ICOT Technical Report: TR-064

TR-064

Query Processing Flow on RDBM Delta's Functionally-Distributed Architecture

by

Shigeki Shibayama, Takeo Kakuta,
Nobuyoshi Miyazaki, Haruo Yokota, and
Kunio Murakami

April, 1984

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191 - 5
Telex ICOT J32964

Institute for New Generation Computer Technology

Query Processing Flow

on RDBM Delta's Functionally-Distributed Architecture

Shigeki SHIBAYAMA, Takeo KAKUTA, Nobuyoshi MIYAZAKI,

Haruo YOKOTA, and Kunio MURAKAMI

ICOT Research Center

Institute for New Generation Computer Technology,

Mita Kokusai Bldg. 21F, 1 - 4 - 28, Mita

Minato-ku, Tokyo 108

Query Processing Flow on RDBM Delta's Functionally-Distributed Architecture

Shigeki SHIBAYAMA, Takeo KAKUTA, Nobuyoshi MIYAZAKI,
Haruo YOKOTA, and Kunio MURAKAMI

ICOT Research Center

Abstract

This paper presents the implementation details of a relational database machine Delta being developed at the Institute for New Generation Computer Technology (ICOT). We focus on how a transaction comprised of Delta access commands are received, analyzed, managed and executed by using an example. As a result, this paper is mainly concerned with the software configuration and functionalities distributed among the processors which comprise Delta. A detailed description of relational database engine (RDBE), one of the major hardware features, will be given in another paper [Sakai 84]. An approach to use Delta as a part of the knowledge base machine research tool is also briefly described.

1. Background and Introduction

In the scope of Japan's Fifth Generation Computer System (FGCS) project, a relational database machine is being developed. The relational database machine (Delta) is planned to be finished at the end of the initial 3-year stage of the project. Then it is offered for use as a backend store to sequential inference machine (SIM) users in the intermediate 4-year stage. The inference machine is another system planned to be completed in the initial stage. An FGCS prototype is shown in Fig.1.1. Delta receives concurrent queries issued from SIM's via an Ethernet-like local area network. Delta will be used not only as a backend store in the network, but also as a research

tool for the investigation of a knowledge base machine. An experiment system where a SIM and Delta are tightly-coupled will be used for this purpose.

Delta is basically a relational database system featuring a hardware relational-type database processor and a large capacity semiconductor disk. It is a functionally-distributed multi-processor system. It is also provided with most of database management facilities, for example, multi-user support and recovery functions.

In this paper, the basic architectural concept is described in chapter 2 and hardware configuration being implemented is described in chapter 3. Chapter 4 describes the software configuration and how a transaction is decomposed, dispatched and executed across the subsystems in detail. Knowledge base machine research plan is briefly described in chapter 5.

2. Architecture

Delta's conceptual architecture is shown in Fig.2.1. Delta adopted a functionally-distributed architecture for a relational database machine design. The reason why this approach was taken was that necessary functions to cooperatively perform database operations, for example, query translation, interfacing with hosts, relational algebra operations and database storage management, could be separated. By separating the functions, efficient hardware and software implementation can be applied to each of the functional

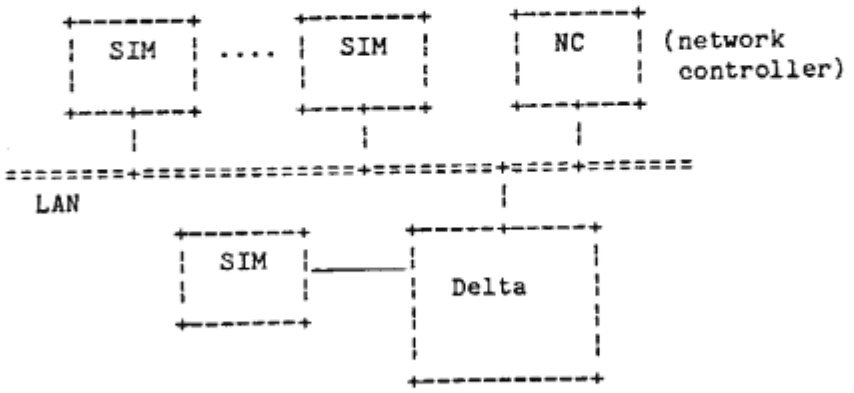

Fig.1.1 An FGCS Prototype

subsystems. We divided the database functions into five categories. Those are (1) host interfacing, (2) query analysis and hardware resource management in the subsystem level, (3) relational database processing, (4) database storage management and (5) system supervising. We provided an Interface Processor (IP), a Control Processor (CP), Relational Database Engines (RDBE), a Hierarchical Memory (HM) and a Maintenance Processor (MP), respectively, to perform the above distributed functions. The implementations of the subsystems are described in the following chapter. The functions distributed to the subsystems are as follows:

(1) Interface Processor (IP)

- **Local area network interface** (physical and logical)

  This interface provides a loosely-coupled connection between the geographically distributed SIM's.

- **A tightly-coupled interface**

  This provides an experimental tightly-coupled interface with a SIM physically close to Delta.

- **Concurrent command-sequence management**

  Delta provides a multi-user environment necessary for a backend special-

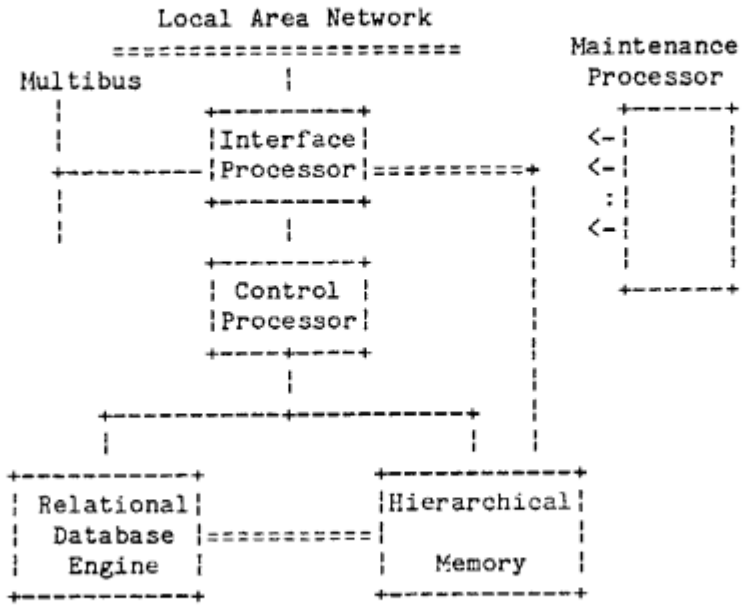

Fig.2.1 Delta global architecture

purpose processor in a network. IP acts as the front-end of Delta and resolves concurrent command-sequence arrivals from multiple SIM's.

- o **Tuple transfer** (tuple insertion and result tuple response)

  IP has a high-bandwidth transfer path to HM needed to perform this function.

(2) **Control Processor (CP)**

- o **Transaction management**

  Transaction management function is responsible for grasping each transaction's status and for database resource management.

- o **Command-tree management**

  For each transaction, Delta command (command-tree) analysis, subcommand generation, subcommand execution control are performed by this manager.

- o **Dictionary/directory management**

  In Delta's terminology, dictionary is a set of meta-relations which users can refer to. Directory is an internal data structure which CP's software refers to. We discriminated these for an efficiency reason. However, the consistency between them should be maintained at transaction commit or abort times.

- o **Recovery management**

  At system-down times and users' transaction aborting, CP works in accordance with MP and HM to maintain the database.

(3) **Relational Database Engine (RDBE)**

- o **Relational database processing**

  To execute relational algebra operations and set operations fast, this portion should have special-purpose hardware and software. We observed that the hardware two-way merge sorting could be used to carry out fast relational database operations, for example, join with no restriction on operable attributes. Our first idea for constructing a relational database machine was to build a fast relational database engine

according to this algorithm and to make use of it.

By fast, we mean, first, that the database operations are performed overlapped with the data transfer. This simultaneous execution of data transfer and relational database operation is useful when the data amount is fairly large. A mere transfer of data, in that case, will consume much time. We wanted to avoid the visible data transfer time as much as possible. Second, the transfer speed should be fast. To enjoy the fast engine hardware, a bi-directional high-bandwidth path to the storage portion is necessary. This requirement led to the incorporation of large semiconductor memory with fast channels into the Delta system (HM subsystem).

(4) Hierarchical Memory (HM)

o Cache and moving-head-disk management

To conceive a fast database machine, good care has to be taken in the database storage portion. As Delta is determined to accommodate current technologies for an implementation, the storage device cannot be thought without moving-head-disks. The problem here is that without using a large cache with a wise replacement algorithm, it is not possible to satisfy the fast relational database operations performed at RDBE. The cache and moving-head-disk space management should be separated to a dedicated portion. Therefore, HM is responsible for preparing and receiving source and result relations (data) at a high transfer rate to/from RDBE.

o Data recovery

Besides the space management, another important role assigned to HM is the data recovery functions. As HM is the dedicated storage portion, it is better to make HM take care of the data recovery functions. This should be done, of course, together with CP and MP's recovery management portions.

(5) Maintenance Processor (MP)

o **System supervising**

The Maintenance Processor (MP) is responsible for the system supervising. As Delta is a large computer complex, some global system maintenance portion is needed. The following list shows the tasks in a little more detail.

o **Start-up and shut-down of each subsystem**

o **Subsystem status management**

o **Operator command execution**

o **Delta system state management**

o **Database loading and saving**

o **Statistical data collection**

3. Hardware Configuration

The actual hardware configuration of Delta system is shown in Fig.3.1. It is the best to design special-purpose processors for the specific requirements to implement the conceptual architecture to work most efficiently. For example, CP could be built to have a special instruction set to handle database locking and unlocking capability, to manipulate dictionary/directory and to cope with transaction concept. This choice, however, is not practical. The hardware design is concentrated on the most influential portions to obtain required performance within the environment where Delta is placed. The hardware design is thus focused upon the implementation of RDBE.

RDBE's hardware organization is shown in Fig.3.2. The two-way merge-sorter and the merger are responsible for most relational algebra operations and set operations. The merge-sorter is a one-dimensional special-purpose processor array for pipelined merge-sorting [Todd 78]. The merge-sorter can sort 2**N items in (2*2**N + N) time units where a time unit is a time to transfer one item. Merger merges input data streams and performs relational database processing algorithm on them. The input to the merger is sorted by the former stage sorter. The IN module is responsible for appropriately reordering the item's fields to place the sort key at the top of the item. Fields order

readjustment is performed within the merger unit when the item is output from it. A set of four RDBE's are installed for parallel transaction execution. Each RDBE works independently to execute separate queries or they can work cooperatively to execute a single relational database operation. The details of RDBE's load distribution is not determined yet. An RDBE has a general-

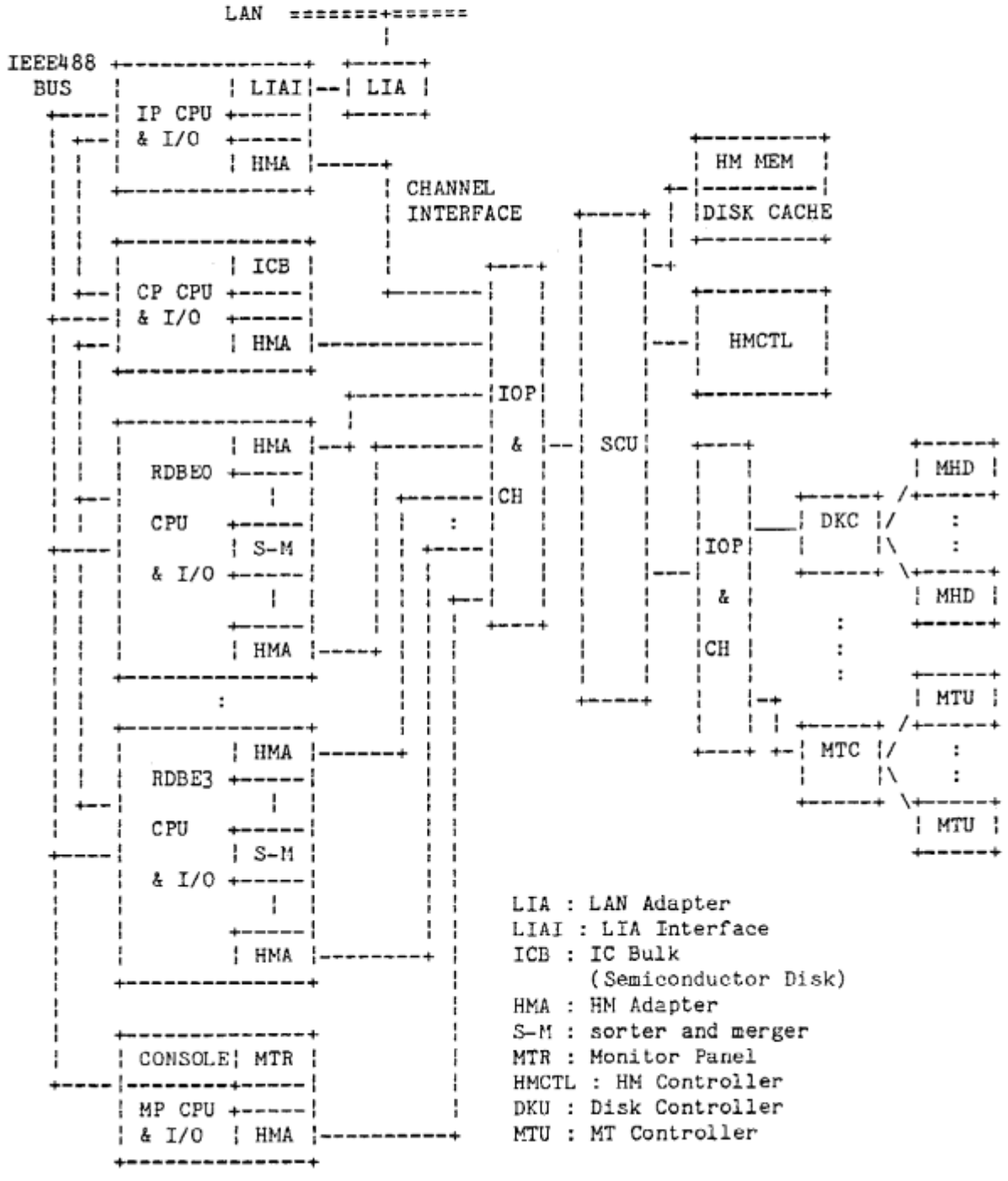

FIG.3.1 Delta Hardware Configuration

purpose mini-computer CPU with 512KB main storage for controlling the hardware portion and performing a part of a variety of RDBE commands which hardware portion does not support. The IN module, sorter and merger are controlled by the mini-computer as I/O devices.

IP, CP and MP are comprised of the same general-purpose mini-computers as RDBE. The main storage sizes of IP, CP and MP are 1MB, 1MB and 512KB, respectively. Each processor is provided with a local disk storage for the operating system use. CP's disk is a semiconductor disk storage. IP and MP are provided with the Winchester-type fixed disks. CP's control program stores temporarily the directory and other tables in the semiconductor disk. This is adopted to rapidly access the information swapped back in the secondary storage.

HM is implemented using a larger general-purpose CPU as its controller. The main storage of the CPU is used both for the program storage and the disk cache. The size of the main storage is 128MB. Instead of developing a new dedicated disk cache storage, we decided to make the main storage as large as possible and simulate the large semiconductor disk cache. The disk cache size and replacement algorithm can be altered for later system parameters tuning. The main storage is made non-volatile (at least from software point of view) by an emergency power supply to avoid disk accesses invoked by a write-through storage management and to facilitate the database recovery tasks. The moving-head-disks, the lowest storage hierarchy, have a capacity over 2GB per drive. Eight such drives are attached to the I/O channels; thus total storage capacity of about 20GB is provided.

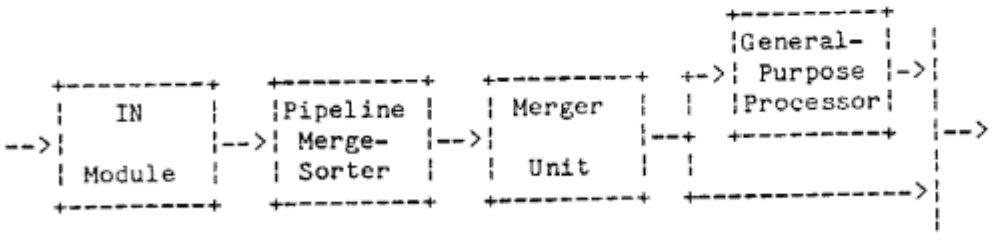

Fig. 3.2 RDBE Schematic Configuration

CP, IP, RDBE and MP form a processors group called the RDBM supervisory processing subsystem (RSP for short). While HM in itself is a different subsystem. Communication between the RSP constituents and HM is done via a standard channel interface. The transfer speed of the channel is maximum of 3MB/second. An RDBE is provided with two independent channels to HM for dedicated input and output uses. The other RSP constituents are provided with a single channel for bi-directional communication. Communication between RSP constituents are done using the IEEE488 buses. The transfer speed of the bus is about 150KB/second. As shown in Fig.3.1, the buses are independently installed between the RSP constituents. The amount of information flown between RSP constituents are small compared with that between HM and RSP. So the concern is communication overhead time rather than the burst transfer speed.

4. Function Distribution of Delta

4.1 Software Configuration

IP, CP and MP consist of the same CPU board. There are some pieces of software which are common to all these subsystems. Most principal one is the operating system. We adopted the same operating system to them. The operating system is a modified version of an existing one. The IEEE488 bus driver and HM adapter driver are included in the operating system. The IEEE488 bus driver is used to perform inter-subsystem communication except HM. In IP, the IEEE488 driver is also used as the LIA (LAN Adapter) physical level interfacing. RDBE has the same CPU board as the controller. However, its software does not have to perform concurrent operations, so an operating system is not used.

One of the major operating system modifications is the expansion of segments which the operating system occupies. The memory management is done on 64Kbyte segment basis.

(i) IP

IP is responsible for communicating with SIM's in the LAN environment. IP is connected to LIA physically by the IEEE488 bus. The physical level LIA interface in IP is performed by the **LIA communication task**. The logical level LIA interface is performed by the **IP control task**. IP has multiple **command-tree-buffers** for receiving plural command-trees sent concurrently to it. IP's **command-tree management task** aligns the command-trees sent from a host in arrival order into a command-tree-buffer. The **command-tree management task** then forwards the command-trees in a buffer to CP using the **CP communication task**. As the physical connection between IP and CP is also the IEEE488 bus, this task mainly consists of the IEEE488 bus driver, identical to LIA physical level interface.

Usually, the Delta command-trees in a transaction are taken out serially from the command-tree-buffer. However, asynchronous-type commands such as **sense-status** and **abort-processing** are served as soon as possible when they arrive at IP and recognized.

(ii) **CP**

CP is responsible for controlling transactions, managing database and Delta resources, command analysis, subcommand issuance, dictionary/directory (D/D) management, rollback and statistical data collection.

When CP is triggered by the IP's **command-tree management task** via the **CP communication task** with the transfer of a **start-transaction** command, a **command-tree processing task** is created. A **command-tree processing task** corresponds to a transaction. The task is killed upon transaction termination. An operating system facility (**transaction supervisor task**) is notified of the states of command-tree processing tasks.

The **command-tree processing task** gets command-trees one after another. It analyzes the command-tree, and if necessary, locks permanent relations at execution time. It then generates a sequence of subcommands for RDBE and HM referring to the directory for schema checking. After all the commands are translated into corresponding RDBE and HM subcommand sequences, a **command-tree**

processing task forwards the sequences to **RDBE and HM communication task** to send them to the subsystems. At execution time, by checking the responses from RDBE and HM, the **command-tree processing task** determines the parameters appeared in the generated subcommand sequence. If the resultant temporary relation of a command is found to contain, for example, no tuples at all, the subsequent subcommand sequence is skipped.

Dictionary/directory (D/D) management task is responsible for maintaining the directory for internal use and dictionary as users' reference. The difference of dictionary and directory is summarized in table 4.1. Dictionary relations are comprised of two meta-relations, "Relations" and "Attributes" relations. Essentially, dictionary and directory holds the same schema information. Unnecessary attributes in the dictionary for the schema checking, however, are not included in the directory. The reasons why we separated the dictionary and directory are (1) the efficiency in look-up, and (2) the concurrent access control. Usually, we decided to lock relations for resolving the concurrency control. If the dictionary is also locked during a look-up, locking conflicts will be frequent. Instead, we use the directory for finer concurrency control. HM supports the directory storage in it (directory area) separate from disk cache or program area. CP's semiconductor disk caches the directory for repeated use. If necessary, CP obtains directory pages from HM by directory-access subcommands.

As Delta is used in a network environment, there is a consistency problem

table 4.1 Dictionary and Directory

name	definer updater	reference	external interface	implementation	HM interface
dictionary	host* Delta	host	relation	permanent relation	permanent relation
directory	Delta	Delta	not defined	linked data	page

* : Only qualified attributes can be updated by hosts.

of dictionaries. Each Delta control program in hosts will hold a part of Delta dictionary for itself. A Delta control program has no means to know the updates to the dictionary done by another control program after taking it in. To save this situation, an attribute (**Redefined-at**) is provided to the **"Relations"** relation to check the obsoleteness of the dictionary. In the **"Redefined-at"** attribute, the time stamp of the last redefinition is kept. Every permanent relation access should be associated with this value. If it does not match the current value, Delta informs the host of the dictionary obsoleteness. The host should read the dictionary relations to continue.

The **concurrency control task** is a subtask of the **transaction supervisory task.** This task manages the concurrency among transactions. Concurrency control is done by locking the resources (relations). We selected the two-phase locking method. Permanent relations to be locked during a transaction can be explicitly locked in a **start-transaction** command, or they are automatically locked before command-tree execution process. Unlocking of all the relations is done at the end of a transaction whether it is a normal or an abnormal termination.

(iii) RDBE

RDBE's software system does not have an operating system. The whole program works as a single task. The software structure of RDBE is comprised of three layers. The first and uppermost layer is the **subcommand executive layer.** The second layer is the **I/O traffic control layer** and the third and lowest layer is the **interrupt handling layer.**

The **subcommand executive layer** receives subcommands from CP and MP, analyzes and executes the subcommands and sends responses to their senders. Basically, **subcommand executive** repeats this cycle. It calls the I/O traffic **control layer** as libraries to handle I/O devices. Another role that this layer performs is the extended operations which hardware core portion does not support. The operations are listed below:

(1) Filtering of result data by a complex criterion

The hardware core portion can perform operations only with a simple criterion such as a range search. The conjunction of one-term criteria, for example, is performed within the CPU.

(2) Arithmetic operations on attribute fields and result value assignment

(3) Aggregate operations on grouped streams

These operations are specified as a parameter of RDBE subcommands.

The I/O traffic control layer provides library routines for the **subcommand executive layer** and interrupt handling routines for the **interrupt handling layer**. Retriable I/O errors are for the most part retried here.

The **interrupt handling layer** does the following tasks:

(1) It sets up an environment in which RDBE control program runs at start-up time.

(2) It calls an interrupt handling routine provided by the **I/O traffic control layer** by checking the I/O device status when an interrupt occurs.

(3) It handles the other interrupts such as CPU internal interrupts.

(4) It provides libraries for manipulating special machine-dependent instructions such as PSW modifications.

iv) **HM**

HM's software configuration is as follows:

(1) **Operating system**

(2) **HM control program**

(2.1) **Control module**

This module is comprised of **HM task control submodule** which manages the HM multi-tasking and **RSP interface submodule** responsible for the RSP interfacing.

(2.2) **Subcommand processing module**

This module processes various HM subcommands issued from other subsystems. The subcommands are classified to the processing types.

(2.3) **Common function module**

This module is a collection of common functions used commonly among

other portions. Cluster management, log management, recovery management, memory management and disk space management are the principal tasks.

(2.4) **Initialization and termination module**

(3) **Support programs**

Utilities to support the software development and system operation

(4) **Test programs**

HM control program is the main portion to perform the database operations specified by the form of subcommands. Each module in the program is further divided into submodules for the logical unit of HM internal operations. These submodules run under the **HM task control submodule.** Subcommands from RSP are collectively managed by the **RSP interface submodule.** A **subcommand process submodule** is organized to correspond to a (set of) RSP subcommand(s). The **attribute definition/operation submodule** is, for example, activated to a set of attribute definition type subcommand sequence. HM is usually a passive subsystem. It is activated upon the receipt of a subcommand.

4.2 A Sample Transaction Processing Flow

As described in the architecture chapter, Delta performs a set of database machine requirements by the functionally-distributed configuration. By showing how a sample transaction is received, analyzed and executed, we will exemplify the functionalities distributed among Delta's subsystems.

A **transaction,** in Delta's terminology, is a bunch of command-trees beginning with a **start-transaction** command and ending with an **end-transaction** command. When the database update is specified, a transaction is the unit of update; an unsuccessful transaction, whether on account of users' specification of an **abort-transaction** command or a Delta's error, is rolled back to the previous database state before the **start-transaction** command, or the update to a successful transaction is fully done. In a read-only sequence of Delta commands, a transaction defines a scope in which intermediate relations (typically a result of a command-tree) are kept and used across

command-trees. Once a **commit-transaction** is received, only **input/output type commands** are accepted until an **end-transaction** command is received.

We will consider the following sample transaction. The permanent relations used are **company(company_name,location)** and **icot(name,age,company,laboratory, group)**. We consider a SQL-like query which means "Select the name, belonging laboratory and group of the persons with ICOT who was with a company in Yokohama," as:

```
start-transaction
select name, laboratory, group
from icot
where company = next
select company_name
from company
where location = [yokohama]
end-transaction.
```

This query is translated into the Delta command sequence by a translator (SIM software) as follows (the parameters are modified or abbreviated for readability):

(1.1) **Start-transaction**

define the beginning of a transaction scope

(2.1) **Selection(company,[2]=[yokohama],temp1)**

select from company relation where the second ([2]) attribute equals yokohama, put resultant relation into a temporary relation temp1

(2.2) **Projection(temp1,[1],temp2)**

project the temp1 relation against the first ([1]) attribute into temp2 relation

(2.3) **Projection(icot,[1,3,4,5],temp3)**

project icot relation against the first, third, fourth and fifth attributes into temp3

(2.4) **Natural-join(temp3,temp2,[2]=[1],temp4)**

natural-join temp3 and temp2 with the second and the first attribute, respectively, put into temp4

(2.5) **Projection(temp4,[1,3,4],int1)**

project temp4 against the first, third and fourth attributes into an intermediate relation int1

(3.1) **Commit-transaction**

freeze the transaction; freezing means to inhibit further modification of the resultant intermediate relation in case of a read-only transaction

(4.1) **Get(int1)**

fetch the intermediate relation (int1) from the top tuple

(5.1) **Get-next(int1)**

fetch the intermediate relation from the point next to the last get or get-next command

(6.1) **End-transaction**

conclude the transaction

This command-sequence is comprised of six command-trees; the first command number (before the period) denotes a command-tree number, the second number (after the period) denotes a command number within a command-tree. Each transaction-control command and input/output type command forms a command-tree only by itself. The six command-trees are packed with a chain identifier, command-tree identifiers and physical delimiters by SIM's translator software. This pack is called a command-tree chain. A command-tree chain is a unit which is transferred in a sequence of LIA commands (LAN packets). LAN interface adapter (LIA) is a LAN's subsystem responsible for the interfacing between SIM and Delta. Then they are forwarded to the SIM's network subsystem (NS), which deals with the local area network interfacing task.

NS uses the predetermined network protocols to send the command-trees to Delta. The first thing that NS must do is to form a **communication-group** to Delta per user database job. A user job is comprised of a set of serialized transactions.

The sample command sequence is translated into the RDBE and HM subcommand sequence shown in Fig.4.2.1.

Delta's internal storage schema is attribute-based. Every attribute is stored separately with tuple-identifier (tid) and tag fields. So the subcommand sequence does not directly correspond to the Delta command sequence. With a tuple-based schema, the order of the relational database operations affects the performance. Projections, for example, are preferably done before other operations for storage access saving. With an attribute-based schema, as the subcommands only work on attributes which appear explicitly in the operations, the order of projection commands are of less importance. Other optimizations, for example, selections before a join, are still effective with the attribute-based schema.

The subcommand group (SG for short) 1 in Fig.4.2.1 filters the tid fields

```
{ subcommand group 1 }                    ;tid1:company-location
 HM:prepare-qualified-buffer(buf1)        ;company-location = yokohama
 HM:prepare-buffer(buf2)                  ;output buffer
 RDBE:restrict                            ;company-location = yokohama
 RDBE-HM:start-stream-in(buf1)            ;only tid list is obtained
 RDBE-HM:start-stream-out(buf2)           ;buf2 = tid1
{ subcommand group 2 }                    ;sorting of tid1
 HM:prepare-buffer(buf3)                  ;buffer for sorted tid
 RDBE:sort                                ;sorting RDBE command
 RDBE-HM:start-stream-in(buf2)
 RDBE-HM:start-stream-out(buf3)           ;buf3 = tid1 (sorted)
{ subcommand group 3 }                    ;tid1:company-name
 HM:prepare-qualified-tid-buffer(buf4)    ;tid is in buf2
 HM:prepare-buffer(buf5)                  ;output buffer
 RDBE:restrict                            ;tid1 selection
 RDBE-HM:start-stream-in(buf4)            ;tid1:company-name
 RDBE-HM:start-stream-in(buf3)            ;tid1 (sorted)
 RDBE-HM:start-stream-out(buf5)           ;buf5 = tid1:company-name
{ subcommand group 4 }                    ;
 HM:prepare-qualified-buffer(buf6)        ;icot-company
 HM:prepare-buffer(buf7)                  ;tid triplets output buffer
 RDBE:join                                ;company-name = icot-company
 RDBE-HM:start-stream-in(buf5)            ;company-name
 RDBE-HM:start-stream-in(buf6)            ;icot-company
 RDBE-HM:start-stream-out(buf7)           ;tid triplets
{ subcommand group 5 }                    ;icot tid sort
 HM:prepare-buffer(buf8)                  ;output buffer
 RDBE:unique                              ;icot tid extract and sort
 RDBE-HM:start-stream-in(buf7)            ;tid triplets buffer
 RDBE-HM:start-stream-out(buf8)           ;buf8 = tid2 (sorted)
```

Fig. 4.2.1-(a) HM and RDBE subcommand sequence (to continue)

of the **company-location** attribute the value of which is "yokohama" in buf2. The SG2 sort the tid fields in buf2 into buf3. The SG3 joins the tid fields with the **company-name** attribute's tid field. The **company-name** attribute values are obtained in buf5. The SG4 joins the **company-name** attribute values in buf5 with **icot-company** attribute values. This corresponds to the **natural-join** command. The content of buf5 is a set of triplets of tid's, that is, the

```
{ subcommand group 6 }                    ;new-tid attach
 HM:prepare-qualified-tid-buffer(buf9)    ;icot-name
 HM:prepare-buffer(buf10)                 ;
 RDBE:restrict
 RDBE-HM:start-stream-in(buf9)            ;icot-name
 RDBE-HM:start-stream-in(buf7)            ;tid triplets
 RDBE-HM:start-stream-out(buf10)          ;selected icot-name
{ subcommand group 7 }                    ;new-tid attach
 HM:prepare-qualified-tid-buffer(buf11)   ;icot-lab
 HM:prepare-buffer(buf12)
 RDBE:restrict
 RDBE-HM:start-stream-in(buf11)
 RDBE-HM:start-stream-in(buf7)
 RDBE-HM:start-stream-out(buf12)          ;selected icot-lab
{ subcommand group 8 }                    ;new-tid attach
 HM:prepare-qualified-tid-buffer(buf13)   ;icot-group
 HM:prepare-buffer(buf14)                 ;
 RDBE:restrict
 RDBE-HM:start-stream-in(buf13)
 RDBE-HM:start-stream-in(buf7)
 RDBE-HM:start-stream-out(buf14)          ;selected icot-group
{ subcommand group 9 }                    ;for get preparation
 HM:prepare-buffer(buf15)                 ;
 RDBE:sort
 RDBE-HM:start-stream-in(buf10)           ;
 RDBE-HM:start-stream-out(buf15)          ;sorted-tid:icot-name
{ subcommand group 10 }                           ;for get preparation
 HM:prepare-buffer(buf16)
 RDBE:sort
 RDBE-HM:start-stream-in(buf12)
 RDBE-HM:start-stream-out(buf16)          ;sorted-tid:icot-lab
{ subcommand group 11 }                   ;for get preparation
 HM:prepare-buffer(buf17)
 RDBE:sort
 RDBE-HM:start-stream-in(buf14)
 RDBE-HM:start-stream-out(buf17)          ;sorted-tid:icot-group
{ subcommand group 12 }
 HM:transpose-to-tuples                   ;tuple reconstruction
 IP-HM:start-packet-in                    ;for get
 IP-HM:start-packet-in                    ;for get-next
```

The HM: and RDBE: prefixes denote that they are issued from CP.
The XX-HM: prefix denotes that it is issued from XX to HM.
The instructions next to the prefix are the subcommands.

Fig. 4.2.1-(b) HM and RDBE subcommand sequence (continued)

tid of company relation, the tid of icot relation and the new tid for the generated relation by the join. The buf5 is one form of an intermediate relation and the result of the second command-tree, specified as "int1". If the "int1" is used by another command, the following subcommand sequence is changed. In this example, the next command-tree is a **commit-transaction**, so the reconstruction process of tuples into an output form (usual tuple-based form) takes place. The SG5 extracts and sorts the tids of the icot relation from the triplet buffer for later tid-restriction, that is, selection of other attributes needed for the output relation. The SG6, 7 and 8 select the corresponding attributes, icot-name, icot-lab, icot-group, respectively, to the selected icot tids. Then the output attributes are sorted and reconstructed to a tuple form by the SG9 to 12. The buffers used in the transaction are dynamically released within the transaction. The buffers for the output form relation are released after the **end-transaction** command is received.

5. Delta in a logic programming research environment

Delta is used as a backend database storage in the intermediate Fifth Generation Computer project. Number of SIM machines are connected via an Ethernet-like local area network. Delta is accessed from a Delta interface program (called **RDBMS** for relational database machine management system) and SIM operating system. There are a number of interfaces to access Delta from SIM users.

There are several software layers through which users access Delta (Fig.5.1). The lowest layer is responsible for handling the physical network protocols. This layer corresponds to IP's LIA communication task. The next lowest layer is logical network protocol handling. Most part of the network layer is supported by the SIM operating system's network subsystem (NS).

The translator layer will be supported by the RDBMS. During the translation, RDBMS refers to the prefetched dictionary relation, forms and manages transactions, controls the access rights for security and performs related operations.

We expect the following usages by the users:

(1) Users handle logical Delta command directly. This uses the physical Delta command translation layer.

(2) Users define (at programming time) some special predicates (in logic programming sense) and write programs in a usual way. The relational algebra translation layer in RDBMS is responsible for both the interpretation of users' programs and the generation of relational database accesses in a Delta-command form. This method is presented in [Yokota 84].

(3) Users will have to write their own translation layer program if they want a special-purpose software application system. The output of the program is passed to the logical Delta interfacing layer.

(4) Users use only a high-level database query language, for example, SQL or QBE, built upon the RDBMS.

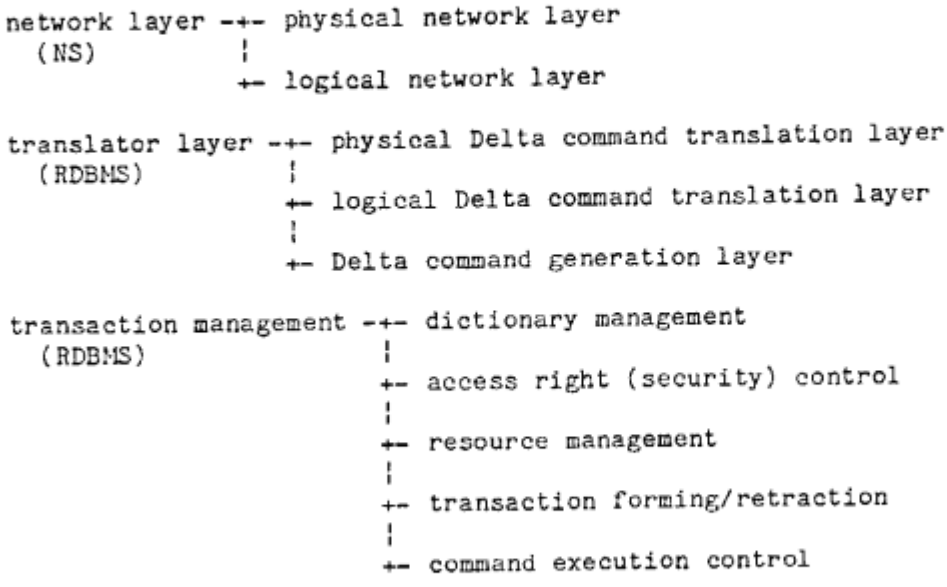

Fig. 5.1 RDBMS software layers

In the local area network environment, however, the LAN communication overhead is rather large for interactive database accesses. In the usage pattern (2), we proposed a method combining a logic programming language and relational database. In this method, Delta accesses are collected and issued in a bunch, considering the access characteristics of LAN.

We also plan to provide a more tightly-coupled SIM to Delta interface. We think that the tight coupling of inference part and database access part is necessary for future to make a knowledge base machine, a machine which deals with large amount of knowledge base and does inferences.

We think that there are a few candidates for the knowledge base machine approach based on logic programming languages.

(1) Addition of a virtual memory system to the SIM architecture and stores the whole knowledge base along with inference programs

(2) Setting of an interface between logic programming languages and knowledge base and coupling them tightly

(3) Investigation of totally new architecture to manipulate knowledge base and inferences

The first approach will perform poorly, because the internal data structure suited to perform inferences and the one to access a large amount of data are different, although for programmers this one-level storage treatment is most favorable. Efforts should be made to make the programmers' interface of knowledge base system close to this. However, this is not considered a good approach for an implementation.

The third approach does not seem to be ripe taking the current knowledge base research state. To think of a new architecture, a clear and sound principle is needed. This approach will be taken after the research stages to investigate such principles.

We think that the second approach is practical using the resources now available. The interface between the logic programming languages and the knowledge base portion should be investigated. We think that the base concept of the interface is relational database. The relational database is not, however, fully appropriate for the knowledge base model. We estimate that **unit-clause** interface is better for the knowledge base model in conjunction with the logic programming languages [Yokota 83]. This implies that the introduction of a simple **unification** capability into relational model is a natural extension of the relational model for a knowledge base model in logic programming.

To make an experimental knowledge base system under these assumptions, a tightly-coupled connection between SIM and Delta is needed. As LAN is supposed to cause some amount of transfer overhead, a more responsive and fast interface is necessary for the knowledge base machine research. We decided to use the buffer memory in the SIM system as a **communication memory**. The I/O buffer memory is connected to the IEEE796 bus (Multibus) under an I/O controller. By adding an interface board in the SIM system to access the bus and connecting it to IP, the path between Delta and SIM is established. We will make a software system in SIM for a close connection between a logic

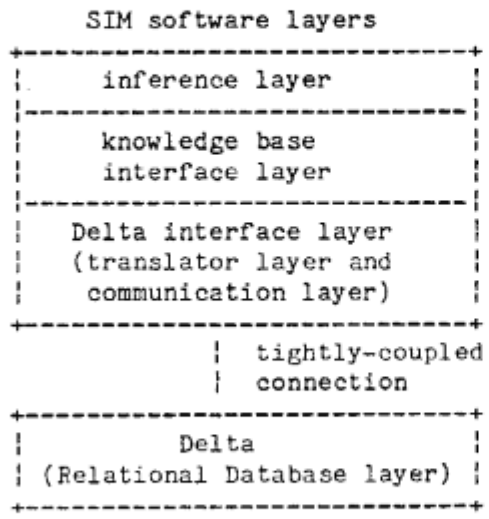

Fig. 5.2 Software Layers for a Knowledge Base Experiment

programming language and relational database. By providing a software layer over RDBMS, we can simulate an experimental interface between knowledge base and inference portion. Thus we can vary the interface levels and will investigate an appropriate interface for a future knowledge base machine (Fig.5.2).

6. Conclusion

We have described the functionally-distributed architecture of Delta, its hardware and software configuration. We have presented a detailed processing flow by taking up a sample transaction. The standpoint of Delta in the logic programming environment and a future research plan for a knowledge base machine is finally described. Rough estimation of Delta's performance is given in [Shibayama 84]. More accurate performance estimation, with the measurements on the actual machine, is the next evaluation step. Knowledge base mechanism research will be focused and conducted at the end of the initial-stage project to the intermediate stage.

7. Acknowledgment

The authors show their appreciation to the Toshiba and Hitachi researchers and engineers for the implementation decisions and designs, and the efforts for the urgent work.

[REFERENCES]

[Sakai 84] Sakai, H., Iwata, K., et al., "Design and Implementation of a Relational Database Engine", to appear in Proc. 2nd FGCS Conference, November, 1984.

[Shibayama 84] Shibayama, S., Kakuta, T., Miyazaki, N., Yokota, H., Murakami, K., "A Relational Database Machine with Large Semiconductor Disk and Hardware Relational Algebra Processor", ICOT Technical Report TR-055 and also to appear in New Generation Computing, Vol.2, No.2, March, 1984.

[Todd 78] Todd, S., "Algorithm and Hardware for a Merge Sort Using Multiple Processors", IBM Journal of Res. and Develop., 22, 1978.

[Yokota 83] Yokota, H., et al., "An Investigation for Building Knowledge Base Machines", ICOT Technical Memorandum TM-0019, 1983. (in Japanese)

[Yokota 84] Yokota, H., Kunifuji, S., Kakuta, T., Miyazaki, N., Shibayama,

S., Murakami, K., "An Enhanced Inference Mechanism for Generating Relational Algebra Queries", Proc. 3rd ACM SIGACT-SIGMOD symposium on Principles of Database Systems, pp. 229 - 238, April, 1984.