NPS52-85-001

NAVAL POSTGRADUATE SCHOOL

Monterey, California

NEW DIRECTIONS IN DATABASE-SYSTEMS RESEARCH AND DEVELOPMENT

Steven A. Demurjian and David K. Hsiao

February 1985

Prepared for:

Chief of Naval Research
Arlington, VA 22217

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Commodore R. H. Shumaker
Superintendent

D. A. Schrady
Provost

The work reported herein was supported by Contract from DoD STARS Program and from the Office of Naval Research.

This report was prepared by:

[illegible]
DAVID K. HSIAO
Professor of Computer Science

Reviewed by:

[illegible]
BRUCE J. MACLENNAN
Acting Chairman
Department of Computer Science

Released by:

[illegible]
KNEALE T. MARSHALL
Dean of Information and
Policy Science

unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		**READ INSTRUCTIONS BEFORE COMPLETING FORM**
1. REPORT NUMBER NPS52-85-001	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) New Directions In Database-Systems Research and Development		5. TYPE OF REPORT & PERIOD COVERED
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Steven A. Demurjian and David K. Hsiao		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61153N; RR014-08-01 N0001485WR24046
11. CONTROLLING OFFICE NAME AND ADDRESS Chief of Naval Research Arlington, Virginia 22217		12. REPORT DATE February 1985
		13. NUMBER OF PAGES 29
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report)
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report)		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)		

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

In this paper, three new directions in database-systems research and development are indicated. One new direction is the emergence of the multi-lingual database systems where a single database system can execute many transactions written respectively in different data languages and support many databases structured correspondingly in various data models. Thus, a multi-lingual database system allows the old transactions and existing databases to be migrated to the new system, the user to explore the strong features of the various data languages and data models in the same system, the hardware upgrade to be focused

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

on a single system instead of a heterogeneous collection of database systems, and the database application to cover wider types of transactions and interactions in the same environment.

One other new direction is the emphasis of the multi-backend database systems where the database system is configured with a number of microprocessor-based processing units and their disk subsystems. These processing units and disk subsystems are called database backends. The unique characteristics of the backends are that the number of the backends is variable, the system software in all of the backends is identical, and the multiplicity of the backends is proportional to the performance and capacity of the system. Thus, for the first time, a multi-backend database system enables the user to relate the amount of hardware used (i.e., the number of the backends) to the degree of performance gain and capacity growth of the system.

The third new direction is the possibility of the multi-host database systems where a single database system can communicate with a variable number and heterogeneous collection of mainframes in several different data languages and allow the mainframes to share the common database store and access.

This paper attempts to articulate the background, benefits, requirements and architectures of these new types of database system, namely, the multi-lingual the multi-backend, and the multi-host database systems.

NEW DIRECTIONS IN DATABASE-SYSTEMS RESEARCH AND DEVELOPMENT *

Steven A. Demurjian and David K. Hsiao

Department of Computer Science
Naval Postgraduate School
Monterey, California 93943
U. S. A.

ABSTRACT

In this paper, three new directions in database-systems research and development are indicated. One new direction is the emergence of the *multi-lingual database systems* where a single database system can execute many transactions written respectively in different data languages and support many databases structured correspondingly in various data models. Thus, a multi-lingual database system allows the old transactions and existing databases to be migrated to the new system, the user to explore the strong features of the various data languages and data models in the same system, the hardware upgrade to be focused on a single system instead of a heterogeneous collection of database systems, and the database application to cover wider types of transactions and interactions in the same environment.

One other new direction is the emphasis of the *multi-backend database systems* where the database system is configured with a number of microprocessor-based processing units and their disk subsystems. These processing units and disk subsystems are called *database backends*. The unique characteristics of the backends are that the number of the backends is variable, the system software in all of the backends is identical, and the multiplicity of the backends is proportional to the performance and capacity of the system. Thus, for the first time, a multi-backend database system enables the user to relate the amount of hardware used (i.e., the number of the backends) to the degree of performance gain and capacity growth of the system.

* The work reported herein is supported by grants from the Department of Defense STARS Program and from the Office of Naval Research and conducted at the Laboratory for Database Systems Research, Naval Postgraduate School, Monterey, CA 93943.

QA
76.9
S88
D45
1985

The third new direction is the possibility of the *multi-host database systems* where a single database system can communicate with a variable number and heterogeneous collection of mainframes in several different data languages and allow the mainframes to share the common database store and access.

This paper attempts to articulate the background, benefits, requirements and architectures of these new types of database systems, namely, the multi-lingual, the multi-backend, and the multi-host database systems.

TABLE OF CONTENTS

1. INTRODUCTION

In this paper, three new directions in database-systems research and development are indicated. One new direction is the emergence of the *multi-lingual database systems* where a single database system can execute many transactions written respectively in different data languages and support many databases structured correspondingly in various data models. For example, a multi-lingual database system can run DL/I transactions on IMS databases, CODASYL-DML transactions on network databases and SQL transactions on relational databases, where the system appears to the user like a heterogeneous collection of database systems. Thus, a multi-lingual database system allows the old transactions and existing databases to be migrated to the new environment, the experienced user to continue to utilize certain favorite features of existing data languages and data models, the new user to explore the strong features of the various data languages and data models, the hardware upgrade to be focused on a single system instead of a heterogeneous collection of database systems, and the database application to cover wider types of transactions and different modes of interactions.

One other new direction is the emphasis of the *multi-backend database systems* where the database system is configured with a number of microprocessor-based processing units and their disk subsystems. These processing·units and disk subsystems are known as *database backends.* The unique characteristics of the system are that the number of the backends is variable, the system software in all of the backends is identical, and the multiplicity of the backends is proportional to the performance and capacity of the system. For example, by doubling the number of backends of the original multi-backend database system, the response time of the transactions in the new system for the database can be reduced to nearly one half of the response time of the same transactions for the same database running in the original system. Similarly, as the database grows in the original system, the response set for a transaction may also grow. By doubling the number of backends of the original multi-backend database system, the response time of the transaction in the new system can be held nearly constant, despite an increase of twice as much responses for the same transaction. Thus, for the first time, a multi-backend database system enables the user to relate the amount of hardware used (i.e., the multiplicity of the backends) to the degree of response-time reductions and performance gains of the system.

The third new direction is the possibility of the *multi-host database systems,* where a single database system can interface with a large, variable number of heterogeneous computers which do not have database-systems software, but, nevertheless, require diverse and cost-effective database services and support.

The remainder of this paper is organized as follows. In Section 2 we describe the multi-lingual database system, focusing on its practical merits, new functionalities, theoretical issues, and basic structure. In Section 3, we examine the multi-backend database system, focusing on its background, motivation, requirements, and issues. In Section 4, we report on research and development work being conducted on multi-lingual, multi-backend and similar database system. Finally, in Section 5 we conclude this paper by introducing the new direction of the multi-host database systems and by speculating on what will be the database systems of the future.

2. THE MULTI-LINGUAL DATABASE SYSTEM (MLDS)

Data models, data languages and database systems have evolved over the past 20 years. For instance, in the mid-sixties, IBM introduced the Information Management System (IMS), which supports the hierarchical data model and the hierarchical-model-based data language, Data Language I (DL/I). In the early seventies, Sperry Univac introduced the DMS-1100 database system, offering the network data model and the network-model based data language, CODASYL Data Manipulation Language (DML). The evolution continued with IBM's introduction of the SQL/Data System in 1981 which supports the relational model and the relational-model-based data language, Structured English Query Language (SQL). As in the evolution of software-laden database systems, the hardware-assisted database systems followed the same pattern. Thus, the Britton-Lee Corporation introduced the IDM/500 in 1982 and the Teradata Corporation began marketing the DBC/1012 in 1984. Both systems support the relational data model and relational-model-based data languages similar to SQL.

Throughout the past twenty years, the conventional approach to the design and implementation of a database system involved two key decisions. First, a specific data model for the database system is chosen. Second, a corresponding model-based data language is then specified. The result of this traditional approach to the database system development is a homogeneous database system where the user sees and uses the database system with a specific data model and its model-based data language. The accepted practice for the database-systems design and implementation mandates that a database system must be restricted to a single data model and a specific model-based data language.

Why should a database system be restricted to a single data model and a specific model-based data language? Let us review the evolution of operating systems before answering this question. The early operating systems, like the present database systems, individually supported a specific set of data structures and a single programming language which defines and manipulates the structured data. For example, the Fortran Monitor System of the late fifties supported an operating system environment for a single programming language (i.e., Fortran) and its corresponding data structures (e.g., Fortran arrays and variables). As operating systems evolved through the sixties and seventies and into the eighties, the same operating environment supported a variety of data structures and their programming languages. For example, the Unix operating system supports traditional programming languages, such as C, Pascal, and Fortran, list-processing programming languages, such as Lisp, and logic programming languages, such as Prolog. Each of these programming languages has its own set of data structures. All programs written in the aforementioned languages and data structures can be run in the same operating system which is also responsible for managing all of the physical resources shared by the running programs and their data structures.

Given this characterization of the operating-systems evolution, we can draw an interesting analogy between operating systems and database systems. The concepts of the modern operating systems, programming languages, data structures, and shared resources are analogous to the concepts of modern database systems, data languages, data models and shared databases. Since a modern operating system executes and supports the user's programs in different programming languages and data structures, a modern database system should also execute and support the user's transactions in different data languages and data models. Since a modern operating system provides access to and management of a common set of resources for the running programs, a modern database system should also provide access to and management of a large collection of shared databases for the running transactions. Finally, since a modern operating system provides many modes of access, such as interactive programming and batch processing, a modern database system should also provide many modes of access, such as ad-hoc queries and transaction processing. With this analogy, we respond to the question in the previous paragraph that a modern database system should be able to support multiple data models and their different data languages and provide various modes of access to the databases. Such a modern database system is termed the *multi-lingual database system (MLDS).*

2.1. Issues and Merits of a Multi-Lingual Database System

The issues and merits of a multi-lingual database system fall into three categories. First, by studying the *practical merits* of a MLDS, we are able to demonstrate the concrete and useful features of such a system. Second, by identifying the *new functionalities* inherent in a MLDS, we are able to provide the incentives for the user to move from a conventional database system to the MLDS. Third, by verifying the *theoretical issues* required to support multiple data models and data languages in a MLDS, we may gain a better understanding into the structures of and relationships among different data models and data languages.

2.1.1. Practical Merits

One practical advantage of a multi-lingual database system involves the reusability of database transactions developed on existing database systems. Since MLDS provides an environment for running database transactions written in different data languages, the transactions written in a specific data language on another database system can also be executed in MLDS. There is no need to translate a transaction written in one data language to another data language in order to run the transaction in the other database system. For example, had we wanted to run a transaction (written in DL/I and running on IMS) in SQL/DS, we would have to translate the transaction from DL/I to SQL, since SQL/DS is a relational system and does not run DL/I transactions. However, in a multi-lingual database system, although both SQL and DL/I are supported, there is no need of any translation from DL/I to SQL. Nor is there a need of translation from SQL to DL/I. A MLDS can execute transactions written in either DL/I or SQL. Thus, a MLDS provides an environment in which "old" transactions never die and "new" transactions can continue to be written in the same (old) data languages.

The second practical advantage of a multi-lingual database system lies in the economy and effectiveness of hardware upgrade. As for any database system there comes a time when a hardware upgrade is required due to technology advancement or system demand. The upgrade of a MLDS will benefit all of the user transactions whether the transactions are written, for instance, in SQL, DL/I, or CODASYL-DML. In the conventional environment where there are separate database systems for separate data languages, all of the database systems would need to be upgraded. For our example, the conventional upgrade involves the hardware of SQL/DS, of IMS and of DMS-1100, resulting in greater expense and effort.

2.1.2. New Functionalities

One new functionality of a multi-lingual database system is to allow the new users to explore the strong points of different data models and to utilize desirable features of different data languages for their applications. This is because a MLDS can be used to support databases structured in any of the well-known data models, such as relational, hierarchical, or network, and to execute transactions written in any of the well-known data languages, such as SQL, DL/I, or CODASYL-DML.

The other new functionality of a MLDS is the availability of its native data model and data language. The native data model of the MLDS is called the *kernel data model (KDM),* and the native data language the *kernel data language (KDL).* The term "kernel" is meant to be "central" or "core" or "essential". The difference between a conventional data model and the kernel data model is that all of the databases structured in a conventional data model can be transformed into equivalent databases structured in the kernel model. Further, all of the conventional data languages can be translated into the kernel data language. It is important to note that the KDM (KDL) as a data model (language) is at a high level like other data models (languages), such as the relational data model (SQL data language), the hierarchical data model (DL/I data language) and the network data model (CODASYL-DML data language). Thus, there is no reason why the users should not also explore the strong points of KDM and the desirable features of KDL for their applications.

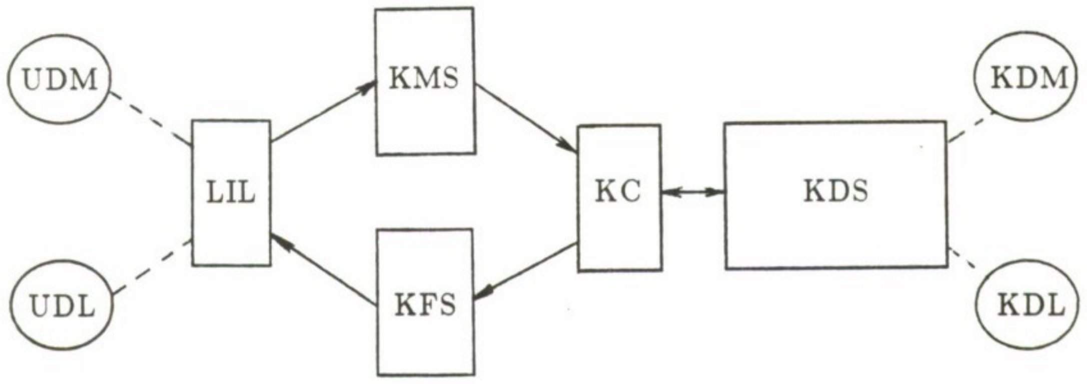
2.1.3. Theoretical Issues

In searching for a kernel data model and kernel data language with a high-level structure, which will support different data models and data languages, we are examining the transformations of various data models into the kernel data model and the translations of various data languages to the kernel data language. The mapping process from a given data model to KDM is called *data-model transformation.* The mapping process from a given data language to KDL is called *data-language translation.* To design a multi-lingual database system, the data-model transformations and data-language translations must be specified. By specifying the various data-model transformations, e. g., from the relational model to the KDM, from the hierarchical model to KDM, and from the network model to the KDM, we may also examine the transformation process to determine the commonalities and differences of the different transformations. Similarly, by providing various data-language translations, e. g., from SQL to KDL, from DL/I to KDL, and from CODASYL-DML to KDL, we may also study the translation process to identify the common and different translation techniques. Finally, once all of the data-model transformations and data-language translations have been specified, we can examine the complexity of the transformation and translation processes.

2.2. The Organization of a MLDS

The system structure of a multi-lingual database system is shown in Figure 1. Users issue transactions through the language interface layer (LIL) using a chosen data model (UDM) and written in a corresponding model-based data language (UDL). LIL then routes the user transactions to the kernel mapping system (KMS). KMS has two tasks. First, if the user specifies that a new database is to be created, KMS transforms the UDM database definition to a kernel-data-model-based (KDM) database definition. The KDM data definition is then sent to the kernel controller (KC). KC sends the KDM database definition to the kernel database system (KDS). Upon completion, KDS notifies KC, which in turn, notifies the user that the database definition has been processed and that the loading of the database may commence.

The second task of KMS is to handle UDL transactions. In this situation, the KMS translates the UDL transaction to a kernel-data-language (KDL) transaction. KMS then sends the KDL transaction to KC, which in turn, sends the KDL transaction to KDS for execution. Upon completion, KDS sends the results in KDM form back to KC. KC forwards these results to the kernel formatting system (KFS) for transforming them from the KDM form to the UDM form. After the data is transformed, KFS returns the results, i.e., the response set, to the user via LIL.



UDM	: User Data Model
UDL	: User Data Language
LIL	: Language Interface Layer
KMS	: Kernel Mapping System
KC	: Kernel Controller
KFS	: Kernel Formatting System
KDM	: Kernel Data Model
KDL	: Kernel Data Language
KDS	: Kernel Database System

Figure 1. The Multi-Lingual Database System (MLDS)

There is one final note of importance on the general system structure. Four of the five components of the multi-lingual database system, namely, LIL, KMS, KC, and KFS, are referred to as a *language interface,* and are duplicated for each chosen data language. For example, there will be a set of LIL, KMS, KC, and KFS for a relational/SQL language interface, a separate set of these four components for a hierarchical/DL/I language interface, and a third set of components for the network/CODASYL-DML language interface. KDS, on the other hand, is a single and major component that is accessed and shared by all of the various language interfaces.

3. THE MULTI-BACKEND DATABASE SYSTEM (MBDS)

In Section 2, the progression of database-systems research and development in terms of their support of single or multiple data models and data languages has been followed. The progression of database-systems research and development can also be followed in view of their architectural configurations. A taxonomy of the architectural configurations of database systems can be found in the preface of [Hsia83]. Similar and simplified taxonomies have appeared in [Cham78, Hsia80]. The *conventional approach to database managcment* has the database-system software running as an applications program on a mainframe computer system. In this case, the database system must share the use and control of the resources with all of the other applications of the mainframe system. As the workload of a conventional database system increases, the performance of the database system degrades.

The solution to the problems of performance degradation and resources sharing and control is to offload the database-system software from the mainframe computer to a separate, dedicated computer with its own disk system. This approach, called the *software single-backend approach,* was adopted by Bell Laboratories in their work on XDMS [Cana74]. As quoted below, the main goals of XDMS were to:

> (1) obtain a cost saving and a performance gain through specialization of the database operations on a dedicated backend processor,
>
> (2) allow the use of shared databases [by different mainframe computers, now called *hosts*],
>
> (3) provided centralized [i. e., physical] protection of the databases, and
>
> (4) reduce the complexity when developing software for a stand-alone and new machine.

Software single-backend systems can achieve goals 2, 3, and 4, but have had difficulty in meeting goal 1 entirely. Although single backends may be cost-effective, these systems suffer from performance problems; in fact, they suffer from the same performance problems of the database systems running on the mainframes. As the use of a software single-backend database system increases, the single backend can no longer maintain the desired performance which had been gained by offloading the database software from the mainframe and by utilizing the dedicated hardware. Like the hardware upgrade of mainframe computers, the conventional approach to the hardware upgrade of software single-backend systems is to use the next more powerful backends. Unfortunately, such an upgrade does not yield precise, direct and proportional performance gains with respect to cost differentials. There is, however, an "unconventional" approach to the hardware upgrade process.

To overcome the performance problems of the software single-backend approach, the use of multiple backends for the database management operations, as an unconventional approach, is being considered. This approach, known as the *software multiple-backend approach,* may overcome the performance failings of the single-software backend approach. We will refer to a system that uses the software multiple-backend approach as a *multi-backend database system (MBDS).*

MBDS attempts to provide performance gains through specialization of the database operations on dedicated, multiple backends. Unlike XDMS, MBDS does not restrict itself to a single backend. Instead, MBDS utilizes multiple backends connected in a parallel fashion in order to achieve performance gains and capacity growth. These backends have identical and replicated software and their own disk systems. In MBDS, there is a backend controller (i. e., master) which is responsible for supervising the execution of database transactions and for interfacing with the hosts and users. The backends (slaves) perform the database operations with the database stored on the disk systems of the backends. The controller and backends are connected by a communications bus. Users access the system either by way of the hosts or through the controller directly. Figure 2 depicts the basic architectural configuration of MBDS.

The two goals of a multi-backend database system are of course to overcome the performance problems of single-backend database systems. First, by increasing the number of backends, while the size of the database and the size of the responses to the transactions remain constant, MBDS is to produce a reciprocal decrease in the response times of the user transactions. Second, by increasing the number of backends proportionally to the increase of transaction responses, MBDS is to produce invariant response times for the user transactions. The first goal allows the multiplicity of the backends of MBDS to be directly related to the *performance gains* of MBDS in terms of the *response-time reduction.* The second goal enables the multiplicity of the backends of MBDS to be directly related to the *capacity growth* of MBDS in terms of *response-time invariance.*

How can a multi-backend database system be designed and implemented to meet the two aforementioned goals? In the following subsections the necessary and sufficient features of a "good" MBDS are given. These are the *design requirements,* which underscore the major characteristics of MBDS. The characteristics that a MBDS must have in order to satisfy the major design requirements are also given. These are termed

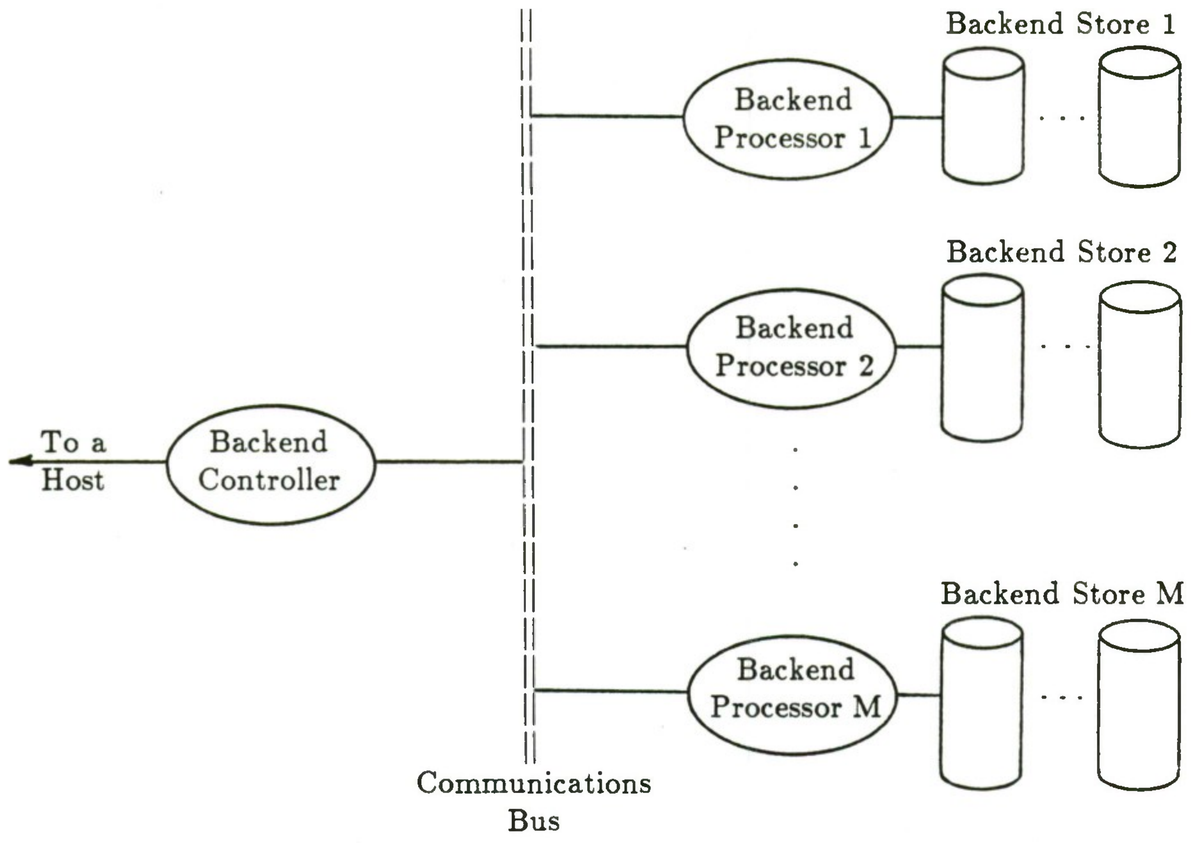


Figure 2. The Multi-Backend Database System (MBDS)

design issues.

3.1. Design Requirements

There are three requirements that underscore a multi-backend database system. One requirement states that MBDS be *expandable,* in order to support the addition of backends for performance enhancement and capacity growth. This expansion must require no modification to the existing database software, no new programming necessary for the expansion, no modifications to the hardware and no major disruption of system activity when additional backends are being incorporated into the system.

The second requirement mandates that both the hardware and software are *generic.* The hardware of the backends should be typical and readily available (i. e., off-the-shelf) and can be added to the system with minimal interruption of the system activity. This creates a system that permits a smooth and ready expansion without relying on costly, atypical, special-purpose hardware and without noticeable system interruption. The backend software should be designed so that a new backend can be integrated into the system by simply replicating the database system software of another backend into the new backend.

The third requirement suggests that, for storage, a database is *evenly distributed* across the disk systems of the backends. Thus, when a transaction is being processed, a backend works on its own portion of the database in parallel with other backends working on their own portions of the same database. By exploiting the parallelism of the backends and by distributing a database evenly for storage, the system should gain in performance.

3.2. Design Issues

There are several issues which must be resolved in order to meet the design requirements of a multi-backend database system. In particular, these design issues involve the specification of the characteristics of the backend controller, the communications bus, the backends, and the database.

The first issue specifies the features of the backend controller. The overall design goal of a backend controller should focus on minimizing the work done by the controller. (See Figure 2 again.) The controller receives a user transaction either from a host or through a terminal and broadcasts the transaction to all of the backends for execution. The controller also collects all of the results produced by the backends for the user transaction and routes the results to the host or to the terminal. As such, the controller becomes a prime candidate for the bottleneck of the system. By minimizing the work of the controller, and by offloading **all** of the database management operations to the backends, the controller may reduce its possibility of becoming the system bottleneck. Overall, the functions of the controller are reduced to the pre-processing of the user transactions, the post-processing of the transaction results, the sending and receiving of data from the backends and the hosts, and the arbitration of data insertion into the database.

The second design issue is the communications bus between the controller and the backends. Consider two extreme choices: one where a broadcast bus is shared by the controller and backends or another where a point-to-point high-speed bus between the controller and each backend is utilized. While the high-speed bus may offer a higher communications rate, the broadcast bus is a cost-effective solution, since data-intensive transfers are between the backends and their disk systems and are not between the backends and the controller (i. e., not on the broadcast bus). The choice of the broadcast bus as a cost-efficient solution for both backend communication and backend addition may be warranted.

The third class of issues involves the backends of the system. The backends of the system must have identical software to allow replication of the software on a new backend. Additionally, the backends must have complete software to perform all of the database management functions. These functions include directory management, concurrency control, record processing, and communications. The directory management function is responsible for managing indices, calculating record clusters, allocating the secondary-storage addresses for record insertion, maintaining the secondary-storage tables of indices, cluster numbers, and addresses, processing transactions against the directory tables, and providing record addresses for subsequent database access operations. The concurrency control function oversees various accesses to the directory tables and the user data and facilitates the concurrent execution of transactions. The record processing function is used to stage the user data from the secondary storage to the primary memory, to process the staged data, to store data onto the secondary storage, and to return the responses to the controller. Finally, there are communication functions in each backend to control communications among backends and between the backend and the controller. It is necessary to minimize the communications among backends, in order to reduce the communications traffic among them.

The fourth and final design issue is concerned with the database. In a multi-backend database system, a database must be placed on the secondary storage in such a way so that all of the subsequent accesses to the database will result in block-parallel-and-record-serial operations at the individual backends. In other words, all of the backends are accessing, in parallel, the secondary-storage blocks of the same database in their respective disk systems, although the records in the blocks which may satisfy the same transaction or different transactions are being accessed by the backends serially. Thus, the issue really focuses on how to ensure an even distribution of the user database across the disk systems of the backends. Such a distribution is referred to as *data placement,* and requires an algorithm for specification and implementation. To achieve an even distribution of data, there must be a processor in the multi-backend database system that is responsible for overseeing the record-insertion process. The controller has an overview of the entire system, and is the logical choice for arbitrating the record insertion process, i. e., controlling the data placement.

4. RESEARCH AND DEVELOPMENT IN MULTI-LINGUAL AND MULTI-BACKEND DATABASE SYSTEMS

In this section we review and characterize the research and development efforts being conducted on both multi-lingual and multi-backend database systems. We organize the section into two major subsections, and examine each of the two classes of database systems, namely, multi-lingual and multi-backend classes of database systems. Since we are also actively pursuing research in both of these two classes of database systems at the Laboratory for Database Systems Research, Naval Postgraduate School, each of these subsections describes our approach and other approaches to the design and implementation of the corresponding class of database systems.

4.1. On the Multi-Lingual Database System

4.1.1. Our Effort

As a prerequisite for examining our experience with a MBDS, we first explore the two goals of the mapping process, i. e., data-model transformation and data-language translation. In data-model transformation, we must be sure that the data semantics are preserved. When converting a database (modeled in, for example, one of the three major models) to an kernel database, we must ensure that a complete and consistent database has been created. In data-language translation, we must guarantee the operational equivalence of the translated transaction. Thus, when translating a source transaction (written in, for example, one of the three major data languages) to a target transaction written in the kernel data language, we must ensure that the access of the stored database by the target transaction results in the correct action on the database as required by the source transaction. To us, the key decision in the development of a multi-lingual database system is therefore the choice of a kernel data model and kernel data language. In our effort, we experiment with the attribute-based data model proposed by [Hsia70], extended by [Wong71], and studied by [Roth74] as the kernel data model for a MLDS. The attribute-based data language (ABDL) defined in [Bane77] is therefore chosen as the kernel data language. The main question is whether the attribute-based data model and data language are capable of supporting the required data-model transformations and data-language translations. Is it easy to transform a relational, hierarchical or network database to an attribute-based database with the data semantics being preserved? Can SQL, DL/I and CODASYL-DML constructs be translated easily to ABDL constructs with the guarantee of operational equivalence?

The series of papers [Bane78c, Bane80, Bane78a] have shown how the relational, hierarchical, and network data can be transformed to attribute-based data and also presented some preliminary work on the corresponding data-language translations. More recently, the work of [Macy84, Roll84] provides a complete set of algorithms for the data-language translation from SQL to ABDL, and the work of [Weis84] provides a complete set of algorithms for the data-language translation from DL/I to ABDL. Currently, the algorithms for translating from CODASYL-DML to ABDL are under investigation. Software development efforts for the language interface, i. e., one set of LIL, KMS, KFS, and KC for the relational interface and another set for the hierarchical interface, are being pursued.

4.1.2. Other Efforts

In the data-model transformation and data-language translation areas, there are other efforts. These efforts have emphasized different approaches to a MLDS. In one approach, the goal is to examine the capability of an existing database system in supporting another data model and language on the existing system. The work of [Katz82] supports the network data model and CODASYL-DML data language on a relational system. Furthermore, the support of the relational data model and data language on a network system and the support of the network data model and data language on a relational database system have been examined in [Lars83]. Each of these examinations is essentially restricted to the mapping from one data model and data language to another data model and data language. We define this approach as the *one-to-one mapping approach.*

The other effort in data-model transformation and data-language translation focuses on communicating with a heterogeneous collection of database systems via a local-area network. In this effort, a global data model and global data language is defined. By using a global data model and global data language, the user is able to obtain uniform access to a heterogeneous collection of database systems based on different data models and data languages [Glig84]. The CCA Multibase System [Rose82], UCLA DBMS [Card80], and NBS XNDM [Kimb81] are examples of database systems each of which maps a single global data model and global data language to a collection of data models and data languages, e.g., the *one-to-many mapping approach.*

It is interesting to note that the MLDS described in Section 4.1.1 maps respectively many different data models and data languages to the kernel data model and kernel data language which is the *many-to-one mapping approach.* The one-to-many mapping approach is the inverse of the many-to-one mapping approach and the one-to-one mapping approach is a special case of the more general case of the many-to-one mapping approach.

4.2. On the Multi-Backend Database System

4.2.1. Our Research

Our research on the multi-backend database system had its origins at the Ohio State University in 1980 and has been based at the Naval Postgraduate School since 1982. The original design and analysis of MBDS can be found in [Hsia81a, Hsia81b]. The implementation and new design efforts are documented in [Kerr82, He82, Boyn83b, Demu84]. A review of the message-passing structure of MBDS is given in [Boyn83a]. MBDS attempts to meet all of the goals and requirements outlined in Section 3. It has been implemented on a VAX-11/780 (VMS OS) as the controller and two PDP-11/44s (RSX-11M OS) and their disk systems as the backends. The disk system of each backend can support one or more disk drives. Communication between computers is accomplished by time-divisioned-multiplex buses, known as parallel communications links (PCLs). When the implementation of MBDS began, neither the microprocessor-based computers nor the broadcast-based communications devices were available. Currently, MBDS is being down-loaded to an initial configuration of five microprocessor-based, Ethernet-connected, and Winchester-drive-supported workstations (4.2 BSD Unix OS) with one of the five being used as the controller and the other four as backends.

As indicated above, MBDS is a message-oriented system. In a message-oriented system, each process corresponds to one system function. These processes, then, communicate among themselves by passing messages. User transactions are also passed among processes as messages. The message paths between processes are fixed for the system. The MBDS processes are created at the start-up time and exist throughout the entire running time of the system. In addition to two message-passing processes, the MBDS controller has three main processes, the request prepartion, the insert information generation, and the post processing. Each MBDS backend also has two message-passing processes. Further, each backend has three different main processes, the directory management, the concurrency control, and the record processing. As research software, MBDS consists of other software for testing and evaluation purposes [Kova83]. However, the basic system is relatively small, with 3,000 lines of C code for the controller and 10,000 lines of C code for each backend.

MBDS provides a centralized database where the database is evenly distributed across the disks of the backends. Only a single copy of the database is stored. The underlying data model for MBDS is the attribute-based data model. The attribute-based data model stores data in files of records. MBDS stores records of a file in clusters. A *cluster* is a group of records such that every record in the cluster satisfies the same set of attribute-value pairs or ranges. Thus, a file is divided into one or more clusters. The distribution of the clusters is accomplished by a cluster-based data placement algorithm.

The cluster-based data placement is arbitrated by the controller and carried out by the backends. New clusters are formed by the backends. When a new record is to be included in its cluster, the controller decides which backend will insert the new record into the cluster. The record insertion into the cluster is accomplished by the chosen backend with the placement of the new record on a block of the backend's secondary storage. Under the direction of the controller, the chosen backend will continue to place additional new records of the same cluster in the block until the block of the secondary storage is filled. When this occurs, the backend notifies the controller that the block is full. The controller then directs another backend to continue the placement of new records of the same cluster. The controller maintains the identification of the backends whose secondary-storage blocks may be used for the insertion of new records into the existing clusters. In a multiple-backend configuration, the cluster-based data placement algorithm achieves a cluster-parallel-and-record-serial operation for any subsequent access to the database [Bane78d].

A preliminary performance measurement of MBDS has recently been conducted. The measurement indicated that when experimenting with a system that has up to two backends, the two performance goals can be nearly met. In other words, the testing found that when going from one to two backends, where both the size of the database and the responses of the transaction remained the same, the response time for the same transactions decreased by an average of forty-seven percent (47%). The testing also found that when going from one to two backends, where both the size of the database and the size of the responses were doubled, the response time for the same transactions stayed basically invariant, with an average increase of only one percent (1%). The performance measurement studies have been documented in [Teka84] and published in [Demu85a, Demu85b].

4.2.2. Other Development

The Teradata Corporation began marketing the DBC/1012 in 1984. Users interact with the system via a program running on a host. Communication between the controller and its backends is accomplished using a network developed by Teradata called the Ynet. Essentially, the Ynet is a sorting and merging network, with a hierarchical structure. The Teradata Corporation indicates that there may be up to 968 backends and controllers on the Ynet. The DBC/1012 also uses a data placement algorithm to evenly distribute the database across all of the backends. Although DBC/1012 utilizes a specialized communications network, it is a database system of the software multiple-backend approach. Both the controller and the backend are microprocessor-based. The DBC/1012 can be configured with one or more controllers managing multiple backends. Each backend supports one large-capacity disk drive of the Winchester type.

Despite the similarity of NPS MBDS and DBC/1012 in their approaches to database management, there is no testing and measurement data on DBC/1012 to report herein, since none has been made available. It should be interesting to see how DBC/1012 measures up to the performance goals of the software multiple-backend approach; i. e., the multiplicity of backends in DBC/1012 should be related to either the performance gains or the capacity growth of DBC/1012, as these goals have been benchmarked on the NPS MBDS.

5. CONCLUDING REMARKS - YET ANOTHER NEW DIRECTION

5.1. The Third New Direction

In the first four sections of the paper, two new directions of database-systems research and development have been articulated, motivated, and outlined. The multi-lingual database system (MLDS) is mainly characterized by its capability of supporting many different and well-known data models and executing transactions written in many different model-based data languages. The multi-backend database system (MBDS) is chiefly evidenced by its expandability for performance gains and capacity growth.

However, the success of MLDS rests on our ability to provide effective algorithms for data-model transformations and data-language translations **and** an efficient database system to manage and access the transformed databases and to interface and execute the translated transactions. The emphasis here is that effective mapping algorithms without an efficient database system will render MLDS impractical. In other words, even if all of the software language interfaces can be devised and written, so that MLDS is very multi-lingual, the slowness and inefficiency of the underlying database system may render the interfaces useless and untimely.

Both the one-to-one and one-to-many mapping approaches are indented for use with existing, conventional database systems, which are either software single-backend or mainframe-based. In either case, it is difficult, if not impossible, to significantly enchance the performance gain and capacity growth of existing, conventional database systems. On the other hand, the many-to-one mapping approach is not restricted to the existing, conventional database systems. Instead, it is intended for use with a new, kernel database system (KDS). If we could build a high-performance and great-capacity kernel database system, then the problem of performance gains and capacity growth would not be an issue.

What we advocate for the future database system is to combine the many-to-one mapping approach of the multi-lingual database system with the multi-backend database system. The former provides the multiple language interfaces and the latter provides the kernel database system. In this combination, both the effectiveness and versatility of the language interfaces of MLDS and the performance gains and capacity growth of MBDS are realized in a future database system. Consequently, the future database system will be both multi-lingual and multi-backend, allowing a third direction of database-systems research and development to be realized by the other two new and seemingly different directions.

5.2. MLDS + MBDS = MHDS

Our current research efforts focus on the third direction, combining the multi-lingual and multi-backend features into one database systems. (Review Sections 4.1.1 and 4.2.1, respectively.) In our design, we intend to use our MBDS as the kernel database system in our MLDS. As mentioned in Section 4.1.2, MBDS is also implemented with the attribute-based data model and data language, which has been the focus of our experimental efforts in MLDS. Given this compatibility, we simply point out two advantages offered by this combination.

First, with MBDS as KDS, the addition of new backends with replicated software to KDS will provide a uniform performance enhancement and capacity growth to the entire MLDS. Thus, the process of hardware upgrades is made easier and simpler for our multi-lingual database system.

A second advantage involves the architectural configuration of our MLDS. Such a configuration is shown in Figure 3. In this environment, the four software components of a language interface, i. e., LIL, KMS, KC, and KFS, are placed on a host. Each host

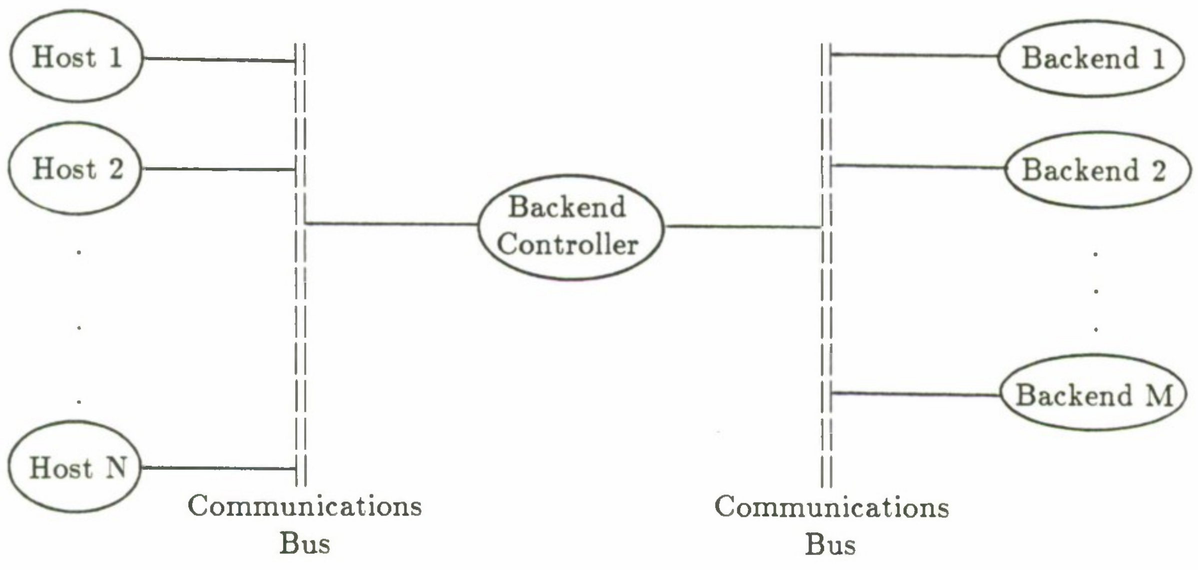


Figure 3. The Multi-Host Database System (MHDS).

may have one or more different language interfaces, e. g., host 1 may have a SQL language interface, host 2 a DL/I language interface as well as a CODASYL-DML language interface, host 3 a SQL language interface and a DL/I language interface, and so on. The distribution of language interfaces on a host-by-host basis allows a large number of hosts to have their favorite data language interfaces and to share the common database store and access. In addition, by offloading the language interface to the hosts and by realizing the kernel database system with a multi-backend database system, we benefit the overall performance of our MLDS.

This third new direction turns the multi-lingual and multi-backend directions into a multi-host one. It is our belief that the future database system will be *multi-host database systems (MHDS)* with multi-lingual and multi-backend capabilities.

REFERENCES

[Bane77] Banerjee, J. and Hsiao, D. K., "DBC Software Requirements for Supporting Relational Databases," The Ohio State University, Tech. Rep. No. OSU-CISRC-TR-77-7, November 1977. This work appeared subsequently in [Bane78c, Bane78d].

[Bane78a] Banerjee, J. and Hsiao, D. K., "A Methodology for Supporting Existing CODASYL Databases with New Database Machines," *Proceedings of National ACM Conference,* 1978.

[Bane78b] Banerjee, J., Buam, R. I. and Hsiao, D. K., "Concepts and Capabilities of a Database Computer," *ACM Transactions on Database Systems,* Vol. 4, No. 1, December 1978.

[Bane78c] Banerjee, J. and Hsiao, D. K., "The Use of a Database Machine for Supporting Relational Databases," *Proc. 5th Workshop on Computer Architecture for Nonnumeric Processing,* August 1978.

[Bane78d] Banerjee, J. and Hsiao, D. K., "Performance Study of a Database Machine in Supporting Relational Databases," *Proceedings of the 4th International Conference on Very Large Data Bases,* September 1978.

[Bane80] Banerjee, J., Hsiao, D. K., and Ng, F., "Database Transformation, Query Translation and Performance Analysis of a Database Computer in Supporting Hierarchical Database Management," *IEEE Transactions on Software Engineering,* Vol. SE-6, No. 1, January 1980.

[Boyn83a] Boyne, R., et al., "A Message-Oriented Implementation of a Multi-Backend Database System (MBDS)," in *Database Machines,* Leilich and Missikoff (eds.), Springer-Verlag, 1983.

[Boyn83b] Boyne, R., et al., "The Implementation of a Multi-Backend Database System (MBDS): Part III - The Message-Oriented Version with Concurrency Control and Secondary-Memory-Based Directory Management," Technical Report, NPS-52-83-003, Naval Postgraduate School, Monterey, California, March 1983.

[Cham78] Champine, G. A., "Four Approaches to a Data Base Computer," *Datamation,* Vol. 24, No. 13, December 1978.

[Cana74] Canaday, R. E., et al., "A Back-end Computer for Data Base Management," *Communications of the ACM,* Vol. 17, No. 10, October 1974.

[Card80] Cardenas, A., and Pirahesh, M. H., "Data Base Communication in a Heterogeneous Data Base Management System Network," *Information Systems,* Vol. 5, No. 1, 1980.

[Demu84] Demurjian, S. A., et al., "The Implementation of a Multi-Backend Database System (MBDS): Part IV - The Revised Concurrency Control and Directory Management Processes and the Revised Definitions of Inter-Process and Inter-Computer Messages" Technical Report, NPS-52-84-005, Naval Postgraduate School, Monterey, California, March 1984.

[Demu85a] Demurjian, S. A., et al., "Performance Evaluation of a Database System in Multiple Backend Configurations," *Proceedings of the 1985 International Workshop on Database Machines,* March 1985.

[Demu85b] Demurjian, S. A. and Hsiao, D. K., "Benchmarking Database Systems in Multiple Backend Configurations," *IEEE Database Engineering Bulletin*, March 1985.

[Glig84] Gligor, V. D., and Luckenbaugh, G. L., "Interconnecting Heterogeneous Database Management Systems," *IEEE COMPUTER,* Vol. 17, No. 1, January 1984.

[He82] He, X., et al., "The Implementation of a Multi-Backend Database System (MBDS): Part II - The First Prototype MBDS and the Software Engineering Experience," Technical Report, NPS-52-82-008, Naval Postgraduate School, Monterey, California, July 1982; also appeared in *Advanced Database Machine Architecture,* Hsiao (ed.), Prentice Hall, 1983.

[Hsia70] Hsiao, D. K., and Harary, F., "A Formal System for Information Retrieval from Files," *Communications of the ACM,* Vol. 13, No. 2, February 1970; Corrigenda, Vol 13., No. 4, April 1970.

[Hsia80] Hsiao, D. K., "Data Base Computers," in *Advances in Computers,* Yovits (ed.), Vol. 19, Academic Press, 1980.

[Hsia81a] Hsiao, D. K. and Menon, M.J., "Design and Analysis of a Multi-Backend Database System for Performance Improvement, Functionality Expansion and Capacity Growth (Part I)," Technical Report, OSU-CISRC-TR-81-7, The Ohio State

University, Columbus, Ohio, July 1981.

[Hsia81b] Hsiao, D. K. and Menon, M.J., "Design and Analysis of a Multi-Backend Database System for performance Improvement, Functionality Expansion and Capacity Growth (Part II)," Technical Report, OSU-CISRC-TR-81-8, The Ohio State University, Columbus, Ohio, August 1981.

[Hsia83] Hsiao, D. K., (ed.), *Advanced Database Machine Architectures,* Prentice-Hall, 1983.

[Katz82] Katz, R. H., and Wong, E., "Decompiling CODASYL DML into Relational Queries," *ACM Transactions on Database Systems,* Vol. 7, No. 1, March 1982.

[Kerr82] Kerr, D.S., et al., "The Implementation of a Multi-Backend Database System (MBDS): Part I - Software Engineering Strategies and Efforts Towards a Prototype MBDS," Technical Report, OSU-CISRC-TR-82-1, The Ohio State University, Columbus, Ohio, January 1982; also appeared in *Advanced Database Machine Architecture,* Hsiao (ed.), Prentice Hall, 1983.

[Kimb81] Kimbleton, S. R., and Wang, P., "Application and Protocols," in *Distributed Systems: Architecture and Implementation,* Lecture Notes in Computer Science, Paul Lampson and Siegert, eds., Vol. 105, Springer Verlag, New York, 1981.

[Kova83] Kovalchik, J. G., "Performance Evaluation Tools for a Multi-Backend Database System," Master's Thesis, Naval Postgraduate School, Monterey, California, December 1983

[Lars83] Larson, J. A., "Bridging the Gap Between Network and Relational Database Management Systems," *IEEE COMPUTER,* Vol. 16, No. 9, September 1983.

[Macy84] Macy, G., "Design and Analysis of an SQL Interface for a Multi-Backend Database System," Master's Thesis, Naval Postgraduate School, Monterey, California, March 1984.

[Roll84] Rollins, R., "Design and Analysis of a Complete Relational Interface for a Multi-Backend Database System," Master's Thesis, Naval Postgraduate School, Monterey, California, June 1984.

[Rose82] Rosenberg, R. L., and Landers, T., "An Overview of MULTIBASE," *Distributed Data Bases,* H.-J. Schneider, ed., North-Holland Publishing Company, 1982.

[Roth74] Rothnie, J. B. Jr., "Attribute Based File Organization in a Paged Memory Environment," *Communications of the ACM,* Vol. 17, No. 2, February 1974.

[Teka84] Tekampe, R. C., and Watson, R. J., "Internal and External Performance Measurement Methodologies for Database Systems," Master's Thesis, Naval Postgraduate School, Monterey, California, June 1984.

INITIAL DISTRIBUTION LIST

Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
Dudley Knox Library Code 0142 Naval Postgraduate School Monterey, CA 93943	2
Office of Research Administration Code 012A Naval Postgraduate School Monterey, CA 93943	1
Chairman, Code 52M1 Department of Computer Science Naval Postgraduate School Monterey, CA 93943	20
David K. Hsiao Professor Department of Computer Science Naval Postgraduate School Monterey, CA 93943	150

[Weis84] Weishar, D., "Design and Analysis of a Complete Hierarchical Interface for a Multi-Backend Database System," Master's Thesis, Naval Postgraduate School, Monterey, California, June 1984.

[Wong71] Wong, E., and Chiang, T. C., "Canonical Structure in Attribute Based File Organization," *Communications of the ACM,* Vol. 14, No. 9, September 1971.