Database Research at HP Labs

Marie-Anne Neimat and Ming-Chien Shan
Hewlett-Packard Laboratories
1501 Page Mill Road
Palo Alto, CA 94304

1 Introduction

Database research at HP Labs[1] has focused in the last few years on the productive application development and use of database systems. The approach centered on developing a rich database model for expressing the semantics and behavior of information. The *Iris* object-oriented database management system is the result of that effort. Iris combines the advantages of object-oriented concepts with database functionality.

The advent of open systems and fast networks has fostered the belief that information is available at one's fingertips. Yet these advances have pointed to heterogeneity as the most serious challenge to programmer and knowledge worker productivity. Heterogeneous database systems must resolve the semantic discrepancies between different databases, handle the conversions between multiple data models, schemas and formats, etc., while preserving the autonomy of the underlying systems. Preserving the autonomy of the component databases is essential because they are supplied by different vendors or are large systems that took many years to develop, which makes modifying them either impossible or impractical.

There is another class of data management systems that database researchers have not attempted to integrate, and yet these systems manage the bulk of data currently stored on computer systems. They are the specialized database systems such as spatial, geographical and CAD/CAM DBMSs. These customized systems were developed and optimized to meet the performance requirements of their respective applications. A new challenge is to interoperate these heterogeneous database systems while preserving their performance requirements. In integrating this class of systems, preserving their autonomy is no longer the primary requirement, but rather preserving the performance of the integrated system. Exploiting parallel and distributed execution is essential for meeting/exceeding the performance requirements for the integrated systems. We view autonomy and performance as two requirements of heterogenous databases at two ends of a spectrum where one must be traded to obtain the other.

This report describes the highlights of the Iris work and [illegible] research efforts in traditional heterogeneous database integration as well as the integration of customized data managers. The *Pegasus* project addresses the integration of heterogeneous databases while preserving their requirements for autonomy. The *Papyrus* project addresses the integration of customized data managers while preserving their requirements for performance. The *Perseus* project is focused on the presentation and browsing of heterogeneous data.

[1] This report describes the research of the Database Technology Department of HP Labs, Palo Alto.

2 Iris

The goal of the Iris project was to develop a rich database model for expressing the semantics and behavior of information. It was believed that such a model would improve the productivity of database programmers and accommodate the needs of newer database applications such as office systems, engineering design, computer integrated manufacturing and knowledge-based systems. It was decided early on that in order for such a model to become successful, its implementation could not compromise on the functionality of already existing database management systems. Thus, powerful non-procedural query facilities, support for persistent data, multi-user access and updates, and resilience to software and hardware crashes were all requirements.

These efforts resulted in the Iris model and in a series of prototypes that were developed over a period of four years. Iris is an object-oriented database management system. The first Iris prototype was developed in Lisp and demonstrated that a rule-based approach to query processing was viable. Subsequent prototypes were written in C and represented improvements and enhancements to the earlier prototypes.

The following sections describe various aspects of Iris. A general description of the Iris model and system may be found in:

[FISH89] D. H. Fishman, J. Annevelink, E. Chow, T. Connors, J. W. Davis, W. Hasan, C. G. Hoch, W. Kent, S. Leichner, P. Lyngbaek, B. Mahbod, M.-A. Neimat, T. Risch, M.-C. Shan, W. K. Wilkinson. Overview of the Iris DBMS. In *Object-Oriented Concepts, Databases, and Applications*, (ed.) W. Kim, F. H. Lochovsky, Addison-Wesley Publishing Company, 1989. Also HPL-SAL-89-15, January 1989.

2.1 Data Model

The Iris object model supports abstract types, inheritance and encapsulation. Three basic constructs are at the heart of the Iris model. They are *objects*, *types* and *functions*. Objects are used to represent entities from the domain being modeled. Each object has a unique identifier. Objects can have multiple types and types are organized in an inheritance graph. Functions are defined over types and are used to define attributes of objects and relationships among objects belonging to those types. *Stored functions* maintain their extensions in persistent tables while *derived functions* are defined in the Iris query language and are computed by executing their definition. *Procedures* are functions with side-effects.

Iris provides extensibility not only through abstract data types, but also through *foreign functions*. Foreign functions are defined in a general-purpose programming language that is not known to Iris. They are registered with Iris and may be invoked with other functions whose implementation is understood by Iris. Foreign functions may be used for a variety of applications such as arbitrary computations or data format conversions. A useful application of foreign functions was in providing access to other database management systems.

Versioning is often a requirement of engineering systems to keep track of design changes. Iris objects may be versioned. A versioned object has a *generic* object and one or more distinct *versioned* objects.

Access control is supported through an authorization scheme based on the single concept of controlling function evaluation. The granularity of access can be controlled using subtyping, user-defined operations, and function resolution.

[AHAD92] R. Ahad, J. W. Davis, S. Gower, P. Lyngbaek, A. Marynowsky, E. Onuegbe. Supporting Access Control in an Object-Oriented Database Language. *EDBT '92*, Vienna, Austria, March 1992 (to appear).

[ALBE91] J. Albert. Algebraic Properties of Bag Data Types. *Proc. 17th VLDB*, Barcelona, Spain, September 1991.

[BEEC88] D. Beech and B. Mahbod. Generalized Version Control in an Object-Oriented Database. *Proceedings of IEEE Data Engineering Conference*, Los Angeles, February 1988.

[CONN88] T. Connors and P. Lyngbaek. Providing Uniform Access to Heterogeneous Information Bases. *In Advances in Object-Oriented Database Systems, Lecture Notes in Computer Science 334, Springer-Verlag*, September 1988.

[KENT89] W. Kent. The Many Forms of a Single Fact. *COMPCON 89*, San Francisco, California, 1989.

[LYNG88] P. Lyngbaek and V. Vianu. Relational Translations of Semantic Models: A Case Study Based on Iris. *IEEE Database Engineering Bulletin*, Vol. 11, No. 2, June 1988.

2.2 Query Language

OSQL (Object SQL) is the Iris query language. Like SQL, OSQL is a declarative query language providing data definition and data manipulation capabilities and set-oriented access to information. It supports the Iris model while preserving many of the syntactic constructs of SQL.

Abstract types and function invocation are the main concepts in OSQL. OSQL manipulates objects that are instances of abstract types. The properties and behavior of objects are encapsulated by functions defined over their types. Actual data is extracted through function invocation, and although the OSQL syntax is similar to that of SQL, the actual details of data structures are hidden from users.

OSQL was extended into a general purpose database programming language by embedding it into a functional language. The Iris query evaluator was modified to allow the invocation of an embedded interpreter, thus extending OSQL with the power of a general purpose programming language.

[ANNE91] J. Annevelink. Database Programming Languages: A Functional Approach. *Proceedings of the ACM SIGMOD Conference*, Denver, Colorado, May 1991.

[BEEC88] D. Beech. A Foundation for Evolution from Relational to Object Databases. In *Lecture Notes in Computer Science 303, Advances in Database Technology - EDBT 1988*, J. W. Schmidt, S. Ceri, M. Missikoff, Eds. Springer-Verlag, 1988.

[LYNG91] P. Lyngbaek, et al. *OSQL: A Language for Programming Object Databases.* HPL-DTD-91-4, January 1991.

2.3 System Architecture

The Iris system implements the Iris model. Like the Iris model, the main paradigm of the Iris architecture is that of function evaluation. Furthermore, the kernel is reentrant thus allowing system functions to be implemented in terms of other system functions. The kernel consists of several modules: the executive, the query translator, the query interpreter, the object manager, the cache manager, and the storage manager.

The *executive* manages the interaction between Iris clients and the Iris system. It implements the main entry point to the Iris kernel. The *query translator* compiles an Iris functional expression into an optimized relational algebra expression. The *query interpreter* executes optimized expressions. The *storage manager* is a relational storage manager supporting the usual relation maintenance; tuple retrieval, insertion and deletion; indexed maintenance and access; concurrency control and recovery. The *cache manager* keeps a cache of the most recently used system tuples as well as some selected user tuples.

[LYNG90] P. Lyngbaek, W. K. Wilkinson, W. Hasan. The Iris Kernel Architecture. *Proceedings of EDBT*, Venice, Italy, March 1990.

[WILK89] W. K. Wilkinson, P. Lyngbaek, W. Hasan. The Iris Architecture and Implementation. *IEEE Transactions on Knowledge and Data Engineering*, December 1989.

2.4 Query Processing

The Iris query processor uses a rule-based approach to support extensibility - rules can be added or removed to support new data types or operations without affecting the rest of the system. Each rule consists of a precondition, and a transformation to be performed when the rule is applied; both are implemented as C procedures.

Query optimization procedures are expressed as rules and the entire query process is carried out in a way analogous to the operation of production systems in AI. The rule system is actually compiled, thus requiring no interpretation during query optimization. Two kinds of rules are used: the first kind transforms a query represented as an Iris functional expression, into an equivalent, but simpler, extended relational algebra expression; the second kind adds information to the relational algebra representation to specify join order and data access methods.

[DERR89] N. Derrett, M. C. Shan. Rule-Based Query Optimization in Iris. *Proceedings of ACM Annual Computer Science Conference*, Louisville, Kentucky, February 1989.

[SHAN88] M. C. Shan. Optimal plan search in a rule-based query optimizer. *Proceedings of EDBT*, Venice, Italy, March 1988.

2.5 Recursion

Iris was extended to support specific classes of recursively defined functions. The most common recursive queries are supported through this extension. The implementation is based on a data access method that allows the efficient computation of recursive functions. The extensions to the Iris system provide a complete treatment of recursion that spans extensions to OSQL, the query processor and the storage manager.

[DESM91] P. DeSmedt, S. Ceri, M.-A. Neimat, M.-C. Shan, R. Ahmed. *Recursive Functions in Iris.* HPL-DTD-91-7, February 1991.

[SHAN88] M.-C. Shan and H. Lu. A New Access Method Supporting Least Fixpoint Computation of Recursive Relations. *Proceedings of the International Computer Science Conference 1988*, Hong Kong, December 1988.

[SHAN91] M.-C. Shan and M.-A. Neimat. Optimization of Relational Algebra Expressions Containing Recursive Operators. *Proceedings of the 1991 ACM Computer Science Conference*, San Antonio, March 1991.

2.6 Active databases

Database monitors were introduced to continuously observe modifications to the values of objects in a database and to notify application programs of such changes. Monitors are specified declaratively through OSQL. The applicability of monitors can be localized both in time and space, so that Iris clients can have control over the amount of data being monitored and how frequently it is monitored. Such localization provides for efficient implementation.

[RISC89] T. Risch. Monitoring Database Objects. *Proceedings of the Fifteenth International Conference on Very Large Data Bases*, Amsterdam, The Netherlands, 1989.

2.7 Applications

The Iris system was used in two different applications to experiment with object-oriented database technology and to take advantage of features of the Iris system not found in other DBMSs. In the first case, Iris was used to design the schema of a computer integrated manufacturing application. The application's schema had been previously designed as a relational schema. The Iris schema was easier to understand and was substantially smaller than the relational schema.

In the second application, Iris was used to manage the data for a physician's workstation. The goal of that project was to provide a physician access to his/her own patients' data and, in addition, to provide access to numerous other external information bases. The external databases were accessed through Iris's foreign function facility. End-users and applications were given a single, logically centralized, object-oriented database schema, and could use it to interrogate the various underlying databases.

[ANNE91] J. Annevelink, C.Y. Young, P.C. Tang. Heterogeneous Database Integration in a Physician Workstation. *Proc. 15th Annual Symposium on Computer Applications in Medical Care*, New York, IEEE Computer Society Press, 1991.

[KETA90] M.A. Ketabchi, S. Mathur, T. Risch, J. Chen. Comparative Analysis of RDBMS and OODBMS: A Case Study. *COMPCON 90*, San Francisco, California, March 1990.

3 Heterogeneous Information Management

Advances in technology have produced an explosion in available data and an expectation that information is at one's fingertips. New databases, varied information bases and databases that evolved over the years and were of interest to restricted organizations must now be accessible to enterprise-wide decision makers. While the technology has made the data accessible, application developers are now faced with numerous

interfaces, conventions, protocols, semantics, schemas, query languages etc. Research in heterogeneous data management is not a new idea, but technological advances and business requirements are such that it has become the most urgent database challenge.

While one may be tempted to limit heterogeneous data management to the interoperability of autonomous general-purpose DBMSs, there exists a plethora of customized data managers that have been developed to solve very specific problems. They are typically narrow in scope, optimized to meet a specific need and are often driven by a high performance requirement. The interoperability of such data managers may be needed to construct new applications. For example a Geographic Information System may be constructed using attribute, vector and image data managers. The interoperability of specialized data managers may also be used to export specialized functionality to a more general framework. For example, one may want a general-purpose DBMS and a text retrieval system to interoperate. Specialized data managers are not typically guardians of corporate data. Because of their specialization, they are smaller and simpler than general-purpose DBMSs. Thus, the problem of integrating specialized data managers is not dominated by an autonomy requirement but rather by a performance requirement. In that context, parallelism and distribution are exploited to achieve high performance.

The current database research effort is divided between 3 complementary projects: *Perseus*, *Pegasus* and *Papyrus*. Perseus is focused on the graphical presentation and browsing of heterogeneous data. Pegasus is focused on the integration of heterogeneous data models, schema and semantics while preserving the autonomy of the underlying systems. Papyrus is focused on the integration and parallelization of specialized data managers that are unconstrained by an autonomy requirement. Both Pegasus and Papyrus are addressing different but complementary aspects of distributed and parallel query processing.

The *Perseus* project is focused on developing a graphical user interface that conveys to users the semantics of the underlying information bases. To achieve this goal, various presentation views are made available to users. Built on top of X-windows and the InterViews toolkit, it supports the graphical presentation of type hierarchies and function/type relationships, forms for browsing and querying specific object instances, and graphical query composition facilities. Extensibility is provided through customizable presentations so that it can be easily tailored to reflect the needs of specific users and applications.

The goal of the *Pegasus* project is to provide a seamless and integrated information environment from which users can easily and naturally obtain information and carry transactions. The data repositories are heterogeneous and distributed databases. Pegasus defines a common object model for unifying the data models of the underlying systems. It allows transparent as well as explicit access to multiple information systems in a declarative manner.

The goal of the *Papyrus* project is to provide tools and an infrastructure to facilitate the integration of specialized data managers. Data managers operators may be built using the Papyrus infrastructure or may be built independently of Papyrus. Special emphasis is placed on the performance of the integrated specialized data managers. In particular, parallel and distributed query execution is exploited to obtain high performance from the integrated systems. We do not target these integrated systems to one specific parallel computer configuration, but instead aim at benefiting from the various configurations available in computer installations ranging from tightly coupled multiprocessor systems to distributed systems. This is done by defining a language that permits the invocation and composition of data manager operators in a way that is independent of the computer configuration; and then providing a query optimizer and processor that can transparently target the execution to a variety of computer configurations.

The following sections describe some of the specific aspects of our research.

[AHME91] R. Ahmed, P. DeSmedt, W. Kent, M. Ketabchi, W. Litwin, A. Rafii, M.-C. Shan. Pegasus: A System for Seamless Integration of Heterogeneous Information Sources. *COMPCON 91*, San Francisco, California, March 1991.

[CONN91] T. Connors, W. Hasan, C. Kolovson, M.-A. Neimat, D. Schneider, K. Wilkinson, *The Papyrus Integrated Data Server*. HPL-DTD-91-15, May 16, 1991. Abstract to appear in *Proc. 1st International Conference on Parallel and Distributed Systems*, Miami Beach, Florida, December 1991.

[RAFI91] A. Rafii, R. Ahmed, P. DeSmedt, W. Kent, M. Ketabchi, W. Litwin, M.-C. Shan. Multidatabase Management in Pegasus. *Proceedings of the First International Workshop on Interoperability in Multidatabase Systems*, Kyoto, Japan, April 1991.

[SHAN89] M.-C. Shan. Unified Access in a Heterogeneous Information Environment. *Newsletter of IEEE Office Knowledge Engineering*, Vol. 3, No. 2, August 1989.

3.1 Interoperability

The term interoperability vaguely refers to the ability of different systems to operate with each other. This term is clearly overloaded as numerous factors either hinder or contribute to interoperability. We consider several factors and oversimplify them by grouping them under *semantic interoperability* and *operational interoperability*. In the first category, we broadly group the problems associated with domain, semantic and schema mismatch. In the second category, we group the operational or system components that contribute to interoperability such as factors that contribute to performance of the integrated system, the ability of different components to communicate with

each other, the conversions required to pass data from one system to the other, etc.

3.1.1 Semantic Interoperability

Research in this area is directed towards the ultimate goal of being able to easily express meaningful queries that span heterogeneous databases. Thus, one should be able to map the schemas of the underlying databases to a common schema that a user can subsequently use to interrogate the integrated databases. It is not uncommon that one will encounter semantic discrepancies when doing such a mapping. We are exploring the use of rules to reconcile schematic and semantic discrepancies of integrated databases. In constructing such mappings, one may need to interrogate the schemas of the underlying databases. We are also exploring the use of higher order expressions and higher order views to be able to interrogate database schemas by using variables that range over the schemas.

Uniform and declarative access to multiple heterogeneous databases is provided through a unifying data definition and data manipulation language, HOSQL. HOSQL is a functional object-oriented language that is an extension of Iris's OSQL query language. By using an object-oriented paradigm, the unified schema of multiple databases can easily model conceptual entities that span the underlying databases. HOSQL provides non-procedural statements to manipulate multiple databases. It allows for both transparent and explicit access to participating databases.

[AHME90] R. Ahmed and A. Rafii. Relational Schema Mapping and Query Translation in Pegasus. *Workshop on Multidatabases and Semantic Interoperability*, Tulsa, Oklahoma, November 1990.

[KENT91] W. Kent. Solving Domain Mismatch and Schema Mismatch Problems with an Object-Oriented Database Programming Language. *Proc. 17th VLDB*, Barcelona, Spain, September 1991.

[KRIS91a] R. Krishnamurthy, W. Litwin, W. Kent. Interoperability of Heterogeneous Databases with Schematic Discrepancies. *Proceedings of the First International Workshop on Interoperability in Multidatabase Systems*, Kyoto, Japan, April 1991.

[KRIS91b] R. Krishnamurthy, W. Litwin, W. Kent. Language Features for Interoperability of Databases with Schematic Discrepancies. *Proceedings of the ACM SIGMOD Conference*, Denver, Colorado, May 1991.

[NEUH91] E. Neuhold, W. Kent, M.-C. Shan. Object Identification in Interoperable Database Systems. *Proceedings of the First International Workshop on Interoperability in Multidatabase Systems*, Kyoto, Japan, April 1991.

[RAFI92] A. Rafii, R. Ahmed, P. DeSmedt, W. Du. Integration Strategies in Pegasus Object Oriented Multidatabase System. *HICSS*, January 1992 (to appear).

3.1.2 Operational Interoperability

While HOSQL captures the behavior of abstract data types, it does not capture their operational properties. Describing the operational properties of functions is essential to optimizing their usage. Such properties describe the cost of executing functions and how they interact with other functions. The Papyrus Interface Language (PIL) is a functional language with built-in optimizable constructs for handling data sets. Its primary purpose is to invoke data managers' operators, provide mechanisms for processing and communicating data among operators, and to permit optimization and parallelization. PIL places emphasis on the characteristics of data manipulation algorithms through operator properties. Operator properties may be explicitly declared or the operator may be declared to match a template whose behavior is well-understood by the optimizer. Examples of such templates are *pick* an element from a bag, *scan* a bag, or apply a *filter* to an atom. Through these properties and templates the optimizer can gain an understanding of the operational characteristics of functions and exploit them to come up with better execution plans. Similar properties and templates can be used to capture the potential for parallelism between functions and within individual functions.

3.2 Transaction Management

Transaction management poses a serious problem when integrating heterogeneous databases. Some databases may have a transaction manager, others may not. Among the databases that do have a transaction manager, their transaction model may not all be the same. The problem faced by heterogeneous database systems is how to reconcile these differences. Even if all the integrated systems abide by a two-phase commit protocol, the fact that the underlying databases may be distributed can pose a severe performance degradation if one participating database does not respond to the request to commit for any of a multitude of reasons. We have explored alternatives to the two-phase commit protocol.

A challenge in integrating special-purpose data managers is that they may have very diverse transaction management requirements. Assuming that autonomy can be sacrificed, we explore the extent to which they can participate in global transactions while preserving their individual requirements.

[LITW91] W. Litwin and M.-C. Shan. Value Dates for Concurrency Control and Transaction Management in Interoperable Systems. *Proceedings of the First International Workshop on Interoperability in Multidatabase Systems*, Kyoto, Japan, April 1991.

[NEIM90] M.-A. Neimat, W. K. Wilkinson. Extensible Transaction Management in Papyrus. *HICSS*, Jan 1990.

3.3 Parallelism and Distribution

Parallel and distributed execution of queries will play an essential role in improving the performance of integrated systems. Autonomous legacy databases are likely to be distributed in an enterprise's computer network. The execution of queries that span the legacy databases should be executed in parallel on the distributed nodes to achieve reasonable response time. For the specialized, highly-tuned data managers, integration can mean loss in performance. We focus on the parallelization of such integrated data managers and on the retargetability of their executions to a variety of parallel and distributed computer architectures.

An important class of queries is the Select-Project-Join queries so commonly found in relational database systems. We have researched various ways of optimizing their execution on shared memory multiprocessor architectures. We are also exploring their optimization on more general parallel architectures.

To process queries that span heterogeneous data managers and can execute on parallel and distributed architectures, one needs an execution framework that can accommodate such requirements. We have designed an execution engine that supports efficient interaction among data managers and their operators regardless of the internals of the operator and of the hardware and processor configuration. Operators are only understood through their interface and properties. Some may be implemented as iterators[2] and some may not. A program may execute in several address spaces in the same machine or on different machines. Some of these machines may be multiprocessors. One of the challenges we have imposed on the execution engine is that it be efficient on both single-processor and multiprocessor architectures. Unlike most query processors, the execution engine supports some advanced constructs, such as loops and conditionals. Parallelism depends on many factors that are only known at runtime. For example, the effective number of processors available at any time is dependent on the workload. The execution engine automates much of the cloning of operators, data partitioning and data passing between operators, to adjust the level of parallelism to the runtime environment.

Parallel and distributed query processing is not limited to the server, as client workstations are assumed to have sufficient processing power to reduce contention on the server and thus contribute to overall improvements in performance. We have addressed one aspect of client participation in query processing, namely the maintenance of a consistent cache on client workstations. We have also explored the use of main memory data structures on the client to improve performance.

[CONN90] T. Connors and D. Schneider. Query Processing in Papyrus. *Submitted for publication.*

[2] Iterator functions perform only a partial computation for each invocation of the function. Traditionally, at least three entry points are defined for each function, *open*, *next* and *close*.

[HASA91] W. Hasan and S. Chaudhuri. *Pipelined Execution in an Extensible DBMS.* HPL-DTD-91-10, February 22, 1991.

[LITW91] W. Litwin and T. Risch. *Efficient Processing of an OO Query Language through Datalog with Foreign Predicates.* HPL-DTD-91-17, June 19, 1991.

[LU90] H. Lu, K.-L. Tan, M.-C. Shan. Hash-Based Join Algorithms for Multiprocessor Computers with Shared Memory. *Proc. 16th VLDB*, Brisbane, Australia, August 1990.

[LU91] H. Lu, M.-C. Shan, K.-L. Tan. Optimization of Multi-Way Join Queries for Parallel Execution. *Proc. 17th VLDB*, Barcelona, Spain, September 1991.

[MURP91] M. Murphy and M.-C. Shan. Execution Plan Balancing. *Proc. 7th Data Engineering Conference*, Kobe, Japan, April 1991.

[WILK90] W. K. Wilkinson and M.-A. Neimat. Maintaining Consistency of Client-Cached Data. *Proc. 16th VLDB*, Brisbane, Australia, August 1990.