) into its next translated attribute-based data language request, and sends the new request to the kernel system. The RTM performs this cycle until it receives the final results for its last translated attribute-based data language request. The code for the newly implemented functions are included in Appendix B.

If the transaction involves a database schema definition, an insertion, a delete request or an update request, then control is returned to the Language Interface Layer (LIL) after the kernel system processes the transaction via RTM. If the transaction involves a retrieval request, the RTM sends the translated attribute-based data language request to the kernel system. Next, the RTM receives the results, loads the results into the temporary file, and calls Kernel Formatting System (KFS) to format/display the last results in object-oriented format. Afterwards, control returns to LIL.

The RTM ensures there are no problems or faults with any communications with the backend. If fault messages exist, the RTM attempts to process them prior to initiating any new transactions. If it is unable to do so, the system will fault and the MDBS program will be terminated. This is an extreme situation, however, and normally the result of some catastrophic backend failure.

The RTM sends the translated attribute-based data language request to the kernel system using *TI_S$TrafUnit()*. It then calls *TI_RTM_chk_res_left()* to ensure all requests have been processed and the results from the kernel system have been received. TI_S$TrafUnit( ) is the function that sends the traffic unit, or *trafunit* to Request Preparation (REQP). As stated, the trafunit consists of two arguments, the dbid and the

trafficunit (see Chapter II of this thesis). Figure 6, depicts the placement of the RTM within the existing system.

In order to implement the real-time monitor into the kernel database system, new functions needed to be created. These new functions are *TI_RTM_chk_res_left()*, *TI_ReqRes_RTMoutput()*, and *TI_print_RTMReqRes()*. As stated the code is provided in Appendix B.

The function *TI_RTM_chk_reqs_left()* communicates with the kernel system and receives the message about the condition of the request. If errors exist, the function *TI_R$ErrorMessage()* is called to get the error message. The function *TI_ErrRes_output()* is called to display the error message. However, if no error occurred, *TI_R$ReqRes()* is called to receive the response from the kernel system. The response buffer is then checked to see if this is the last response. If it is the last response of the translated attribute-based data language request, the results are loaded into the file *response_f* by calling *TI_ReqRes_RTMoutput()* so the RTM can process the next request. Next, it calls KFS to format/display the last results in object-oriented format. Finally, control returns to LIL. In the case of an insert, delete or update request, control is transferred back to LIL without calling the KFS.

The function *TI_ReqRes_RTMoutput()* calls the function *TI_print_RTMReqRes()* to store the response for the translated attribute-based data language request into the temporary file, called r*esponse_f*.

The function *TI_print_RTMReqRes()*, prints the results (i.e. the response) from kernel for the translated attribute-based data language request. The results are printed in the file specified by *o_file*, called *response_f*, and this file is created in the same directory with the RTM to maintain its reachability. The line of the code which opens this file is as follows:

o_file = fopen("/u/mdbs/greg/CNTRL/TI/LangIF/src/Dap/Kc/response_f","w");

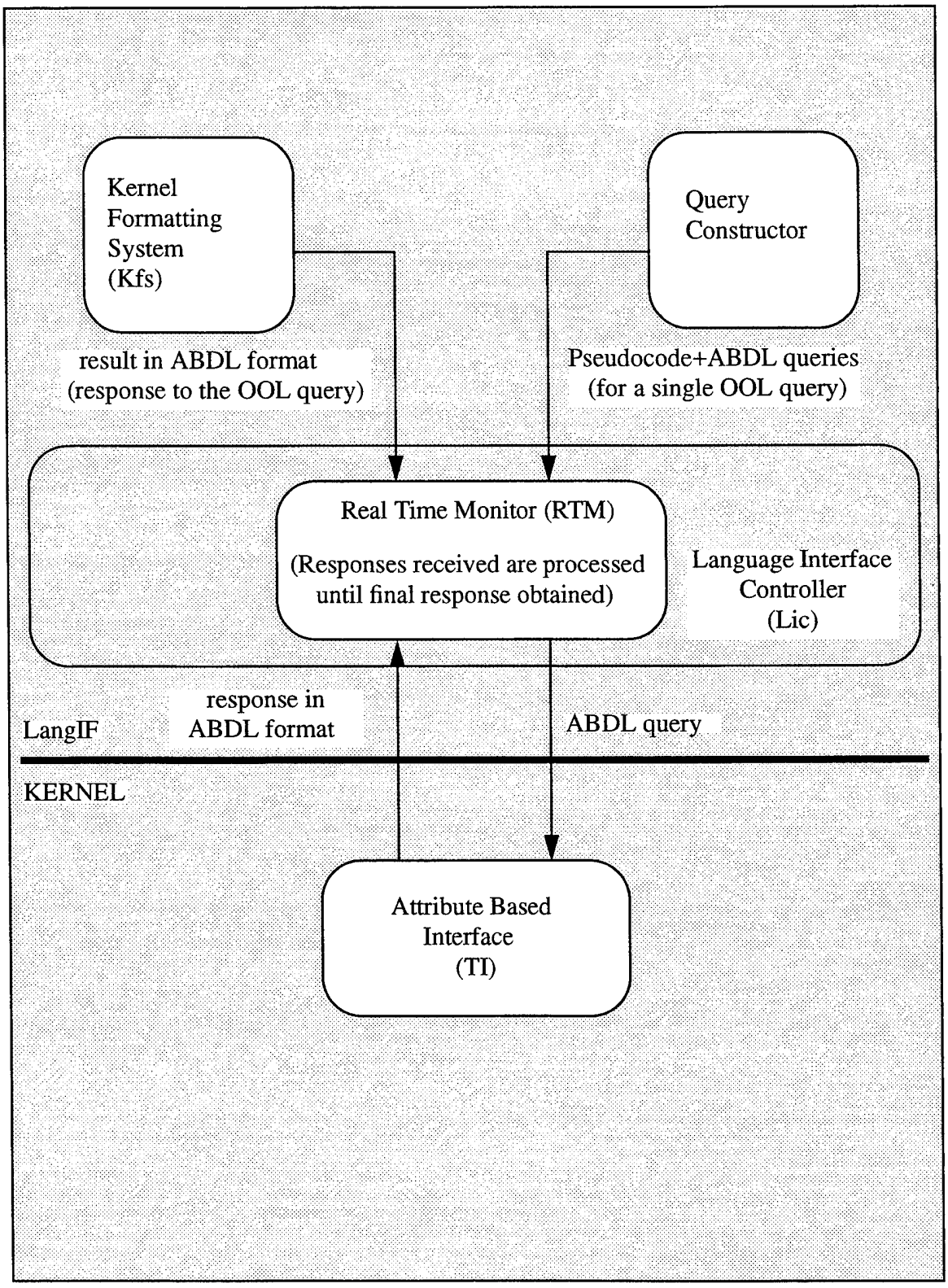


Figure 5. The Real-Time Monitor

V. MODIFICATIONS TO THE KERNEL OPERATIONS FOR THE REAL-TIME MONITOR

A. THE MODIFIED RETRIEVE OPERATION

As stated earlier, the existing Retrieve operation works in two different ways. One method is for the user to include the key attribute for the record in the target list of the retrieve query. The second method is for the user to utilize the *by-clause* in the retrieve query. Please see Chapter 2 for existing implementation of operations.

In the Faculty/Student database (Facstu), the key attribute is *OID*, as defined in the template file. This means that a user must include OID in the target list as a piece of information that must be retrieved. This method of retrieval is not acceptable for the implementation of the object-oriented interface, because the user is not supposed to see the value for OID. This can be avoided if the *by-clause* is used on a value other than the key attribute. However, the use of the by-clause is not needed by our interface in order to conduct queries. Research was devoted to determine if there was a method of getting around this for the Retrieve operation. An in-depth examination of the backend code revealed how the default key value is assigned for the record as well as how the *by-clause* function works.

The RECP process is utilized for the manipulation of the records. This includes selection, retrieval, and value extraction. A review of the code revealed how the Retrieve operation functions and it was in the RECP directory that contained the function for the Retrieve operation.

The file, *retp.c* (retrieve processing) includes several important functions to perform value retrieval. In this file, the first function is *$RETR_PROCESSING()*, which is used for processing a Retrieve request. The important point is, as we mentioned before, Retrieve can include a *by-clause*. Therefore, before processing the Retrieve request, the system should determine the existence of a *by-clause*. To do this, a flag was implemented. This flag is called *by_flag*. Based on the existence of *by-clause* this flag assigned to '1' or

'0' value. Before this point was reached, this flag had to be initialized. Actually, it was not. Because of this, no matter what type of Retrieve (with or without *by-clause*) request is executed, the *BY_HASH_FUNC( )* is called. On the other hand, to make the Retrieve query work without a by-clause, the code had to be modified. It was determined that previous thesis students added a few lines of code into that function, BY_HASH_FUNC(), and maintained an imaginary *by-clause*. An imaginary by-clause because there was a constant attribute assignment for the *by-clause* based on the place in the array which is defined in template file. The part of the code defined in BE/RECP/retby.c is as follows:

```
/*Added 22 October 1993, used for retrieves with no ordering*/
if(!(strcmp(hash_ptr->by_attr,"000")))
```

The above line basically checks to see if the Retrieve request has a *by-clause*. The "000" stands for *by-clause* based on representation in parsed Retrieve request.

```
{
strcpy(hash_ptr->by_attr,rid->RP_ri_tmpl_ptr->rt_entry[1].attr_name);
strcpy(rid->RP_by_hash->by_attr,rid->RP_ri_tmpl_ptr>rt_entry[1].attr_name);
}
```

The **rid->RP_ri_tmpl_ptr->rt_entry[1].attr_name** points to and gets the second field of the array since first field is **rid->RP_ri_tmpl_ptr->rt_entry[0].attr_name** and "**Temp**" for each Template. In other words, what you have in your template file for each template as a second field of the array was the key attribute for your *by-clause*. For instance, for the simple Faculty database we can look at the one of the template files. The part of the code defined in *UserFiles/FACSTU.t* as follows:

Address

TEMP s

OID s

STREET s

CITY s

STATE s

ZIPCODE s

The **rid->RP_ri_tmpl_ptr->rt_entry[1].attr_name** points to and gets the **OID**. If this attribute is not included in the Target List, no attributes may be retrieved without the use of the *by-clause*.

Before processing the Retrieve request, the system must determine whether the Retrieve request contains a *by-clause* or not. As stated before, to make this determination, *by_flag* is defined. While executing script files of the transactions, we discovered that the value of the *by_flag* never changed. This led to the conclusion that there was some mistake in the initialization portion of a *by-clause* Retrieve transaction. A retracing of the code revealed that the related codes were in the *BE/RECP/allsto.c*. file. In the *allsto.c* file, the code for the **Check_for_By**() function is found. A review of the code shows that this function had been implemented incorrectly. A portion of the code as defined in BE/RECP/ allsto.c is as follows:

```
Check_for_By(rid)
{
   int target_ptr, i;
   struct REQtbl_definition  *request;
   struct by_hashing_info  *hash_ptr;
   .
   .
   while(request->req_tbl[target_ptr++][0] != ETList)
```

```
        target_ptr++;

    if (request->req_tbl[target_ptr][0]) {
        .
        .
    } else
        rid->RP_by_hash = NULL;
    .
    .
}
```

The line "**if (request->req_tbl[target_ptr][0])**" is written incorrectly. This line is supposed to check if the Retrieve request has a *by-clause* or not. However, this line only checks whether the Retrieve request has something in that pointer or not. When the code is modified to read "**if (request->req_tbl[target_ptr][0] != '0')**", the fault is solved (recall '0' or '000', indicates the presence of a *by-clause*). Testing of the function after modification, shows that the *by_flag* was assigned the correct value based on the existence of the *by-clause*.

Finally, the correct branch of the 'If statement' was reached. This statement verifies that a by-clause is used. This code, as defined, in *BE/RECP/retp.c* as follows:

```
if (by_flag)
    BY_HASH_FUNC(RP_ri_ptr,result_length,result2);
else
    RB$PUT_SEND(RP_ri_ptr->RB_pointer, result2,result_length);
```

On the other hand, **RB$PUT_SEND( )** did not work properly for Retrieve request. When this function was called, the system crashed and failed to complete the transaction. This function is used by some additional functions in the system. Therefore, this code was not modified in order to maintain the reliability and strength of the existing code.

Additionally, in the *Facstu* database, the type of all the attributes in the template file whether it is integer in reality or not (such as ZIPCODE) are defined as type string, 's', and loaded to the backend with this type definition. This type definition was causing the problems when the request was a Retrieve with *by-clause*.

When **BY_HASH_FUNC( )** is called, depending on the type of the attribute which follows the by-clause, the function **BY_HASH_RECORD( )** is called to hash and store the records. The records are hashed and stored based on the value of a specific variable which is called *bucket_num*. The implementation for the value of *bucket_num* was not sufficient enough to support our object-oriented interface. We were getting negative results for the *bucket_num*, and those values were undefined according to the definition in *COMMON/ commdata.def*. Although the system never reports any error messages, it was not returning any result by executing this type of Retrieve request. Hence, we also modified this part to support our object-oriented interface, this modified function **BY_HASH_RECORD( )** is found in Appendix C.

B. THE MODIFIED DELETE OPERATION

After careful examination of the code, it was determined that no modifications needed to be made to the Delete operation. This operation works correctly and will support the object-oriented interface without any further modifications or adjustments. The real-time monitor will simply pass an attribute-based Delete transaction to the backend in the attribute-based data language. The kernel database will then provide a deletion tag to the specified record or records. This deletion tag will then be used by a garbage collection routine and remove the record at a later time. Therefore we did not need to change the existing Delete operation.

C. THE MODIFIED UPDATE OPERATION

In order to determine what changes needed to be made to the Update operation in the backend, we conducted numerous Update transactions to the *Facstu* database via the attribute-based interface. These tests were run to ensure that the backend could properly

support the object-oriented interface. These transactions included multiple updates of several records, and updates to single records in the database. In addition, we executed queries which included all the relational operators as well as the arithmetic operators.

Upon completion of these tests, several problems were revealed that needed to be dealt with in order for the backend to properly support the object-oriented interface. Two of these problems were within the syntax of the arithmetic operators. The other problem was with the way operands were ordered within the transaction and the results that were given.

The first arithmetic operator that did not work properly was the subtraction operator. When a transaction was conducted with the subtraction operator, the result was incorrect. A sample transaction is as follows:

UPDATE(TEMP=Address)<ZIPCODE=ZIPCODE-10>

The expected result of this transaction was for all of the values of Zipcode to be decremented by 10. The actual result stored back into the records was *ZIPCODE-10*. This indicates that the arithmetic operation was not completed and incorrect data was stored in the record. The system stored the literal value of *ZIPCODE-10* as the new value for the Zipcode attribute-value pair. The same type of problem occurred when using the division operator. The following is a sample transaction using the division operator.

UPDATE(TEMP=Address)<ZIPCODE/2>

Again, the same result occurred with this operator in that the value stored in the record was incorrect. The new value stored within the record was *ZIPCODE/2* instead of the arithmetic operation taking place and giving a new value for the attribute-value pair.

After reviewing the code for the backend operations, it was found that the parser for attribute-based side does not recognize the operands or the subtraction or division operators if there is no space between the operands and the operators. Since, the REQP process

parses the user's request and checks for proper format and syntax before forwarding the request, an investigation of the files in the REQP directory was conducted.

In the REQP directory, there is a file called *lsrc* which includes all the TOKEN definitions for the attribute-based database. The investigation revealed that the subtraction (-) and the division (/) operators had been included in the string TOKEN definition. This string TOKEN definition is defined as *ALPHANUMFIRST*. The name *ALPHANUMFIRST* was delineated by previous thesis students, and is used only in the *lsrc* file.

Because the operators are defined in this fashion, the parser recognizes the operators as part of the string for the attribute being modified. Hence, the operator is viewed as part of the string and not the arithmetic operator. Due to this fact, a space is required between the operands and the operator itself. This case is only true for the subtraction and division operators.

Based on previous thesis work, the (-) and (/) characters were required to properly define attribute-values in other databases. For example, in the *MAINT.r* database, the users needed these characters to describe certain attributes like *PartNumber=prtno-1*. In this case the (-) was a separator between the descriptor *prtno* and the 1. This is the reason that the (-) was included in the string TOKEN definition. The same case applies to the (/) character.

The method of entering a transaction where the subtraction and division operators are used, requires the user to include a space before and after the operator. Once this is accomplished, the transaction works properly. For example, a correct format for the above transaction is as follows:

UPDATE(TEMP=Address)<ZIPCODE=ZIPCODE - 10>

This transaction results in the value for Zipcode to be decremented by 10, and the correct data stored in the record. Please note the spaces before and after the (-) operator.

For this thesis research, we found that it was not necessary to change this string TOKEN definition to implement the object-oriented interface. Hence, the code was not

```
/* lsrc */
.
.
.
%{
    char *mem_ptr;
    .
    .
    .
    #ifdef  ParPrntFlag
    "+"|"-"|"*"|"/" {
                strcpy(yylval.str,yytext);
                return(OP);
               }
    .
    .
    .
    [-_"/"A-Za-z0-9]+ {
                  strcpy(yylval.str,yytext);
    #ifdef  ParPrntFlag
                 printf("%s in lsrc \n",yytext);
    #endif
                  return (ALPHANUMFIRST);
                 }
  }
```

Figure 6: lsrc Code

modified and the correct spaces were inserted by the compiler for transactions that required the subtraction and division operations.

The existing code for the lsrc file that defines the subtraction (-) and division (/) operators is depicted in Figure 6 (the entire *lsrc* file is included in Appendix C):

The problem encountered with the ordering of the operands while using the subtraction and division operators was due to the actual code written in the backend. During the testing of the subtraction and division operators, we found that depending on the order of the operands, the results could be incorrect. For example, the following Update transaction results in incorrect data being stored back into the record.

UPDATE((TEMP=Address)and(ZIPCODE=93940))<ZIPCODE - 5>

This transaction should have decremented the value of Zipcode by 5 and then store the new value in the record. However, the actual value stored in the record for Zipcode was *-93935*. This indicates that the system had taken the value *5* as the first argument in the subtraction operation and the value *93940* as the second argument. This of course yields a negative result, which is incorrect.

The code that was causing these incorrect results was corrected by interchanging the order of the operands. This code was found in the file updp.c. This corrected file is included in Appendix C.

D. THE MODIFIED RETRIEVE-COMMON OPERATION

As stated earlier, the retrieve-common operation works correctly. However, this operation is not required by the object-oriented interface. A similar retrieve-common operation is accomplished through the use of the real-time monitor. To retrieve records that have a common attribute-value pair, then the user retrieves two different sets of records from the system and then completes a cross multiplication of these records to find the one with the common attribute.

It is not required in order for the kernel database system to support the object-oriented interface. Therefore, this operation was not modified either. This operation was left intact and not changed.

VI. CONCLUSIONS

A. ACCOMPLISHMENTS

The primary goal of this thesis research was to ensure that the kernel database system could properly support the newly implemented object-oriented interface. In order to accomplish this task, several steps needed to be completed. The first, being a thorough investigation of the underlying, backend code. The second step, was to determine how each of the primary operations worked. Thirdly, the intercommunications and relationship between the real-time monitor and the backend needed to be implemented. Lastly, we needed to determine what modifications, if any, needed to be made to this backend code in order to support the object-oriented interface.

The review of the backend code required the code for all the operations be tracked down and broken down into their atomic operations. This included documenting of each and every function within these files and determining which functions called other functions. This was vital in order to sketch out the intercommunications and links between the functions. Once the intercommunications were established, we can determine the flow of control throughout the program execution.

Determining how each of the primary operations worked and communicated with each other was another large task. This was accomplished by executing numerous transactions utilizing the attribute-based interface. These queries include Retrievals, Updates, and Delete transactions. In the process of running these transactions we were able to determine what the system peculiarities were for each type of transaction. These peculiarities were noted and described to the compiler team so they could properly generate the queries using the object-oriented interface. As stated in the previous chapters of this thesis, several items were found that required correction in order for the interface to work properly.

Once the problems were corrected in the backend, our focus shifted to the implementation of the real-time monitor with the backend. This involved extensive

research to determine where the actual code for the real-time monitor would be inserted into the kernel code. It was also essential that the real-time monitor was passed the correct parameters and the correct parameters were passed to the backend for execution. This was important, because the kernel system can only process transactions in the attribute-based format. Due to the primitive data language which resides in the kernel, the real-time monitor is required to pass the transactions in the proper order. If this is not accomplished, the system will crash and results will be incorrect.

Lastly, documentation is provided concerning all the modifications made to the kernel primary operations in order to support the object-oriented interface. During this research, we were able to resolve several problems encountered with the implementation of the object-oriented interface. These problems are described in detail in Chapter V.

B. RECOMMENDATIONS FOR FUTURE RESEARCH

This thesis research revealed several areas that would enhance the kernel system to further improve it's capabilities. One such area is the ability for the stem to accept and recognize floating point numbers. Another topic is to research the ability for the system to recognize different key attributes and assign a different key attributes on the fly. The third area is for the implementation of the "OR" and "AND" operators.

Currently, the kernel database system does not recognize floating point numbers. In the *lsrc* file we were able to get the parser for the interface to accept and recognize these floating point numbers, however, the kernel system did not accept them as valid input. A change was made to the *lsrc* file which would allow the user to input floating point numbers but when the transaction was finally sent to the backend it crashed. It was found that the modifications to the backend would be too complicated and were not within the scope of this thesis. However, this problem could be resolved if it were taken into account in the beginnings phases of design and implementation.

The second area of future research would be to implement the ability for the system to retrieve on any attribute passed to the kernel database system. Hence the kernel should

have the ability to Retrieve any record based on it containing the attribute specified by the user, not just the key attribute as defined by the template file. It is true that the use of the by-clause allows the user to retrieve a record on a particular attribute, however, the default key attribute must be in the target list in order for the transaction to work correctly. For our particular application, this is unsatisfactory because the user is not supposed to see this key attribute, OID. We were able to overcome this constraint through the use of the real-time monitor which passed each transaction to the kernel one at a time.

Lastly, the "OR" and "AND" operators could be implemented. This would provide the user a better means for focusing on certain records in the database without having to execute several queries then cross-multiplying the results of those transactions. Instead, the user may utilize the "OR" and "AND" operators in the following fashion.

RETRIEVE((TEMP=Name)(FNAME=Robert) OR
(FNAME=David))(LNAME)

The results of this query would be to display the last name of all the records in database that have a first name of *Robert* or *David.* This saves the user from having to conduct two retrievals of the records in name and then displaying the last names separately. It can be accomplished in one transaction. This same type of transaction can be used but with the "AND"operator. For example, if the user is looking for the records that have the a common attribute, then the following transaction could be used.

RETRIEVE((TEMP=Address)(Number=1564) AND (Zipcode=93940))
(City, Street)

This query would display all the records in the database that have the common attribute where *Number=1564* and the value of *Zipcode=93940* and then display the attribute value for the City and the Street, which are contained in the target list. These

transactions seem simple enough, however, they were not within the scope of this thesis and were left for future research.

C. SUMMARY

In summary, this thesis research accomplished the goals that were set out. The kernel database system is more than capable of supporting the object-oriented interface. As described, only a few modifications were required to be made to the kernel operation modules in order for the kernel database system to support the object-oriented interface. Once the changes were made, the new interface worked properly and as expected.

The use of the DML and DDL compilers along with the real-time monitor were necessary in order for the system to support the rich object-oriented data language/model. However, by utilizing these components, the kernel database system executes transactions and queries most effectively and accurately. This thesis research has proven that the kernel database system can be implemented to support several different data models/languages. This support allows a user to remain within their own familiar data model/language while accessing data from other databases. In addition, there is no requirement to duplicate an entire database in order to support another database language.

APPENDIX A- PROCESSES

A fairly detailed description of the multibackend database supercomputer (MDBS) can be found in (Hsiao Dec91). At the time of this writing, the MDBS is being ported to a new controller and backend. The new controller is a Sun model 4/110 workstation called db11 which has the Sparc 4 RISC architecture with 8 Megabytes of random access memory (RAM) and one 373 Megabyte hard drive. The two backends, db12 and db13, are Sun model 4/280 workstations which each have Sparc 4 RISC architecture with 16 Megabytes of RAM, one 96 Megabyte disk for paging, one 96 MB disk for meta-data, and one 1000 MB hard disk for data (Watkins Sep93).

MDBS is designed so that all of the source code for both the controller and backends are stored on the controller and the mdbs/VERSION/CNTRL and mdbs/VERSION/BE directories.

A brief review of the functions performed by the twelve MDBS processes is included here as an aid in understanding the communication channel design and capturing the meaning of the codes are implemented for these processes (Watkins Sep93). Knowledge of the code will help the programmer in making modifications/updates if needed.

All of the communication channels shown in next page are established during system generation (start-up). Information is shared through message passing between processes.

Network

The controller is connected to the backends via one backplane and to the local computer science department network via a second backplane. When the database system is executing, the controller communicates with the backends via sockets. The magnetic tape drive on the controller can be used for mass loading of a database, back-ups of a database, and normal back-ups of all existing files.

Communications with the controller are handled by the backplane which communicates with the controller and other backends via the Ethernet. The local area network (LAN) consists of the Ethernet cable, the backplanes and their transceivers. There is one

LAN for the MDBS.Each backend maintains the meta-data table for the entire database.

Controller

The six controller processes are controller get (CGET), controller put (CPUT), test interface (TI), request processing (REQP), insert-information generation (IIG), and post processing (PP), all under CNTRL directory. A simplified diagram of how the six processes interrelate is in the figure on page 46.

The CPUT process is responsible for sending DGRAM messages across the ethernet to other MDBS workstations. The CGET process is responsible for receiving DGRAM messages from other workstations. The TI process is the user interface. It contains the routines for activating the selected language interface and capturing the user's instructions from the terminal. High level requests are translated into the native language of the system, which can also be used directly. This process maps each data model and language to the kernel data language (KDL) and kernel data model (KDM). On MDBS, kernel data is stored using the attribute based data model (ABDM) and attribute data language (ABDL). The requests are forwarded to Request Preparation. The REQP process parses the user's request and checks for proper format and syntax before forwarding the request. Formatted requests are broadcast to the Directory Management which maintains the meta-data of the system. For record inserts two way communication between Directory Management and Insert Information Generation determines the location for new records and allows that information to be shared with all backends so that meta-data can be kept consistent.

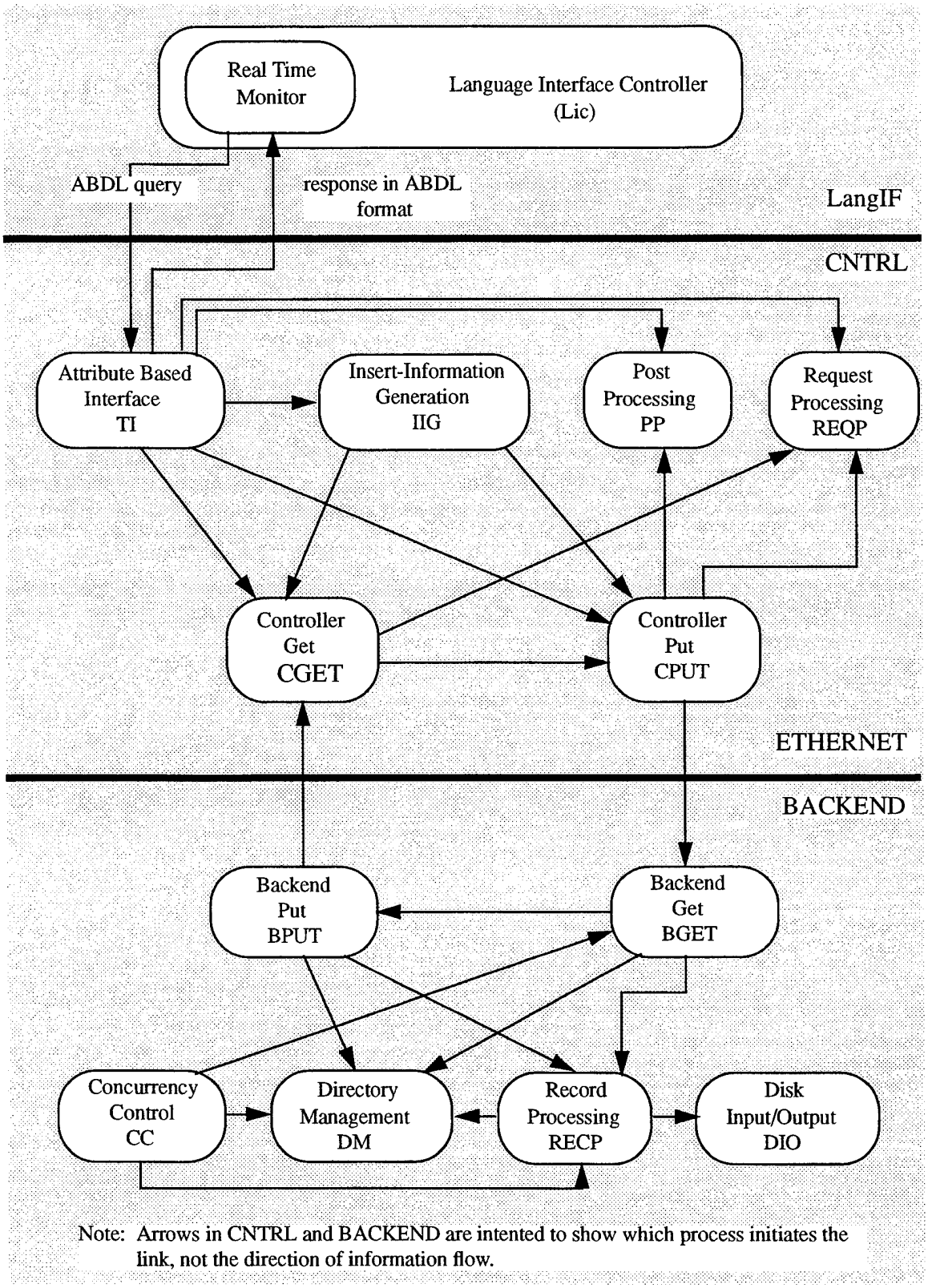
The IIG process is used to determine which backend will insert a particular record. During an insert operation, the IIG process tells a specific backend--via the CPUT process--whether to start a new track or put the record being inserted into an existing track. The IIG process uses a space utilization table from which it can determine for each cluster which backend contains the last full track of the cluster, and can determine the first available track for inserting a new track of clustered records. The PP process properly formats the results received from backend machines so that they can be displayed to the user. It combines records returned from the backends and performs any aggregate operations

required before returning the results to the user.

Backend

The six backend processes are backend get (BGET), backend put (BPUT), record processing (RECP), concurrency control (CC), directory management (DM), and disk input/output (DIO). All six of these processes run on each backend machine participating in MDBS.

The BPUT process is responsible for sending DGRAM messages to other MDBS workstations over the ethernet cable. The BGET process receives these same inter-machine messages for the backend machine on which it is running. Each backend communicates with the front end and the other backends via BGET and BPUT. The RECP process is responsible for the manipulation of records. This includes selection, retrieval, and value extraction. The concurrency Control ensures that records can only be accessed by one user at a time. It is charged with maintaining meta-data and base-data (record) integrity during the processing of transactions. The DM process is responsible for all access to the meta-data disk. It coordinates with the record processing process in gathering information about how the base-data (records) are stored. The DIO process handles all reading from and writing to the base-data (record) disk.



Note: Arrows in CNTRL and BACKEND are intented to show which process initiates the link, not the direction of information flow.

SUMMARY

The job of establishing the inter-machine communication channels start in the controller. During the initialization, the TI process tells the CGET process to set backend numbers with a SetNoBE (message code 923) message. The CGET process then receives an initial identification message (BeWho, message code 925) from a backend over the ethernet. CGET forwards this message to the CPUT process, which transmits a message (setNoBE, message code 925) back to the identified backend over the ethernet. Initialization is complete when that backend's BGET process sends an inter-process message with the backend number (SetBeNo, message code 924) to the CC, DM, RECP, and BPUT processes. These and other process and message related codes are contained in msg.def header file. The shutdown is accomplished by the finish send-receive (finishsr()) function and the close socket (closesock()) function (Watkins Sep93).

APPENDIX B -ADDED PROGRAM CODE

```
/*
*
* File Name   : tisubs.c
* Source      : /u/mdbs/greg/CNTRL/TI/tisubs.c
* Info        : We added three new functions into this file to support RTMs needs. And,
                only these three functions and one other are included here.
*
*/


#include <stdio.h>
#include "flags.def"
#include "dblocal.def"
#include "commdata.def"
#include "tstint.def"
#include "tstint.ext"
#include "msg.def"
#include "beno.def"

TI_ReqRes_RTMoutput(rid, response, reslen)
  /* Output the response for a request to the file   */
  /* For Object_Oriented Interface RTM purposes only */
  /* Added by Necmi YILDIRIM, on 03 MAY 1995        */
  struct ReqId  *rid;
  char   response[];  /* response from mdbs   */
  int    reslen;
{
#ifdef EnExFlag
  printf("Enter TI_ReqRes_RTMoutput \n");
  fflush(stdout);
#endif

  TI_print_RTMReqRes(r_file_ptr, rid, response, reslen);

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

TI_print_RTMReqRes(o_file, rid, res, reslen)
```

```
  /* Print the results (response) from MDBS for a request   */
  /* The results are printed in the file specified by o_file */
  /* For Object_Oriented Interface RTM purposes only        */
  /* Added by Necmi YILDIRIM, on 03 MAY 1995                */

  FILE  *o_file;
  struct ReqId  *rid;
  char  res[];
  int   reslen;
{
  int   k = 1,j = 0;
  char  first_attr[ANLength + 1], attr[ANLength + 1], val[AVLength + 1];
  int first, first_rec, req_done;

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

  /* Following line opens a file called "response_f" to fill the output      */
  /* of the query and used by the Real_Time_Monitor (RTM) for its next query */
  /* Added by Necmi YILDIRIM on 04/11/95                                     */

  o_file = fopen("/u/mdbs/greg/CNTRL/TI/LangIF/src/Dap/Kc/response_f","w");

  if (reslen >= 2) {
    if (res[reslen-3] == CSignal) {
      /* indicate that a request has finished execution */
    req_done = TRUE;
    reslen -= 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 < reslen - 1) {
    /* copy the attribute name */
    j = 0;
    while (attr[j++] = res[k++])
      ;
#ifdef pr_flag
    printf("attr = >%s<\n", attr);
#endif

    /* copy the attribute value */
    j = 0;
    while (val[j++] = res[k++])
      ;
#ifdef pr_flag
    printf("val = >%s<\n", val);
#endif

    first = strcmp(first_attr,attr);

    if (!first_rec && !first){
      fprintf(o_file,")");
     }
    else if (first && attr[0] != '#'){
      fprintf(o_file,", ");
     }
    else
      first_rec = FALSE;

    fflush(stdout);
    if (attr[0] == '#')
      PR_AGOP(attr,val,o_file);
    else {
      if (!first){
        fprintf(o_file,"\n\t(");
       }

       fprintf(o_file, "<%s, %s>", attr, val);

       if (k >= reslen - 1 || res[k] == '#'){
         fprintf(o_file, ")");
        }
     }
```

```
    fflush(stdout);
    if (o_file)
      fflush(o_file);

  } /* end while */

#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 TI_print_RTMReqRes\n");
  fflush(stdout);
#endif
  } else {
    fprintf(o_file,"\n\n\tRequest completed\n");
    }

  fclose(o_file);

} /* end TI_print_RTMReqRes */
```

PR_AGOP(attr,val,ofile)

```
  char attr[], val[];
  FILE *ofile;
{
  char *ptr;
  int  op;

#ifdef EnExFlag
 printf("Enter PR_AGOP\n");
 fflush(stdout);
#endif
#ifdef pr_flag
  printf("attr = >%s<, val = >%s<, ofile = >%d<\n", attr, val, ofile);
#endif
```

```
  ptr = &attr[2];
  op = attr[1] - '0';

  switch (op) {
    case AOMAX:
      fprintf(ofile,"\n\t(<MAX(%s),%s>) ",ptr,val);
      break;

    case AOMIN:
      fprintf(ofile,"\n\t(<MIN(%s),%s>) ",ptr,val);
      break;

    case AOAVG:
      fprintf(ofile,"\n\t(<AVG(%s),%s>) ",ptr,val);
      break;

    case AOSUM:
      fprintf(ofile,"\n\t(<SUM(%s),%s>) ",ptr,val);
      break;

    case AOCOUNT:
      fprintf(ofile,"\n\t(<CNT(%s),%s>) ",ptr,val);
      break;

    default:
      printf("\nERROR: Invalid aggregate operator in PR_AGOP.\n");
      fflush(stdout);

  } /* end switch */

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


TI_RTM_chk_reqs_left()
  /* For Object_Oriented Interface RTM Purposes only  */
  /* Added by Necmi YILDIRIM on 03 MAY 1995         */
{
  char   answer, c, err_msg[RESLength], response[RESLength],
```

```
        str_count[15], tid[TILength + 1], trafunit[TULength];
  int   i, num_count, res_len;
  struct ReqId  rid;
  int j,k;

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

    /* receive the next message */
    receive(msg_q, &header, TRUE);

    switch (header.type) {

      case CH_ReqRes:  /* the results (partial results) for a request */
        /* receive the results */
        TI_R$ReqRes(&rid, response);

        if (Timer_on)
          Num_buffers++;

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

        if (response[res_len - 3] == CSignal)  /* The request is done */
          --reqs_left_count;

        /* output the response */
        TI_ReqRes_RTMoutput(&rid, response, res_len);
        break;

      case ReqsWithErr:  /* a traffic unit that has errors is returned */
        /* receive the traffic unit */
        TI_R$ErrorMessage(trafunit, err_msg);
        /* output the error message */
        TI_ErrRes_output(trafunit, err_msg);
            /* TI_ErrRes_output will update reqs_left_count */
        break;

      case CH_TransDone:  /* a transaction is done */
```

```
        j = k = 0;
        while (tid[j++] = msg_q[k++])
          ;
        printf("The transaction with id (%s) is done \n", tid);
        break;

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

      } /* end switch */

#ifdef EnExFlag
  printf("Exit TI_RTM_chk_reqs_left\n");
  fflush(stdout);
#endif
} /* end TI_RTM_chk_reqs_left */
```

APPENDIX C- MODIFIED PROGRAM CODE

```
/*
*
* File Name   : allsto.c
* Source      : /u/mdbs/greg/BE/RECP/allsto.c
* Info        : This file contains structures to allocate storages for the request.
*
*/



#include <stdio.h>
#include "flags.def"
#include "beno.def"
#include "commdata.def"
#include "recproc.def"
#include "recproc.ext"

struct RP_rid_info *ALL_STO_RP_ri(rid, dbid, req_ptr, addrs,
                                  tmpl_ptr, RP_rb, NewTrack )
  /* Add an entry to RP_rid_info queue.  If the request is */
  /* INSERT caused by UPDATE, this routine increments the number of     */
  /* INSERTs (caused by the UPDATE) that are to be inserted at this     */
  /* backend. */

  struct ReqId *rid;
  char   dbid[];
  struct REQtbl_definition *req_ptr;
  struct addr_set *addrs;
  struct rtemp_definition *tmpl_ptr;
  struct ResultBuffer *RP_rb;
  int    NewTrack;
{
  struct RP_rid_info *new_ptr,
                     *upd_RP_ri_ptr,
                     *FIND_RP_ri();
  int    i, k;
  int    num_req_no,
         upd_num_req_no;
  struct ReqId upd_rid;
  struct aggregate_info *agg_ptr;
  int    flag_agg_op;
```

```
#ifdef pr_flag
  int x;
#endif

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

  /*if this is a retrieve common, RP_rid_info was already allocated */
  if(((req_ptr->req_tbl[4][0] - '0') == RET_COM_S) ||
                    ((req_ptr->req_tbl[4][0] - '0') == RET_COM_T)){
    new_ptr = FIND_RP_ri(rid);
  } else    /*this is not a retrieve common request*/
    {
    /* allocate space for the new RP_rid_info */
    if ( (new_ptr = (struct  RP_rid_info  *)
            malloc( sizeof(struct  RP_rid_info) ) ) == NULL )
       {
       printf("*** ALL_STO_RP_ri 10   Problem with malloc \n");
       fflush(stdout);
       }
#ifdef  m_pr_flag
  printf("ALL_STO_RP_ri   --   new_ptr = %o \n", new_ptr);
  fflush(stdout);
#endif

       /* add the entry to the queue */
       if (!rear_RP_rid_info) { /* there is no node */
          front_RP_rid_info = new_ptr;
          rear_RP_rid_info = new_ptr;
       } else { /* there are some nodes */
          rear_RP_rid_info->next_RP_rid_info = new_ptr;
          rear_RP_rid_info = new_ptr;
       }

       rear_RP_rid_info->next_RP_rid_info = NULL;

       /* store the request id */
       strcpy(new_ptr->RP_ri_rid.traf_id, rid->traf_id);
       strcpy(new_ptr->RP_ri_rid.req_no,  rid->req_no );
```

```
        new_ptr -> RP_ri_hash = NULL;
        new_ptr -> RP_by_hash = NULL;
        new_ptr -> RP_agg_ptr = NULL;
        new_ptr->SrceDone = FALSE;

    } /*end if not Ret Common*/

  /* store the information in the new entry */
  /* store the database id */
  strcpy(new_ptr->RP_ri_dbid, dbid);

  /* store the request */
  k = -1;
  do {
    ++k;
    strcpy(new_ptr->RP_ri_req.req_tbl[k], req_ptr->req_tbl[k]);
  } while (req_ptr->req_tbl[k][0] != EORequest);

#ifdef pr_flag
  for (x = 0; new_ptr->RP_ri_req.req_tbl[x][0] =! EORequest; x++)
    printf("%s\n",new_ptr->RP_ri_req.req_tbl[x]);
  printf("%s\n",new_ptr->RP_ri_req.req_tbl[x+1]);
  fflush(stdout);
#endif

  /* store addresses */
  new_ptr->RP_ri_addrs.as_no_addrs = addrs->as_no_addrs;

#ifdef pr_flag
  printf("addrs->as_no_addrs = %d\n",addrs->as_no_addrs);
#endif

  for (k = 0; k < addrs->as_no_addrs; ++k) {
    new_ptr->RP_ri_addrs.as_addrs[k].cylinder_no =
                              addrs->as_addrs[k].cylinder_no;
    new_ptr->RP_ri_addrs.as_addrs[k].track_no = addrs->as_addrs[k].track_no;
  }

  if (new_ptr->RP_ri_req.req_tbl[4][0] - '0' == RETRIEVE ){
    /* req_tbl searched for agg_op. If present, allocate and initialize */
    /* the structure in the new_ptr  DSM */

    Check_for_By(new_ptr);
```

```
    flag_agg_op = aggr_op(&(new_ptr->RP_ri_req));

    if (flag_agg_op) {
      /* allocate space for the new RP_agg_ptr */
      if ((agg_ptr = (struct aggregate_info *)
              malloc(sizeof(struct aggregate_info))) == NULL) {
         printf("*** ALL_STO_RP_ri 10   Problem with malloc \n");
         fflush(stdout);
      } else {
        for (i = 0; i < MAX_AG_OPS; i++) {
          agg_ptr->nmax[i] = NMAX;
          agg_ptr->nmin[i] = NMIN;
          agg_ptr->ncount[i] = 0;
          agg_ptr->nsum[i] = 0;
          agg_ptr->avg_sum[i] = 0;
          agg_ptr->avg_count[i] = 0;
        }
      new_ptr->RP_agg_ptr = agg_ptr;
      }
    }
 } /* end if RETRIEVE */
else
  new_ptr->RP_agg_ptr = NULL;
 new_ptr->addr_ind = 0;
 new_ptr->RP_ri_NewTrack = NewTrack;
 new_ptr->RP_ri_tmpl_ptr = tmpl_ptr;
 new_ptr->RB_pointer = RP_rb;

 for (k = 0; k < MAX_ADDRS_UPD; ++k)
   new_ptr->RP_ri_urcpt[k] = 0;

 num_req_no = str_to_num(rid->req_no);
 if (num_req_no <= UpdCoef)
   /* the request is not insert caused by update */
   new_ptr->RP_ri_status = NOT_WAITING;
 else {
   /* the request is insert caused by update */
    new_ptr->RP_ri_status = UpdFirstPhaseWaiting;
   /* increment number of inserts (caused by the update) that are to */
   /* be inserted at this backend */
   /* construct the request id of the update request */
   strcpy(upd_rid.traf_id, rid->traf_id);
   upd_num_req_no = (num_req_no / UpdCoef) - 1;
```

```
    num_to_str(upd_num_req_no, upd_rid.req_no);
    /* get a pointer to RP_rid_info entry for the update request */
    upd_RP_ri_ptr = FIND_RP_ri(&upd_rid);
    /* increment count */
    upd_RP_ri_ptr->this_BE_to_ins_count++;
  }

  new_ptr->RP_ri_no_completed_writes = 0;
  new_ptr->this_BE_to_ins_count = 0;
  new_ptr->no_more_gen_ins_msg_rcv = FALSE;

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

  return(new_ptr);

} /* end ALL_STO_RP_ri */

Check_for_By(rid)
  struct  RP_rid_info  *rid;
{
  int target_ptr, i;
  struct REQtbl_definition  *request;
  struct by_hashing_info  *hash_ptr;

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

  request = &(rid->RP_ri_req);

  for (target_ptr = 6;
        request->req_tbl[target_ptr][0] != EOQuery; target_ptr++ );
  target_ptr++;

  while(request->req_tbl[target_ptr++][0] != ETList)
    target_ptr++;
```

Following part of the code is never placed into the running system code. The code which is used is still the old one

```
  if (request->req_tbl[target_ptr][0] != '0') {  /* modified by N.YILDIRIM on 04/21/95   */
                                                 /* to get the correct result for by_flag */

    /* allocate space for the new RP_by_hash */
    if ( (hash_ptr = (struct by_hashing_info * )
            malloc( sizeof(struct by_hashing_info) ) ) == NULL ) {
      printf("*** Check_for_By   Problem with malloc \n");
      fflush(stdout);
    } else {
      rid->RP_by_hash = hash_ptr;
      hash_ptr->new_flag = TRUE;
      strcpy(rid->RP_by_hash->by_attr, request->req_tbl[target_ptr]);

      for (i = 0; i < NUMBER_OF_BUCKETS; i++)
          hash_ptr->hash_table[i] = NULL;
    }
  } else
    rid->RP_by_hash = NULL;

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

struct RP_rid_info *All_Sto_RP_ri_RetCom(rid)
  struct ReqId *rid;
{
  struct  RP_rid_info  *new_ptr;
  struct  hashing_info *hi_ptr;
  int i;

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

  /* allocate space for the new RP_rid_info */
  if ((new_ptr = (struct  RP_rid_info  *)
            malloc( sizeof(struct  RP_rid_info) ) ) == NULL ) {
    printf("*** ALL_STO_RP_ri_RetCom 10   Problem with malloc \n");
    fflush(stdout);
  }
```

```
#ifdef m_pr_flag
  printf("ALL_STO_RP_ri_RetCom  --  new_ptr = %o \n", new_ptr);
  fflush(stdout);
#endif

  /* add the entry to the queue */
  if (!rear_RP_rid_info) {
    /* there is no node */
    front_RP_rid_info = new_ptr;
    rear_RP_rid_info = new_ptr;
  } else {
    /* there are some nodes */
    rear_RP_rid_info->next_RP_rid_info = new_ptr;
    rear_RP_rid_info = new_ptr;
  }

  rear_RP_rid_info->next_RP_rid_info = NULL;

  /* store the request id of the source */
  strcpy(new_ptr->RP_ri_rid.traf_id, rid->traf_id);
  strcpy(new_ptr->RP_ri_rid.req_no,  rid->req_no );

  /*allocate the hashing info structure*/
  if ( (hi_ptr = (struct  hashing_info  *)
            malloc( sizeof(struct  hashing_info) ) ) == NULL ) {
    printf("*** ALL_STO_RP_ri_RetCom 10   Problem with malloc \n");
    fflush(stdout);
  }

  /*attach this hashing info structure to RP_ri_ptr*/
  new_ptr->RP_ri_hash = hi_ptr;

  /*turn on the new request flag so further initializations can take place*/
  /*in the HASH_FUNC routine  */
  hi_ptr->new_request_flag = TRUE;

  /*initailize number of target BEs which have responded*/
  new_ptr->RP_ri_hash->target_cnt = 0;

  /*next is to initialize the hash tables*/
  for( i=0; i<NUMBER_OF_BUCKETS; )
    new_ptr->RP_ri_hash->hash_table[i++] = NULL;
```

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

    return(new_ptr);

} /*end ALL_STO_RP_ri_RetCom*/

/* Determine if an aggregate */
/* operator is present in the req_tbl      */
aggr_op(request)
   struct  REQtbl_definition  *request;
{
  int test, target_ptr;

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

  for (target_ptr = 6;
         request->req_tbl[target_ptr][0] != EOQuery; target_ptr++ );
  target_ptr++;

 while(request->req_tbl[target_ptr++][0] != ETList) {
    test = atoi(request->req_tbl[target_ptr++]);
    if (test > 0)
      return(TRUE);
  }

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

  return(FALSE);

} /* end aggr_op */
```

```
/*
*
* File Name   : lsrc
* Source      : /u/mdbs/greg/CNTRL/REQP/lsrc
* Info        : This file contains TOKEN definitions for the ABDL Parser.
*
*/

%e 220
%p 725
%n 92
%k 29
%a 5527
%o 3500
%{
  char *mem_ptr;
  char memget();  /* after doing 'lex lsrc', we have to go to  */
                  /* file lex.yy.c and change the statement */
                  /* #define  input() ...getc(yyin)... */
                  /* to */
                  /* #define  input() ...memget()... */
%}

%%
"{"         {
            return (BOT);
            }
"}"         {
            return(0);
            }

"["         {
#ifdef  ParPrntFlag
            printf("BOR [ in lsrc\n");
#endif
            strcpy(yylval.str,yytext);
            return (BOR);
            }
"]"         {
#ifdef  ParPrntFlag
            printf("EOR ] in lsrc\n");
#endif
            strcpy(yylval.str,yytext);
```

```
            return(EOR);
            }

DELETE      {
#ifdef ParPrntFlag
            printf("DELETE in lsrc\n");
#endif
            return (TOKDELETE);
            }
RETRIEVE    {
#ifdef ParPrntFlag
            printf("RETRIEVE in lsrc\n");
#endif
            return(TOKRETRIEVE);
            }
COMMON      {
#ifdef ParPrntFlag
            printf("COMMON in lsrc\n");
#endif
            return(TOKCOM);
            }
INSERT      {
#ifdef ParPrntFlag
            printf("INSERT in lsrc\n");
#endif
            return(TOKINSERT);
            }
UPDATE      {
#ifdef ParPrntFlag
            printf("UPDATE in lsrc\n");
#endif
            return(TOKUPDATE);
            }

POINTER     {
#ifdef ParPrntFlag
            printf("POINTER in lsrc\n");
#endif
            return(POINTER);
            }
AVG         {
            return(AVG);
            }
```

```
SUM         {
            return(SUM);
            }
COUNT       {
            return(COUNT);
            }
MAX         {
            return(MAX);
            }
MIN         {
            return(MIN);
            }
and         {
            strcpy(yylval.str,yytext);
            return (TOKAND);
            }
or          {
            strcpy(yylval.str,yytext);
            return (TOKOR);
            }
of          {
            return(TOKOF);
            }
WITH        {
            return(TOKWITH);
            }
BY          {
            return(TOKBY);
            }

"<="        {
            return (LE);
            }
">="        {
            return (GE);
            }
"/="        {
            return (NE);
            }
"<"         {
#ifdef  ParPrntFlag
            printf("LWEDGE in lsrc\n");
#endif
```

```
            return (LWEDGE);
            }
">"          {
#ifdef  ParPrntFlag
            printf("RWEDGE in lsrc\n");
#endif
            return (RWEDGE);
            }
"="          {
            return (E);
            }

"+"|"-"|"*"|"/" {
             strcpy(yylval.str,yytext);
             return(OP);
            }
"("          {
#ifdef  ParPrntFlag
            printf("LPAR in lsrc\n");
#endif
            return (LPAR);
            }
")"          {
            return (RPAR);
            }
","          {
            return(COMMA);
            }
[\t\n]       {
#ifdef  ParPrntFlag
             printf("[\\t\\n] in lsrc \n");
#endif
            ;
            }
[A-Z][A-Z][-_A-Z0-9]* {
                strcpy(yylval.str,yytext);
#ifdef  ParPrntFlag
             printf("%s in lsrc \n",yytext);
#endif
                return (LETTERFIRST);
               }

[+-]?[0-9]+"."[0-9]+ {
```

Modofied to allow floating point numbers in the ABDL query

```
            strcpy(yylval.str,yytext);
            return(NUMS);
                }

[0-9]+        {
            strcpy(yylval.str,yytext);
            return(NUMS);
            }

[-_"/"A-Za-z0-9]+ {
                strcpy(yylval.str,yytext);
#ifdef  ParPrntFlag
            printf("%s in lsrc \n",yytext);
#endif
                return (ALPHANUMFIRST);
                }
%%

yywrap()
{
      /* Tell lex that you want the normal end of file     */
      /* processing.  Lex thinks that the null or zero value means end of */
      /* file. For normal end of file, this returns 1, otherwise 0.       */

#ifdef  ParPrntFlag
      printf("yywrap in lsrc\n");
#endif

      return(TRUE);

} /* end yywrap */
```

'-' and '/' are included here in the string TOKEN definition

```
/*
*
* File Name   : retby.c
* Source      : /u/mdbs/greg/BE/RECP/retby.c
* Info        : This file contains functions to process an RETRIEVE request with
                by-clause in ABDL format
*
*/

#include <stdio.h>
#include "flags.def"
#include "commdata.def"
#include "recproc.def"
#include "recproc.ext"
#include "msg.def"
#include "dblocal.def"
#include "tmpl.def"
#include "tmpl.ext"
#include "beno.def"

extern char *strcpy();

BY_HASH_FUNC(rid, res_len, result)
   struct RP_rid_info  *rid;
   char result[];
   int  res_len;
{
  struct ResultBuffer    *RB_ptr;
  struct by_hashing_info *hash_ptr;
  struct by_block        *blk_ptr;
  char buffer[ResBufSize+1];

      /* Updating the templates after each INSERT has not been */
      /* implemented; so max and min are hardwired */
  int value_type, max = 5000, min = 0, index = 0, range = 0, i;

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

  RB_ptr = rid->RB_pointer;
  for (i = 0; i < res_len; i++)
```

```
    buffer[i] = result[i];

  hash_ptr = rid->RP_by_hash;

  if (hash_ptr->new_flag) {
    hash_ptr->new_flag = FALSE;

/*Added 22 October 1993 for retrives with no ordering*/
if(!(strcmp(hash_ptr->by_attr,"000")))
{
strcpy(hash_ptr->by_attr,rid->RP_ri_tmpl_ptr->rt_entry[1].attr_name);
strcpy(rid->RP_by_hash->by_attr,rid->RP_ri_tmpl_ptr->rt_entry[1].attr_name);
}

printf("prior to while in retby.c index is %d and hash is %s\n",index,hash_ptr->by_attr);
fflush(stdout);

    while (strcmp(rid->RP_ri_tmpl_ptr->rt_entry[index].attr_name,
                                        hash_ptr->by_attr) != 0)
      /*while they are not equal incr the index*/
{
printf("In while in retby.c index is %d and hash is %s\n",index,rid->RP_ri_tmpl_ptr-
>rt_entry[index].attr_name);
fflush(stdout);
      index++;
}
printf("In while in retby.c index is %d and hash is %s\n",index,rid->RP_ri_tmpl_ptr-
>rt_entry[index].attr_name);
fflush(stdout);

    if (rid->RP_ri_tmpl_ptr->rt_entry[index].value_data_type == 's') {

#ifdef ByDebug
printf("The value type is STRING\n");
#endif
      value_type = STRING;
    } else {

      if ((max - min) < NUMBER_OF_BUCKETS)
        value_type = SMALL_INTEGER;
      else {
        range = ((max - min) / NUMBER_OF_BUCKETS) + 1;
        value_type = LARGE_INTEGER;
```

```
        }
      }

#ifdef ByDebug
      printf("value type = %d\n",value_type);
      fflush(stdout);
#endif

      /* initialize hash_ptr */
      hash_ptr->value_type = value_type;
      hash_ptr->min = min;
      hash_ptr->range = range;
    } /* end if */

    /* call the function which will hash and store the records */
    BY_HASH_RECORD(rid, buffer, hash_ptr->min, hash_ptr->range,
                hash_ptr->value_type, res_len);

#ifdef EnExFlag
    printf("Exit BY_HASH_FUNC\n");
    fflush(stdout);
#endif
} /* end BY_HASH_FUNC */


BY_HASH_RECORD(rid, result_buffer, min, range, value_type, res_len)
     char result_buffer[ResBufSize + 1];
     struct RP_rid_info  *rid;
     int  min, range, value_type, res_len;
{
    struct by_hashing_info  *hash_ptr;

    int attr_index = 0,
        i = 0,
        temp, temp_index = 0,
        bucket_num;

    char c1, c2,
         attr_value[AVLength +1],
         tempstr[ANLength+1];

    hash_ptr = rid->RP_by_hash;
```

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

  for (;;) {
    while (tempstr[temp_index++] = result_buffer[attr_index++])
      ;
    temp_index=0;

    if (!strcmp(tempstr, hash_ptr->by_attr))
      break;
  }

  /* get the attribute value for the 'BY' attribute name */
  while (attr_value[i++] = result_buffer[attr_index++])
    ;

   /* now, hash on that value */
  if (value_type == STRING) {
    c1 = attr_value[0];
    c2 = attr_value[1];
  } else
    temp = atoi(attr_value);
/* Due to hard wiring and values in other section
Changed 25 October 1993 to get reasonable values*/

if (temp>5000 || temp<-5000)
 temp%=5000;

  switch(value_type) {
    case STRING:
          if (c2 <= '9' && c1 >= 'A')    /* 'A' added by N.YILDIRIM 5/24/95 */
            bucket_num = ((c1 - 'A') * 36) + (c2 - '0');
          else if (c1 <= '9' && c2 <= '9')      /* Added by N.YILDIRIM 5/24/95 */
            bucket_num = (('M' - c1) * 36) + (c2 - '0');
          else
            bucket_num = ((c1 - 'A') * 36) + (c2 - 'a') + 10;
          break;
    case SMALL_INTEGER:
          bucket_num = temp - min;
          printf("bucket value in Hash Record is %d\n",bucket_num);
          break;
```

Modified 'bucket_num' for Object-oriented interface

```
    case LARGE_INTEGER:
            bucket_num = ((temp - min) / range);
  } /*end switch */

  StoreByRecord(hash_ptr, bucket_num, attr_value, result_buffer, res_len);

#ifdef EnExFlag
  printf("Exit BY_HASH_RECORD \n");
  fflush(stdout);
#endif
} /* end BY_HASH_RECORD */


StoreByRecord (hash_ptr, bucket_num, attr_value, record, res_len)
   struct by_hashing_info  *hash_ptr;
   int bucket_num, res_len;
   char attr_value[], record[];
{
  struct by_block *blk_ptr,
                  *new_ptr,
                  *AllocByBlock();
  int r_index = 0, c_index = 0;

#ifdef EnExFlag
  printf("Enter StoreByRecord \n");
  fflush(stdout);
#endif
#ifdef ByDebug
  printf("Hashing into bucket number %d",bucket_num);
#endif

    /* find the end of the list, add a new bucket */
printf ("\nBUCKET_NUM IN RETBY.c  %d\n", bucket_num);
  new_ptr = AllocByBlock();
  if (!hash_ptr->hash_table[bucket_num]) {
    blk_ptr = new_ptr;
     printf("In Store record and using a new ptr\n");
    hash_ptr->hash_table[bucket_num] = blk_ptr;
  } else {    /* link a new bucket in */
    blk_ptr = hash_ptr->hash_table[bucket_num];
     printf("In Store record and using an old new ptr\n");
    while (blk_ptr->next_block)
       blk_ptr = blk_ptr->next_block;
```

```
    blk_ptr->next_block = new_ptr;
    blk_ptr = new_ptr;
  }

  /* transfer the record into the block */

    /* put the BY value first in the bucket */
  while (blk_ptr->contents[c_index++] = attr_value[r_index++])
    ;

    /* copy the record into the bucket */
  for (r_index=0; r_index < res_len; r_index++)
    blk_ptr->contents[c_index++] = record[r_index];

  blk_ptr->length = c_index;

#ifdef EnExFlag
  printf("Exit StoreByRecord \n");
  fflush(stdout);
#endif
} /* end StoreByRecord */


struct by_block *AllocByBlock()
{
  struct by_block *blk;

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

  if ((blk = (struct by_block *)
        malloc(sizeof(struct by_block))) == NULL) {
    printf("PROBLEM WITH MALLOC IN BUCKET BLOCK");
    sleep(ErrDelay);
  } else {
    blk->length = 0;
    blk->next_block = NULL;
  }

#ifdef EnExFlag
  printf("Exit AllocByBlock\n");
```

```
  fflush(stdout);
#endif

  return(blk);

} /* end AllocByBlock */


Send_Hash_Info(hash_ptr,rid)
   struct by_hashing_info  *hash_ptr;
   struct RP_rid_info  *rid;
{
  struct by_block  *bucket_ptr,
                   *first,
                   *blk_ptr,
                   *Sort_Buckets();

  char bucket_str[MAX_BUCKET_SIZE + 1],
      msg_data[ResBufSize + 1];

  int i, j, m = 0, bucket_num;

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

  msg_data[m++] = hash_ptr->value_type + '0';
  for (bucket_num=0; bucket_num < NUMBER_OF_BUCKETS; bucket_num++) {
    bucket_ptr = hash_ptr->hash_table[bucket_num];
    if (bucket_ptr == NULL)
      continue;

      /* sort only if two or more buckets */
    if (bucket_ptr->next_block)
      first = Sort_Buckets(bucket_ptr,hash_ptr->value_type);
    else first = bucket_ptr;

    do {
       /* send the message if no more room in the buffer */
      if ((ResBufSize - m - MAX_BUCKET_SIZE - 7) < first->length) {
        msg_data[m++] = '\0';
        RB$PUT_SEND(rid->RB_pointer,msg_data,m);
```

```
        m = 0;
        msg_data[m++] = hash_ptr->value_type + '0';
      }

      /* put the bucket number in the message first */
      num_to_str(bucket_num, bucket_str);
      j = 0;
      while (msg_data[m++] = bucket_str[j++])
        ;

        /* move the data into the message queue */
      for (j=0; j < first->length; j++)
        msg_data[m++] = first->contents[j];

       /* always add a mark at the end of a bucket */
      msg_data[m++] = BUCKET_MARK;
      first = first->next_block;

    } while (first);
  }

  msg_data[m++] = '\0';

  if (rid->RP_agg_ptr) {
    msg_data[m++] = '~';
    msg_data[m++] = '\0';
    msg_data[m++] = EOResult;
  }

  RB$PUT_SEND(rid->RB_pointer,msg_data,m);

  /* make sure the message is sent before any agg op results are sent */
  if (rid->RP_agg_ptr)
    rid->RB_pointer->RB_next_empty_pos = ResBufSize;

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

} /* end Send_Hash_Info */
```

```
struct by_block *Sort_Buckets(first, type)
   struct by_block  *first;
   int type;
{
  struct by_block  *ptr, *top, *temp, *blk[MAX_OVERFLOW];
  int i, num = 0, result, changed=TRUE;

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

     /* move the linked list into an array of pointers */
  ptr = first;
  do {
    blk[num++] = ptr;
    ptr = ptr->next_block;
  } while (ptr);

     /* sort the array of pointers using simple bubble sort */
  while (changed) {
    changed = FALSE;
    for (i=0; i < (num-1); i++) {
      switch (type) {
        case STRING: result = strncmp(blk[i]->contents,
                                blk[i+1]->contents, MAX_COMPARE);
                break;
        case SMALL_INTEGER: result = small_comp(blk[i]->contents,
                                        blk[i+1]->contents);
                        break;
        case LARGE_INTEGER: result = large_comp(blk[i]->contents,
                                        blk[i+1]->contents);
      } /* end switch */

      if (result > 0) {
        changed = TRUE;
        swap(blk, i);
      }
    } /* end for */
  } /* end while */

     /* move the pointers back to a linked list */
  top = temp = blk[0];
```

```
  for (i=1; i < num; i++) {
    temp->next_block = blk[i];
    temp = temp->next_block;
  }
  temp->next_block = NULL;

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

  return(top);

} /* end Sort_Buckets */


swap(blk, pos)
   struct by_block  *blk[MAX_OVERFLOW];
   int pos;
{
  struct by_block  *temp;

  temp = blk[pos];
  blk[pos] = blk[pos + 1];
  blk[pos + 1] = temp;

} /* end swap */


small_comp(num1_st, num2_st)
   char  *num1_st, *num2_st;
{
  int  num1, num2;

  num1 = atoi(num1_st);
  num2 = atoi(num2_st);
  if (num1 == num2)
    return(0);
  else if (num1 > num2)
    return(1);
  else  return(-1);
} /* end small_comp */
```

```
large_comp(num1_st, num2_st)
   char  *num1_st, *num2_st;
{
  long  num1, num2, atol();

  num1 = atol(num1_st);
  num2 = atol(num2_st);
  if (num1 == num2)
    return(0);
  else if (num1 > num2)
    return(1);
  else  return(-1);
} /* end large_comp */


free_bucket(ptr)
   struct by_block  *ptr;
{
  if (ptr->next_block)
    free_bucket(ptr->next_block);
  free(ptr);
}


#ifdef ByDebug

show(buf, num)
  char buf[];
  int num;
{
  int i;

  for (i=0; i<num; i++) {
    if (buf[i] == '\0')
      putchar(' ');
    else  putchar(buf[i]);
  }
  printf("\n");
  fflush(stdout);
}
```

```
/*
*
* File Name   : updp.c
* Source      : /u/mdbs/greg/BE/RECP/updp.c
* Info        : This file contains functions to process an UPDATE request in
                ABDL format
*
*/

#include <stdio.h>
#include "flags.def"
#include "beno.def"
#include "commdata.def"
#include "recproc.def"
#include "recproc.ext"

$UPD_PROCESSING(rid, addr)
  /* Used for processing an UPDATE */
  struct  ReqId  *rid;
  struct  addr_definition  *addr;
{
  int    indexB = 0, /* index for track buffer */
         rec_no = 0,
         satisfied;
  struct  RP_rid_info  *RP_ri_ptr, *FIND_RP_ri();
  int    any_updated = FALSE;
  struct  old_new_value  onv;

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

  /* map to the region */
  map_dio_reg(rid, addr);

  /* get a pointer to the RP_rid_info entry for the request */
  RP_ri_ptr = FIND_RP_ri(rid);

  /* Select records in TB one by one */
  while (TB[indexB] != EOTrack) {
    rec_no++;
    /* check to see if record exists */
```

```
    if (TB[indexB+2] == rec_exist) {
      /* Check whether the record satisfies the QUERY */
      CHK_QUERY(&(RP_ri_ptr->RP_ri_req), TB, indexB+2,
                    &satisfied, RP_ri_ptr->RP_ri_tmpl_ptr);
      if (satisfied) {
        /* the record satisfies the query */
        any_updated = TRUE;
        /* increment number of records (in the track) being updated */
        INC_URCPT(RP_ri_ptr, addr);
        /* Update the record (the value of attribute-being-modified in */
        /* TB[indexB] is changed to the new value) */
        $UPD_RECORD(&(RP_ri_ptr->RP_ri_req), RP_ri_ptr->RP_ri_tmpl_ptr,
                                    indexB+2, &onv);
        /* $UPD_RECORD returns old-new-value; the track will be stored on
         * disk after DM responds whether records have changed cluster;
         * send a message to DM asking whether the record changes cluster */
        ONV$RP_S(rid, addr, rec_no, &onv, RP_ri_ptr->RP_ri_dbid);
      }

    } /* end if (TB[indexB+2] == rec_exist) */
    nxtoffst(&indexB);
    if(indexB<0)
        break;
  } /* end while */

  /* unmap from the region */
  TB = NULL;

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

  return(any_updated);

} /* end $UPD_PROCESSING */
```

$UPD_RECORD(request, tmpl_ptr, indexB, onv)

```
  /* Update a record. Construct old_new_value for the record which is sent to
   * DM to determine whether or not the record has changed cluster. */
  struct rtemp_definition *tmpl_ptr;
  struct REQtbl_definition *request;
  int    indexB;
```

```
  struct old_new_value *onv;
{
  int  i, j, k = 0, m, n, pos, pos_t2, modtype, typ_ans, t1val, x, y, att1st;

  char oprnd,typ2attr[ANLength],typ2val[AVLength];
  unsigned short old_len, rec_len;
  char len_change, temp[RecSize+1];

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

#ifdef pr_flag
  print_track();
#endif

  /* find the modifier type code in the request */
  for (i = 6; request->req_tbl[i][0] != EOQuery; i++)
    ;
  i++;
  modtype = request->req_tbl[i][0] - '0';

#ifdef pr_flag
  switch (modtype) {
    case MT0: printf("MT0"); break;   /* attribute = constant */
    case MT1: printf("MT1"); break;   /* attribute = f(attribute) */
    case MT2: printf("MT2"); break;   /* attribute = f(attribute1) */
    case MT3: printf("MT3"); break;   /* attribute = f(attribute1) of Query */
    case MT4: printf("MT4"); break;   /* attribute = f(attribute1) of Pointer */
    default: printf("ERROR"); break;
  }
#endif

  /* update the record according to modifier type */
  switch (modtype) {

    case MT0:  /* type-0 update */
      ++i; /* point to attribute being modified */
      /* find the position of the attribute-being-modified in the record */
      for (pos = 0; pos < tmpl_ptr->no_entries && strcmp(request->req_tbl[i],
                    tmpl_ptr->rt_entry[pos].attr_name); pos++)
        ;
```

```
if (!strcmp(request->req_tbl[i], tmpl_ptr->rt_entry[pos].attr_name)) {
  /* construct old_new_value for the record and update the record
   * (record is updated by copying the new value of the attribute
   * being modified and the rest of the record to a temporary
   * location ('temp'), then copying back 'temp' into the record)*/
  strcpy(onv->onv_attr, request->req_tbl[i]);
  /* skip over to the field of the attribute-being-modified in
   * the record */
  j = pos;
  for (k = indexB + 1; j > 0; k++)
    if (TB[k] == EOField)
      j--;

  m = 0;
  /* copy the new value of the attribute being modified into */
  /* old_new_value and 'temp' */
  i += 2; /* point to the new value for attr-being-modified */
  strcpy(onv->onv_new_value, request->req_tbl[i]);
  j = 0;
  while (request->req_tbl[i][j])
    temp[m++] = request->req_tbl[i][j++];
  /* copy EOField */
  temp[m++] = EOField;
  /* copy the old value of the attribute being modified into */
  /* old_new_value */
  j = 0;
  for (n = k; TB[n] != EOField; ++n)
    onv->onv_old_value[j++] = TB[n];
  onv->onv_old_value[j++] = '\0';
  ++n;
  /* copy the rest of the record */
  while ((temp[m++] = TB[n++]) != EORecord)
    ;

  /* did the UPDATE alter the length of the record? */
  len_change = strlen(onv->onv_old_value)-strlen(onv->onv_new_value);
  if (len_change){
    rec_len = (TB[indexB-2] << 8) + TB[indexB-1];
    old_len = rec_len;
    rec_len -= len_change;
    TB[indexB-2] = (rec_len >> 8);
    TB[indexB-1] = (rec_len & 0xFF);
```

```
      x = indexB; /* get current offset */
      if (len_change > 0) {
        /* value DECREASED in length */
        /* read to beginning of next record */
        while (TB[x] != EORecord)
          x++;
        /* left shift rest of track the amount of decrease */
        while (TB[x] != EOTrack) {
          TB[x-len_change] = TB[x];
          x++;
        }
        TB[x-len_change] = EOTrack;
        for (y = 1; y <= len_change; y++)
          TB[(x-len_change)+y] = 0;
      } else {
        /* value INCREASED in length */
        /* read to end of track */
        y = indexB + old_len - 2;
        while (TB[x] != EOTrack)
          x++;
        /* right shift the rest of track the amount of increase */
        while (x >= y) {
          TB[x-len_change] = TB[x];
          x--;
        }
      }
    }
    /* copy temp back into TB */
    m = 0;
    while ((TB[k++] = temp[m++]) != EORecord)
      ;
  } else {
    printf("***** System error in $UPD_RECORD (RecP) ***** \n");
    printf("case Upd modifier type 0\n");
    printf("...can't find the attribute being modified\n");
    sleep(ErrDelay);
  }
  break;

case MT1:
  ++i; /* point to attribute being modified */
  /* find the position of the attribute-being-modified in the record */
  for (pos = 0; pos < tmpl_ptr->no_entries && strcmp(request->req_tbl[i],
```

```
                tmpl_ptr->rt_entry[pos].attr_name); pos++);

        if (streq(request->req_tbl[i], tmpl_ptr->rt_entry[pos].attr_name)) {
          /* record is updated by calculating the old value of the attribute
           * with the value supplied */

          /* attr name of value to use to modify by (type 1 so the same) */
          strcpy(onv->onv_attr, request->req_tbl[i]);
          i+=1;      /* nxt el in req_tbl is = to a_b_m so skip it */

          /* skip over to the value of the attribute-being-modified in
           * the record */
          j = pos;
          for (k = indexB + 1; j > 0; k++)
            if (TB[k] == EOField)
              j--;

          /* get old value */
          j = 0;
          for (n = k; TB[n] != EOField; ++n)
            onv->onv_old_value[j++] = TB[n];
          onv->onv_old_value[j++] = '\0';
          typ_ans=str_to_num(onv->onv_old_value);

          i++;
          while(request->req_tbl[i][0]!=EOExpr){
            if(streq(request->req_tbl[i],"000")){
                  i++;
                  oprnd=request->req_tbl[i][0];
                  i++;
                  t1val=str_to_num(request->req_tbl[i]);
                  i++;
                  att1st=TRUE;
            } else{
                  t1val=str_to_num(request->req_tbl[i]);
                  i++;
                  oprnd=request->req_tbl[i][0];
                  i+=2;
                  att1st=FALSE;
            }
#ifdef pr_flag
                  printf("oprnd = %c\n",oprnd);
                  printf("t1val = %d\n",t1val);
```

```
            printf("att1st= %d\n",att1st);
            fflush(stdout);
#endif
        switch (oprnd){
            case '+':
                typ_ans+=t1val;
                break;
            case '-':
                if(att1st)
                    typ_ans=typ_ans-t1val; /* Modified by N.YILDIRIM 04/14/95 */
                                /* because of incorrect output    */
                else
                    typ_ans=t1val-typ_ans; /* Modified by N.YILDIRIM 04/14/95 */
                                /* because of incorrect output    */
                break;
            case '*':
                typ_ans *=t1val;
                break;
            case '/':
                if(att1st)
                    typ_ans=typ_ans/t1val; /* Modified by N.YILDIRIM 04/14/95 */
                                /* because of incorrect output    */
                else
                    typ_ans=t1val/typ_ans; /* Modified by N.YILDIRIM 04/14/95 */
                                /* because of incorrect output    */
                break;
          }
        }

        /* copy the new value of the attribute being modified into */
        /* old_new_value and 'temp' */
        num_to_str(typ_ans,temp);
        strcpy(onv->onv_new_value,temp);
        /* copy EOField */
        strcat(temp,"$");
#ifdef pr_flag
  printf("temp = %s\n", temp);
  fflush(stdout);
#endif
        ++n;
        /* copy the rest of the record */
        m = strlen(temp);
        while ((temp[m++] = TB[n++]) != EORecord)
```

Order of operands are corrected for substraction and division

```
    ;
  /* did the UPDATE alter the length of the record? */
  len_change = strlen(onv->onv_old_value)-strlen(onv->onv_new_value);
  if (len_change) {
    rec_len = (TB[indexB-2] << 8) + TB[indexB-1];
    old_len = rec_len;
    rec_len -= len_change;
    TB[indexB-2] = (rec_len >> 8);
    TB[indexB-1] = (rec_len & 0xFF);
    x = indexB; /* get current offset */
    if (len_change > 0) {
      /* value DECREASED in length */
      /* read to beginning of next record */
      while (TB[x] != EORecord)
        x++;
      /* left shift rest of track the amount of decrease */
      while (TB[x] != EOTrack) {
        TB[x-len_change] = TB[x];
        x++;
      }
      TB[x-len_change] = EOTrack;
      for (y = 1; y <= len_change; y++)
        TB[(x-len_change)+y] = 0;
    } else {
      /* value INCREASED in length */
      /* read to end of track */
      y = indexB + old_len - 2;
      while (TB[x] != EOTrack)
        x++;
      /* right shift the rest of track the amount of increase */
      while (x >= y) {
        TB[x-len_change] = TB[x];
        x--;
      }
    }
  }
  /* copy temp back into TB */
  m = 0;
  while ((TB[k++] = temp[m++]) != EORecord)
    ;
} else {
  printf("***** System error in $UPD_RECORD (RecP) ***** \n");
  printf("case Upd modifier type 1\n");
```

```
        printf("...can't find the attribute being modified\n");
        sleep(ErrDelay);
        }
      break;

    case MT2:
      ++i; /* point to attribute being modified */
      /* find the position of the attribute-being-modified in the record */
      for (pos = 0; pos < tmpl_ptr->no_entries && strcmp(request->req_tbl[i],
                    tmpl_ptr->rt_entry[pos].attr_name); pos++);

      if (streq(request->req_tbl[i], tmpl_ptr->rt_entry[pos].attr_name)) {
        /* record is updated by calculating the old value of a different
         * attribute with the value supplied */

        /* attr name of value to use to modify by (type 2 so diff) */
        strcpy(onv->onv_attr, request->req_tbl[i]);
        i++;
        strcpy(typ2attr,request->req_tbl[i]);
#ifdef pr_flag
           printf("onv->onv_attr = %s\n",onv->onv_attr);
           printf("attr name of value to modify by = %s\n",typ2attr);
           fflush(stdout);
#endif

        /* skip over to the value of the attribute-being-modified in
         * the record */
        j = pos;
        for (k = indexB + 1; j > 0; k++)
          if (TB[k] == EOField)
            j--;
        /* get old value */
        j = 0;
        for (n = k; TB[n] != EOField; ++n)
          onv->onv_old_value[j++] = TB[n];
        onv->onv_old_value[j++] = '\0';

      /* find the position of the "new attr" in the record */
        for(pos_t2=0;pos_t2<tmpl_ptr->no_entries && strcmp(typ2attr,
                    tmpl_ptr->rt_entry[pos_t2].attr_name); pos_t2++);

        /* find the position of value in the record */
        j=pos_t2;
```

```
        for(old_len=indexB + 1;j>0;old_len++)
          if (TB[old_len] == EOField)
            j--;
        /* extract the value */
        for (pos = old_len; TB[pos] != EOField; ++pos)
          typ2val[j++] = TB[pos];
        typ2val[j++] = '\0';
        typ_ans=str_to_num(typ2val);

        i++;
        while(request->req_tbl[i][0]!=EOExpr){
          if(streq(request->req_tbl[i],"000")){
                i++;
                oprnd=request->req_tbl[i][0];
                i++;
                t1val=str_to_num(request->req_tbl[i]);
                i++;
                att1st=TRUE;
          } else {
                t1val=str_to_num(request->req_tbl[i]);
                i++;
                oprnd=request->req_tbl[i][0];
                i+=2;
                att1st=FALSE;
          }
#ifdef pr_flag
                printf("oprnd = %c\n",oprnd);
                printf("t1val = %d\n",t1val);
                printf("att1st= %d\n",att1st);
                fflush(stdout);
#endif
        switch (oprnd) {
                case '+':
                        typ_ans+=t1val;
                        break;
                case '-':
                        if(att1st)
                            typ_ans=typ_ans-t1val; /* Modified by N.YILDIRIM 04/14/95 */
                                        /* because of incorrect output    */
                        else
                            typ_ans=t1val-typ_ans; /* Modified by N.YILDIRIM 04/14/95 */
                                        /* because of incorrect output    */
                        break;
```

Order of operands are corrected for substraction and division

```
                case '*':
                        typ_ans *=t1val;
                        break;
                case '/':
                        if(att1st)
                            typ_ans=typ_ans/t1val; /* Modified by N.YILDIRIM 04/14/95 */
                                        /* because of incorrect output    */
                        else
                            typ_ans=t1val/typ_ans; /* Modified by N.YILDIRIM 04/14/95 */
                                        /* because of incorrect output    */
                        break;
            }
        }

        /* copy the new value of the attribute being modified into */
        /* old_new_value and 'temp' */
        num_to_str(typ_ans,temp);
        strcpy(onv->onv_new_value,temp);
        /* copy EOField */
        strcat(temp,"$");
        ++n;
        /* copy the rest of the record */
        m = strlen(temp);
        while ((temp[m++] = TB[n++]) != EORecord)
          ;
#ifdef pr_flag
  printf("temp = %s\n", temp);
#endif
        /* did the UPDATE alter the length of the record? */
        len_change = strlen(onv->onv_old_value)-strlen(onv->onv_new_value);
        if (len_change) {
          rec_len = (TB[indexB-2] << 8) + TB[indexB-1];
          old_len = rec_len;
          rec_len -= len_change;
          TB[indexB-2] = (rec_len >> 8);
          TB[indexB-1] = (rec_len & 0xFF);
          x = indexB; /* get current offset */
          if (len_change > 0) {
            /* value DECREASED in length */
            /* read to beginning of next record */
            while (TB[x] != EORecord)
              x++;
            /* left shift rest of track the amount of decrease */
```

```
        while (TB[x] != EOTrack) {
          TB[x-len_change] = TB[x];
          x++;
        }
        TB[x-len_change] = EOTrack;
        for (y = 1; y <= len_change; y++)
          TB[(x-len_change)+y] = 0;
      } else {
        /* value INCREASED in length */
        /* read to end of track */
        y = indexB + old_len - 2;
        while (TB[x] != EOTrack)
          x++;
        /* right shift the rest of track the amount of increase */
        while (x >= y) {
          TB[x-len_change] = TB[x];
          x--;
        }
      }
    }
    /* copy temp back into TB */
    m = 0;
    while ((TB[k++] = temp[m++]) != EORecord)
      ;
  } else {
    printf("***** System error in $UPD_RECORD (RecP) ***** \n");
    printf("case Upd modifier type 1\n");
    printf("...can't find the attribute being modified\n");
    sleep(ErrDelay);
    }
  break;

case MT3:
  /* <<< TBC >>> */
  break;

case MT4:
  /* <<< TBC >>> */
  break;

default:
  printf("***** System error in $UPD_RECORD ***** \n");
  printf("Invalid modifier type \n");
```

```
      sleep(ErrDelay);

  } /* end case */

#ifdef pr_flag
  printf("struct old_new_value:\n");
  printf("\tonv_attr = %s\n", onv->onv_attr);
  printf("\tonv_old_value = %s\n", onv->onv_old_value);
  printf("\tonv_new_value = %s\n\n", onv->onv_new_value);
  print_track();
#endif

#ifdef EnExFlag
  printf("Exit $UPD_RECORD\n");
  fflush(stdout);
#endif
}
```

INC_URCPT(RP_ri_ptr, addr)

```
  /* Increment number of records (in a track) that are updated */
  struct  RP_rid_info  *RP_ri_ptr;
  struct  addr_definition  *addr;
{
  int urcpt_ind = 0;

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

  /* find the RP_ri_urcpt (updated_rec_count_per_track) entry for the track */
  while (urcpt_ind < MAX_ADDRS_UPD)
    if (addr->cylinder_no ==
          RP_ri_ptr->RP_ri_addrs.as_addrs[urcpt_ind].cylinder_no &&
       addr->track_no ==
          RP_ri_ptr->RP_ri_addrs.as_addrs[urcpt_ind].track_no)
      /* entry found */
      break;
    else
      urcpt_ind++;

  if (urcpt_ind >= MAX_ADDRS_UPD) {
    printf("**** System error in INC_URCPT\n");
```

```
    printf("address was not found \n");
    sleep(ErrDelay);
  } else
    /* increment number of records (in the track) being updated */
    ++RP_ri_ptr->RP_ri_urcpt[urcpt_ind];

#ifdef EnExFlag
  printf("Exit INC_URCPT\n");
  fflush(stdout);
#endif
}
```

RPUPD2(rid, addr, rec_no, changed_flag)

```
  /* Used for UPDATEs after */
  /* DM responds whether a record has changed cluster */
  struct ReqId *rid;
  struct addr_definition *addr;
  int    rec_no;
  int    changed_flag;
{
  int    i, k, m, j;
  struct RP_rid_info *RP_ri_ptr,
                     *FIND_RP_ri();
  int    urcpt_ind;
  static char   record[RecSize + 1];
  struct rtemp_definition *tmpl_ptr;

#ifdef EnExFlag
  printf("Enter RPUPD2\n");
  fflush(stdout);
#endif

  /* map to the region */
  map_dio_reg(rid, addr);

  /* get a pointer to the RP_rid_info entry for the request */
  RP_ri_ptr = FIND_RP_ri(rid);

  if (RP_ri_ptr) {
    tmpl_ptr = RP_ri_ptr->RP_ri_tmpl_ptr;
    /* find the RP_ri_urcpt (updated_rec_count_per_track) entry for */
    /* the track */
    urcpt_ind = 0;
```

```
while (urcpt_ind < MAX_ADDRS_UPD)
  if (addr->cylinder_no ==
       RP_ri_ptr->RP_ri_addrs.as_addrs[urcpt_ind].cylinder_no &&
      addr->track_no ==
       RP_ri_ptr->RP_ri_addrs.as_addrs[urcpt_ind].track_no)
    /* entry found */
    break;
  else
    ++urcpt_ind;

if (urcpt_ind < MAX_ADDRS_UPD) {
  if (changed_flag) {
    /* The updated record has changed cluster; delete the original */
    /* record and insert a new record into the database. */
    m = (rec_no - 1) * RecSize;
    /* Delete the original record */
    TB[m] = rec_deleted;
    /* costruct the new record */
    j = 0;
    for (i = 0; i < tmpl_ptr->no_entries; i++) {
      /* copy the attribute name */
      k = 0;
      while (record[j++] = tmpl_ptr->rt_entry[i].attr_name[k++])
        ;
      /* copy the attribute value from the record in the track */
      ++m;
      while (TB[m] != EOField)
        record[j++] = TB[m++];
      record[j++] = '\0';
    } /* end for */

    /* put EORecord */
    record[j++] = EORecord;
    /* send the new record to REQP which will initiate the actions */
    /* required for the insertion of the record */
    RHCC$RP_S(RP_ri_ptr->RP_ri_dbid, rid, record);

  } /* end if (changed_flag) */

  /* decrement number of records in the track being updated */
  --RP_ri_ptr->RP_ri_urcpt[urcpt_ind];
  /* unmap from the region */
  TB = NULL;
```

```
      if (RP_ri_ptr->RP_ri_urcpt[urcpt_ind] == 0)
        TB_STORE(rid, &(RP_ri_ptr->RP_ri_addrs.as_addrs[urcpt_ind]));

    } else {
      printf("***** System error in RPUPD2 ***** \n");
      printf("no RP_ri_updated_rec_count_per_track entry \n");
      sleep(ErrDelay);
    }
  } else {
    printf("***** System error in RPUPD2 ***** \n");
    printf("no RP_rid_info entry \n");
    sleep(ErrDelay);
  }

#ifdef EnExFlag
  printf("Exit RPUPD2\n");
  fflush(stdout);
#endif
}
#ifdef pr_flag
print_track()
{
  int i = 0, zero_count = 0;
  printf("\n");
  while (TB[i] != EOTrack) {
    if (TB[i] < '!' || TB[i] > '~') {
      printf(" %d ", TB[i]);
      zero_count++;
      if (zero_count > 3)
        break;
    } else {
      printf("%c", TB[i]);
      zero_count = 0;
    }
    i++;
    if (!(i % 60))
      printf("\n");
  }
  printf("%c\n", EOTrack);
  fflush(stdout);
}
#endif
```

APPENDIX D- FLOW CHARTS FOR FOUR BASIC OPERATIONS OF THE SYSTEM

The following four pages describe the control flow of the software supporting four basic operations in CNTRL/TI, CNTRL/COMMON, CNTRL/REQP, and BE/DM processors of twelve. The first two pages are an overview of the execution of all four basic operations in CNTRL/TI. Basically, they describe the execution of Interface. The following pages show more detail of the execution of these four basic operations in important BE processors.

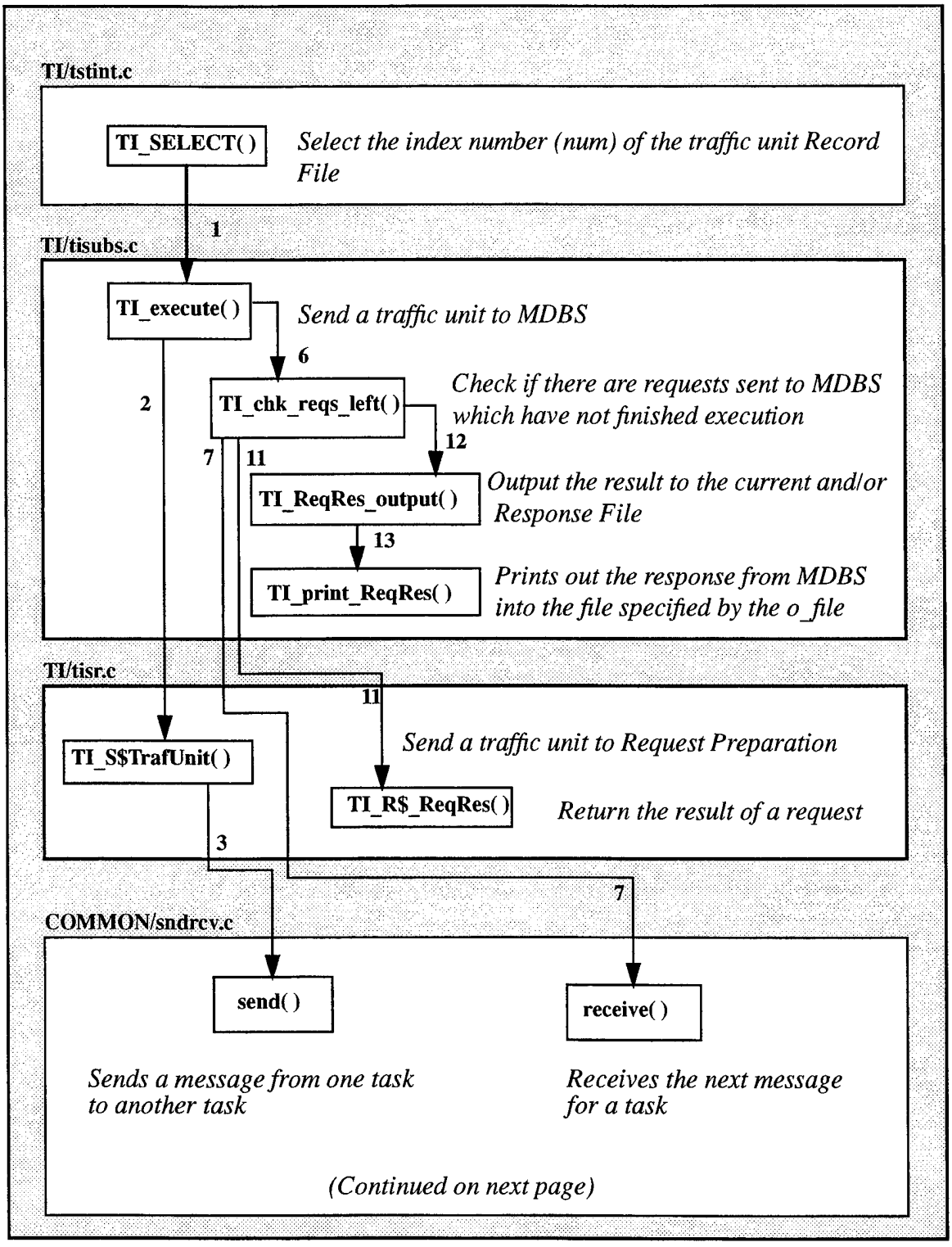


Figure 7: Execution of all Operations based on Function Calls (not all) in CNTRL/TI

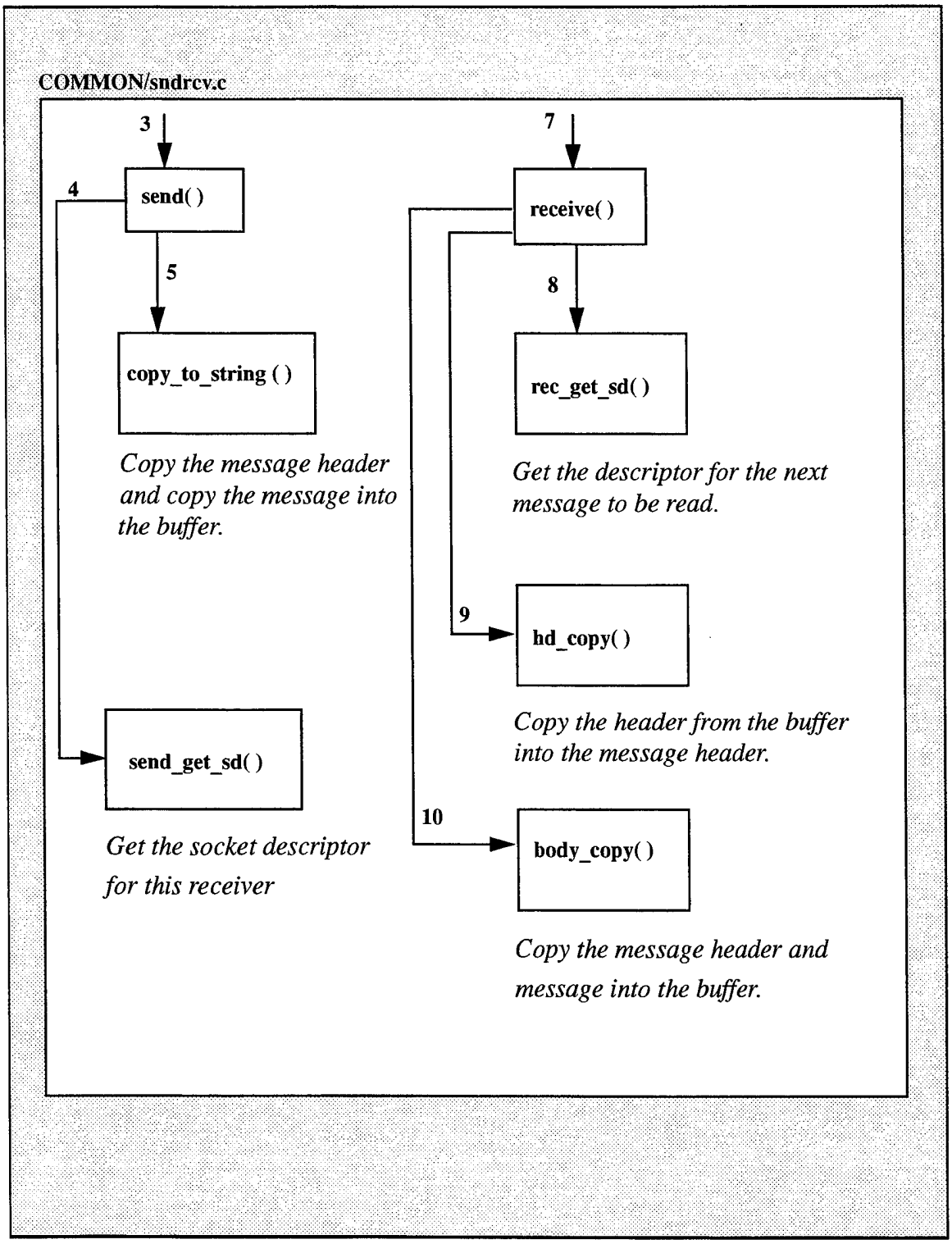


Figure 8: Execution of all Operations based on Function Calls in CNTRL/COMMON

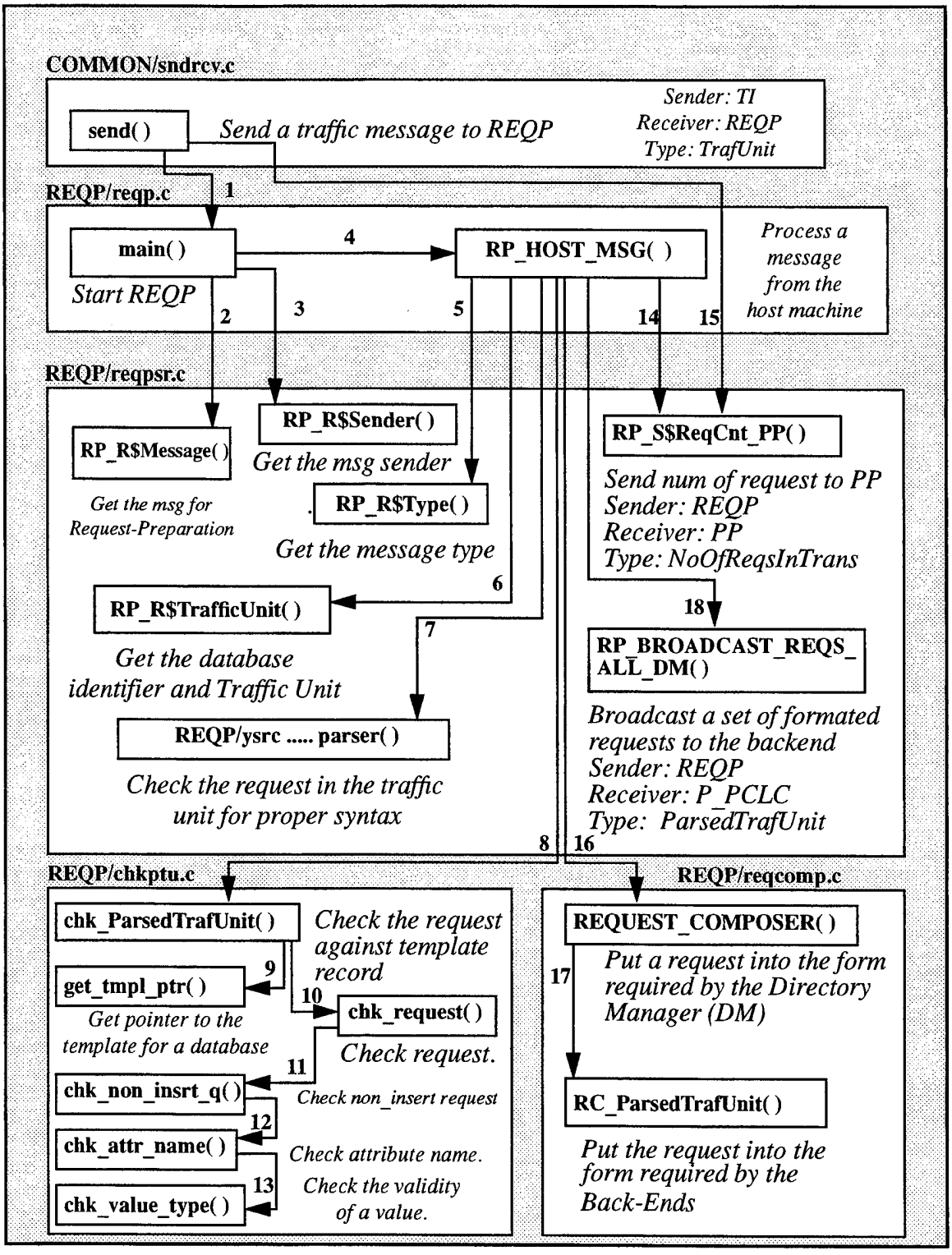


Figure 9: Execution of all Operations based on Function Calls (not all) in CNTRL/REQP

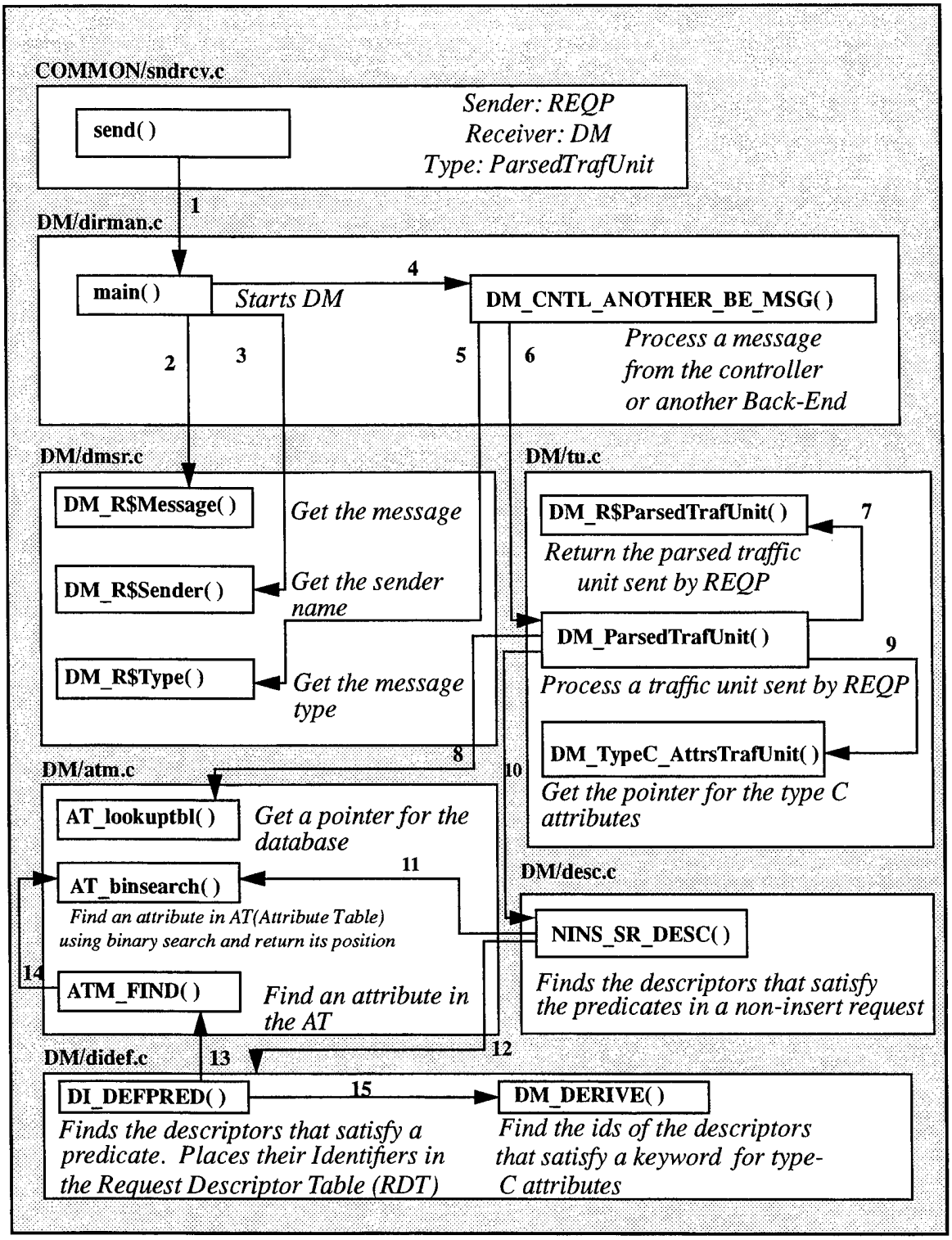


Figure 10: Execution of all Operations based on Function Calls (not all) in BE/DM

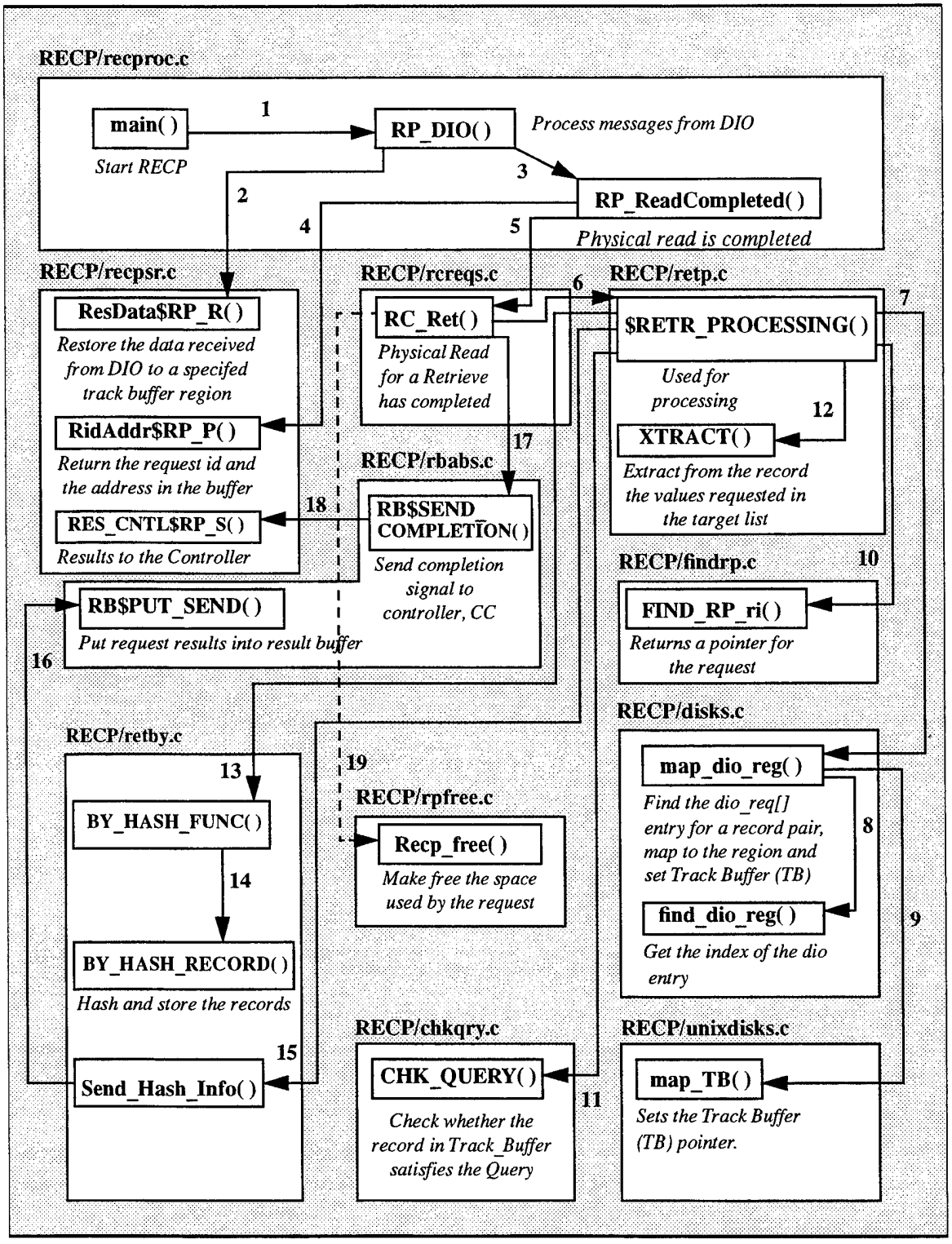


Figure 11: Execution of Retrieve based on Function calls (not all) in BE/RECP

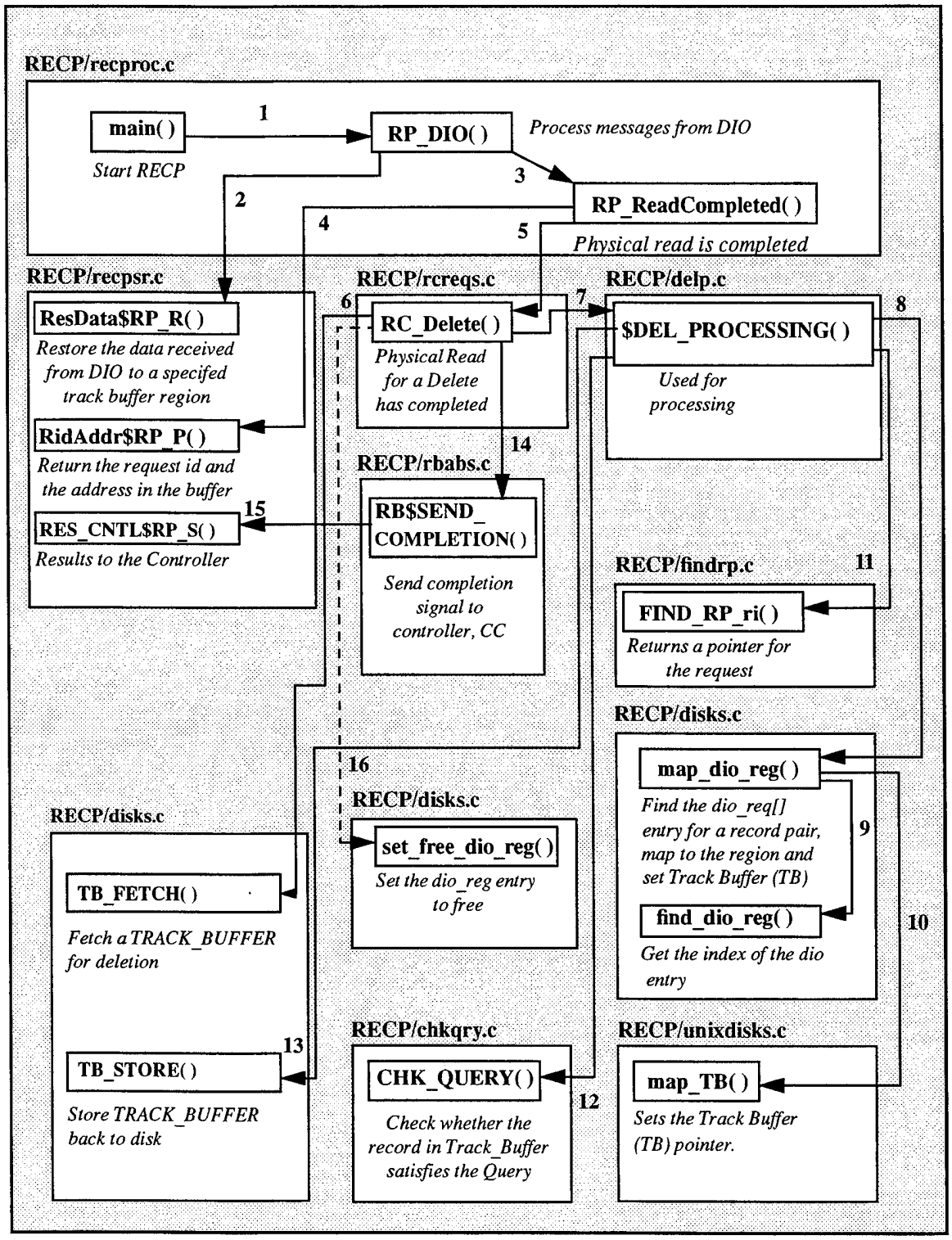


Figure 12: Execution of Delete based on Function calls (not all) in BE/RECP

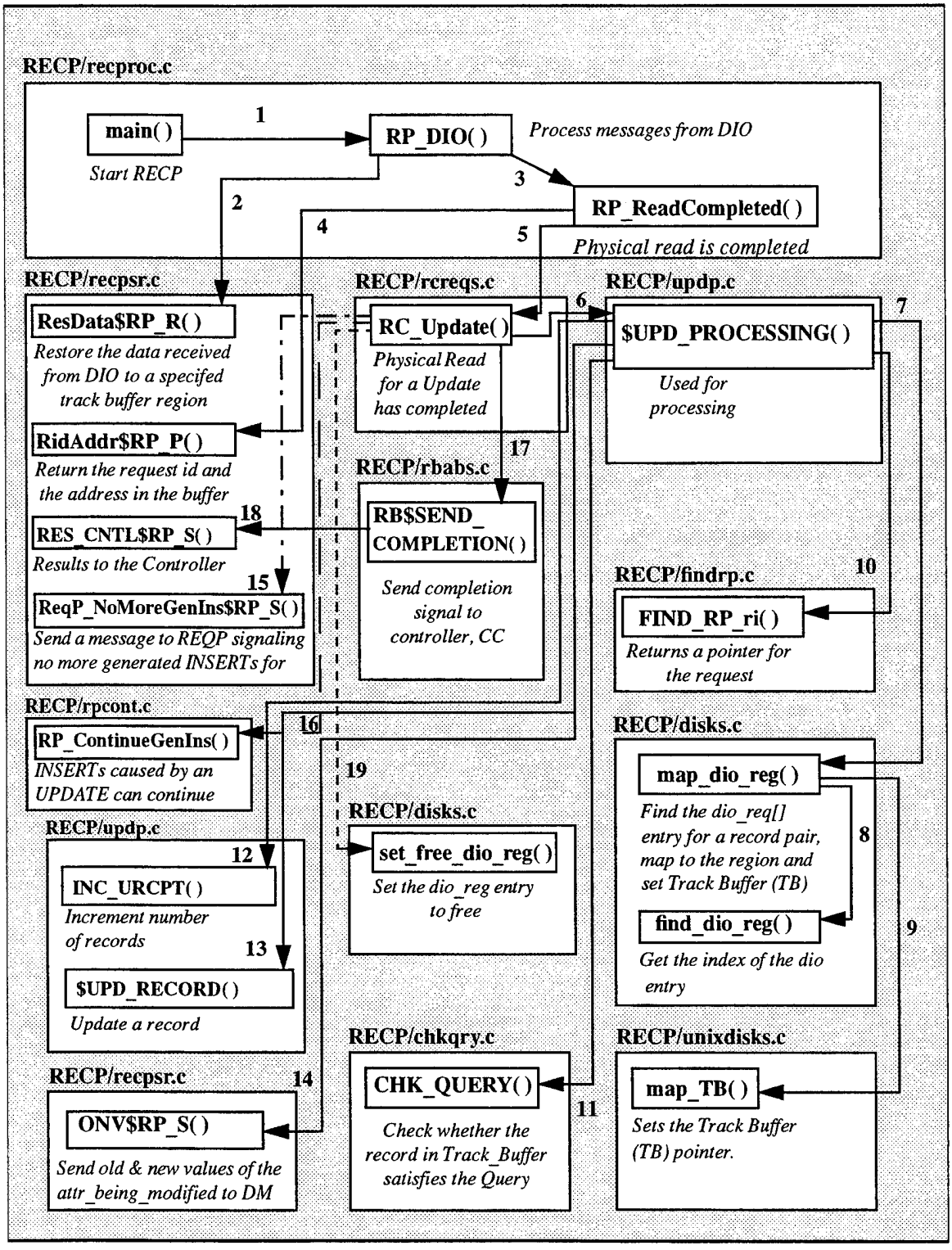


Figure 13: Execution of Update based on Function calls (not all) in BE/RECP

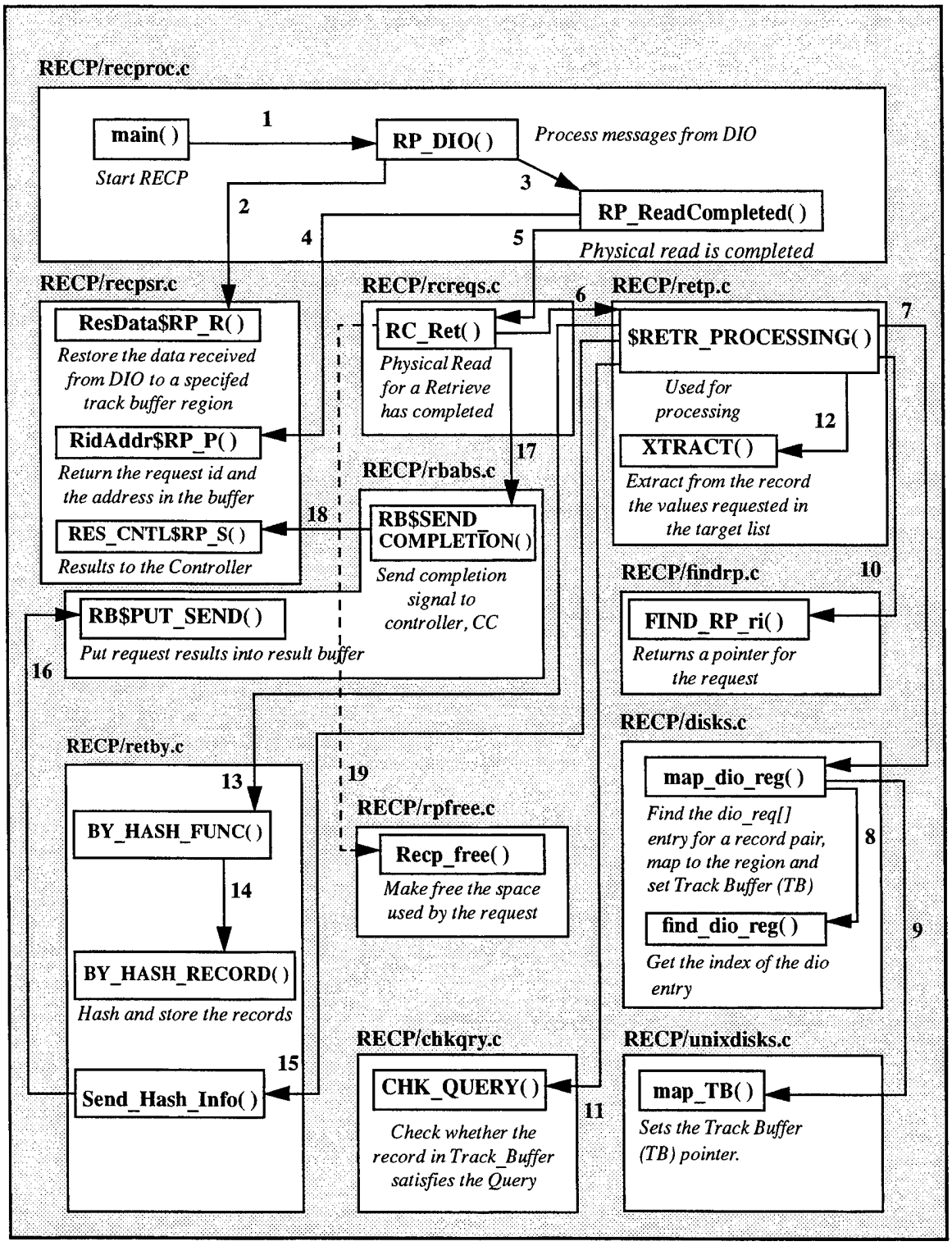


Figure 14: Execution of Retrieve-Common based on Function calls (not all) in BE/RECP

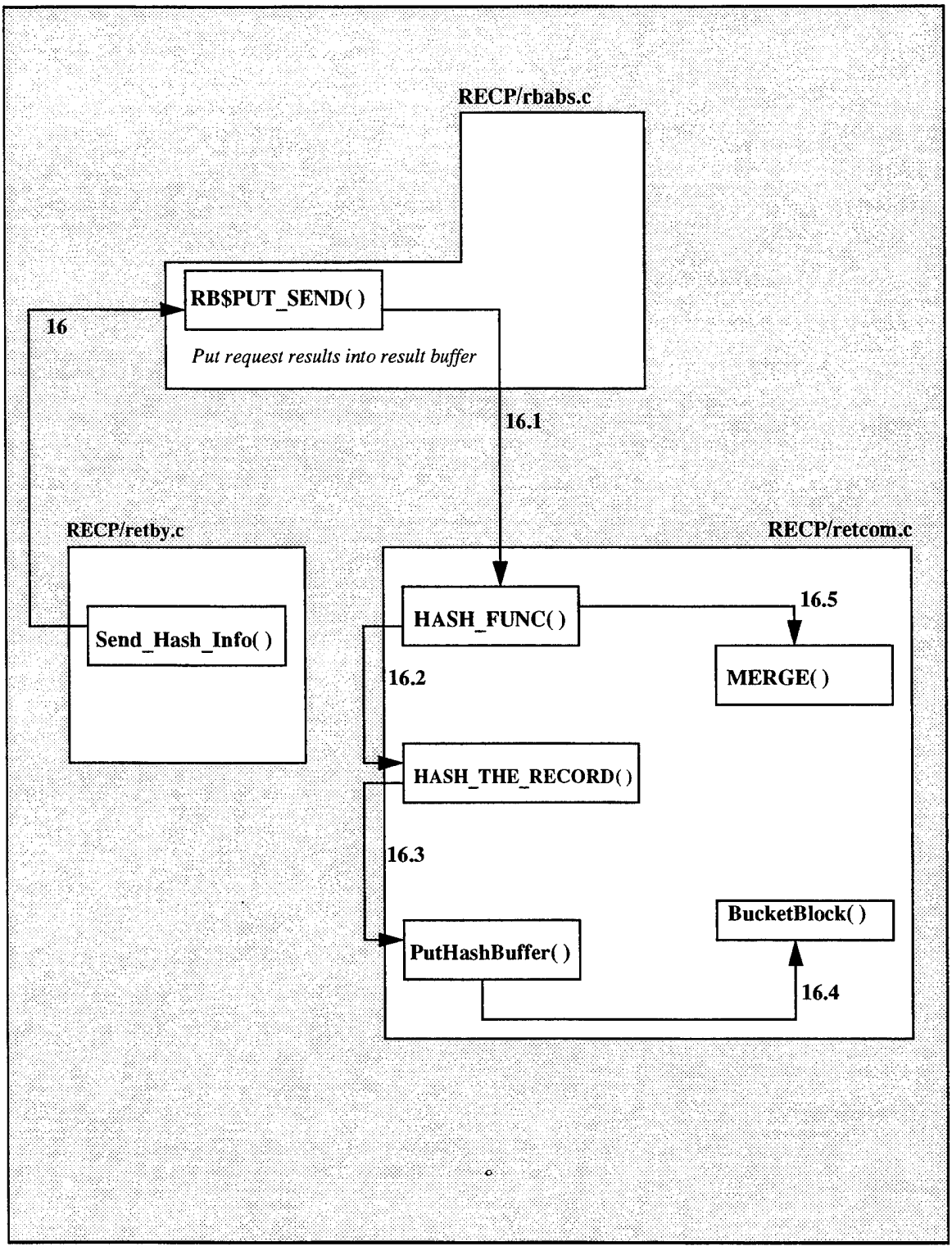


Figure 15: Continue, Retrieve-Common based on Function calls (not all) in BE/RECP

LIST OF REFERENCES

[1] Senocak, E., *The Design and Implementation of a Real-Time Monitor for the Execution of Compiled Object-Oriented Transactions (O-ODDL and O-ODML Monitor)*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1995.

[2] Hsiao, David K., "Interoperable and Multidatabase Solutions for Heterogeneous Databases and Transactions", a speech delivered at **ACM CSC '95**, Nashville, Tennessee, March 1995.

[3] Hsiao, David K., "A Parallel, Scalable, Microprocessor-Based Database Computer for Performance Gains and Capacity Growth", *IEEE Micro*, December 1991.

[4] Ramirez, L. and Tan, R., M., *The Design and Implementation of a Compiler for the Object-Oriented Data Definition Language (O-ODDL Compiler)*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1995.

[5] Barbosa, C. and Kutlusan, A., *The Design and Implementation of a Compiler for the Object-Oriented Data Manipulation Language (O-ODML Compiler)*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1995.

[6] Watkins, S., H., *A Porting Methodology for Parallel Database Systems*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1993.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   Cameron Station
   Alexandria, VA 22304-6145

2. Dudley Knox Library 2
   Code 52
   Naval Postgraduate School
   Monterey, CA 93943-5101

3. Dr Ted Lewis, Chairman, Code CS 2
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943

4. Dr David K. Hsiao, Code CS/HS 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943

5. Dr C. Thomas Wu, Code CS/KA 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943

6. Ms Doris Mleczko 2
   Code P22305
   Weapons Division
   Naval Air Warfare Center
   Pt Mugu, CA 93042-5001

7. Ms Sharon Cain 2
   NAIC/SCDD
   4115 Hebble Creek Rd
   Wright Patterson AFB, OH 45433-5622

8. Lt Robert E. Clark 1
   4763 Tapestry Dr
   Fairfax, VA 22032

9. LtJg Necmi Yildirim 1
Deniz Kuvvetleri Komutanligi
06600 Bakanliklar, Ankara TURKEY