Appeared in Proc. RIDE-DOM '95 (5th Int'l Workshop on Research Issues in Data Engineering: Distributed Object Management), 1995, pp. 124-131.

# Towards Heterogeneous Multimedia Information Systems: The Garlic[1] Approach

**Michael J. Carey, Laura M. Haas, Peter M. Schwarz, Manish Arya, William F. Cody, Ronald Fagin, Myron Flickner, Allen W. Luniewski, Wayne Niblack, Dragutin Petkovic, John Thomas, John H. Williams and Edward L. Wimmers**
**IBM Almaden Research Center**

**Abstract**: We provide an overview of the Garlic project, a new project at the IBM Almaden Research Center. The goal of this project is to develop a system and associated tools for the management of large quantities of heterogeneous multimedia information. Garlic permits traditional and multimedia data to be stored in a variety of existing data repositories, including databases, files, text managers, image managers, video servers and so on; the data is seen through a unified schema expressed in an object-oriented data model and can be queried and manipulated using an object-oriented dialect of SQL, perhaps through an advanced query/browser tool that we are also developing. The Garlic architecture is designed to be extensible to new kinds of data repositories, and access efficiency is addressed via a "middleware" query processor that uses database query optimization techniques to exploit the native associative search capabilities of the underlying data repositories.

## 1 Introduction

In recent years, the problem of heterogeneous distributed data management has attracted a great deal of attention, both from the database systems research community and from developers of commercial database systems. This is due to the fact that improvements in communication technologies, such as affordable high-speed lines, broad deployment of LAN technology, and availability of standardized protocols, have made it feasible to connect historically separate data systems. In commercial environments, such data systems typically include relational databases from various vendors (e.g., IBM, Oracle, Sybase), older, non-relational databases of various genres (e.g., IMS databases), and record-based file systems (e.g., VSAM). Unfortunately, simply connecting all of the related systems does not solve the problem of writing applications that require access to enterprise data from several of them. Though the systems are connected, the data is still represented by different data models, meaning that the programmer must use different access interfaces to get at the data, and must worry about such details as locating the desired data, optimizing access to the different data systems, and managing the transactional consistency of any updates performed.

For traditional business data, solutions are becoming available to some of the problems mentioned above. However, in both commercial enterprises (e.g., retailing or insurance) and in more specialized environments (e.g., health care or engineering design), non-traditional, multimedia data is assuming an increasingly central role. Further, these new types of data are needed for applications (e.g., catalog production, patient

1. Garlic is not an acronym. Most members of the team really like garlic, and enjoy our laboratory's proximity to the Gilroy garlic fields!

records) which also involve traditional, record-oriented data. In both commercial and more specialized environments, the existing multimedia data often resides in file-based data systems that provide media-specific capabilities for searching, storing, and delivering the (often large) multimedia data items. Examples of such media-specific data systems include image management systems (e.g., QBIC [1], Excalibur [8], and Photobook[2]), information retrieval systems for text (e.g., [3], [4], [5], [6] and [7]), and video servers (e.g., MediaServer [9] and Shark [10]). This increased heterogeneity of systems and data types significantly adds to the problems faced by application developers and end-users.

If access to heterogeneous multimedia data is to become as commonplace as the emerging generation of applications seems to demand, it will be necessary to develop uniform interfaces for providing location, network, and data model transparency for application developers. Otherwise, interface heterogeneity will inhibit the deployment of large-scale multimedia information systems. Providing uniform access to heterogeneous multimedia data presents a number of new challenges and system requirements. First, the user and application programming interfaces to the data must be flexible enough to support the new user interaction models that are characteristic when dealing with multimedia data. These include visual query formation and support for both navigation-based querying and similarity queries. Secondly, for multimedia information systems to be scalable, they must provide convenient mechanisms for integrating additional (new or legacy) data sources and data collections into the system and for smoothly extending the system's querying capabilities to cover such newly integrated data sources.

The goal of the recently formed Garlic project at IBM's Almaden Research Center is to build a multimedia information system (MMIS) capable of integrating data that resides in different database systems as well as in a variety of non-database data servers. This integration must be enabled while maintaining the independence of the data servers, and without creating copies of their data. "Multimedia" should be interpreted very broadly to mean not only images, video, audio, but also text and application specific types of data (CAD drawings, medical objects, maps, ...). Since much of this data is naturally modeled by objects, Garlic provides an object-oriented data model that allows data from various data servers to be represented uniformly. This data model is complemented by an object-oriented query language (an object-extended dialect of SQL) and supported by a "middleware" layer of query processing and data access software that presents the object-oriented schema to applications, interprets object queries and updates, creates execution plans for sending pieces of queries to the appropriate data servers, and assembles query results for delivery back to the applications. A significant focus of the project is the provision of support for "intelligent" data servers, i.e., servers that provide media-specific indexing and query capabilities. Database optimization technology will be extended to deal with heterogeneous collections of data servers so that efficient data access plans will be employed for multi-repository queries. It is hoped that the Garlic approach to unifying diverse data sources will significantly simplify application development, make end-user data exploration easier, and simplify the problem of integrating new data sources to support the deployment of large-scale, multimedia applications.

The remainder of this paper provides a general introduction to the Garlic project, the approach being taken, and some of the key research issues in this area. The paper begins with a discussion of related work in Section 2. Section 3 provides a general overview of the project, describing the architecture of the system and the role of each of its components. Section 4 then describes the Garlic data model, with Section 5 describing the Garlic query language and the planned query processing approach. Finally, Section 6 describes the interfaces that Garlic will provide, both for applications and for end users. Section 7 concludes with a discussion of the current status of the project and our plans for the immediate future.

# 2 Related Work

Since the Garlic project is focused on the design and implementation of a heterogeneous multimedia information system based on database concepts and technology, there are two areas where significant related work exists: heterogeneous databases and multimedia systems. We discuss both briefly here, focusing primarily on how Garlic differs from previous work in these two areas.

## 2.1 Heterogeneous Databases

The area of heterogeneous distributed database systems (also known as multidatabase systems) has been the focus of a significant level of research activity over the past five years. Early work in this area actually dates back much further, to projects such as the Multibase effort at CCA [11], but improved communication technologies and resulting commercial demands have led to a recent resurgence of work in the area. Much of the research on heterogeneous distributed database systems falls into one of three broad categories: providing uniform access to data stored in multiple databases that involve several different data models; handling similarities and differences between data representations, semantics, and stored values when the same (or similar) information about a given real-world entity is stored in multiple databases; and supporting transactions in heterogeneous distributed database systems in the face of incompatible concurrency control subsystems and/or uncooperative transaction managers. Good surveys of the relevant work can be found in [12], [13], [14], and [15]. In the commercial realm, products now exist for providing uniform access to data in multiple databases, relational and otherwise, and to structured files, usually through the provision of a unified relational schema. Early examples of such products include Microsoft Access, Information Builders EDA/SQL, Sybase Omni SQL, and IBM's DataJoiner, to name a few. It is projected that such "middleware" products will form one of the fastest-growing segments of the database market over the next few years.

Garlic's use of an object-oriented data model to integrate multiple databases is not new, as models with object-oriented features have been employed in projects such as [11], [16], [18] and others. However, a number of the technical aspects of our approach are quite novel, and again, the intended range of target data repositories and data types is much broader. What distinguishes the Garlic project from the aforementioned efforts is its focus on providing an object-oriented view of data residing not only in databases and record-based files, but also in a wide variety of media-specific data repositories with specialized search facilities. With the exception of the Papyrus [16] and Pegasus [17] projects at HP Labs, we are aware of no other efforts that have tried to address the problems involved in supporting heterogeneous, multimedia applications.

## 2.2 Multimedia Systems

The multimedia area today is expanding at an extremely rapid pace, and the area is extremely broad, including work on hypermedia systems, specialized servers with stringent quality of service requirements (e.g., video servers), image and document management tools, interactive games, structured presentations involving mixtures of text, imagery, audio and video data, scripting languages to support timed and synchronized playback of multimedia presentations, and so forth. This is most evident in the personal computer industry, where a large number of small-scale multimedia software packages and products have emerged due to the availability and affordability of CD-ROM technology. In the information systems industry, despite the fact that there is clearly a strong demand for multimedia technologies, things are moving more slowly. In particular, work on multimedia information systems is still in its infancy, with many problems remaining to be

solved in terms of combining multimedia data with more traditional, record-oriented data and supporting associative retrieval and data-intensive multimedia applications [19].

Most work to date on supporting associative retrieval of multimedia data has focused on retrieval methods for specific data types. For example, years of research have produced a solid technology base for content-based retrieval of documents through the use of various text indexing and search techniques [3], [4], [5], [6]. Similarly, simple spatial searches are well-supported by today's geographic information systems ([47], [48], [49], e.g.), and work in the area of image processing has led to systems such as QBIC [1] and Photobook [2] in which images can be recalled based on features such as the color, texture, and shape of scenes and user-identified objects. However, with the exception of simple approaches like attaching attributes to spatial objects, or associating user-provided keywords with images, these component search technologies remain largely isolated from one another. The only notable exception to this is the approach being taken by systems such as the Illustra object-relational DBMS [20], where media-specific class libraries (which Illustra calls DataBlades(tm)) are provided in order to allow multimedia data to be stored in and managed by the DBMS. Garlic differs in that it aims to leverage existing intelligent repositories, such as text and image management systems, rather than requiring all multimedia data to be stored within and controlled by the DBMS. Our belief is that Garlic's open approach will enable it to take advantage of continuing advances in multimedia storage and search technology. It should also be more effective for legacy environments, where multimedia data collections (such as documents or image libraries) and business data already exist in forms that cannot simply be discarded or migrated into a new DBMS.

# 3 Garlic Overview

Loosely speaking, the goal of the Garlic project is to provide applications and users with the benefits of a database with a schema -- similar to what an object-oriented or object-relational database system might provide -- but without actually storing (at least the bulk of) the data within the Garlic system proper. Viewed from above, then, Garlic looks rather like a DBMS with an object-oriented schema. Internally, however, Garlic looks very different, with the data that makes up its database being scattered across a number of different data sources. To achieve the former view from the latter reality, Garlic provides two forms of "glue": an object-oriented data model and the ability to store additional complex objects to augment the data in the underlying data sources.

## 3.1 The Big Picture

Figure 1 depicts the overall architecture of the Garlic system. At the leaves of the figure are a number of data repositories containing the data that Garlic is intended to integrate. As mentioned earlier, examples of reasonable data repositories include relational and non-relational database systems, file systems, document managers, image managers, and video servers. Above each repository in the figure is a repository wrapper, the purpose of which is to translate information about data types and collections (i.e., schemas) and data access and manipulation requests (i.e., queries) between Garlic's internal protocols and each repository's native protocols. Information about the unified Garlic schema, as well as certain translation-related information needed by the various data repositories, is maintained in the Garlic metadata repository shown in Figure 1. The other repository shown in the figure is the Garlic complex object repository. This repository is used to hold the complex objects that most Garlic applications will need for "gluing" together the under-

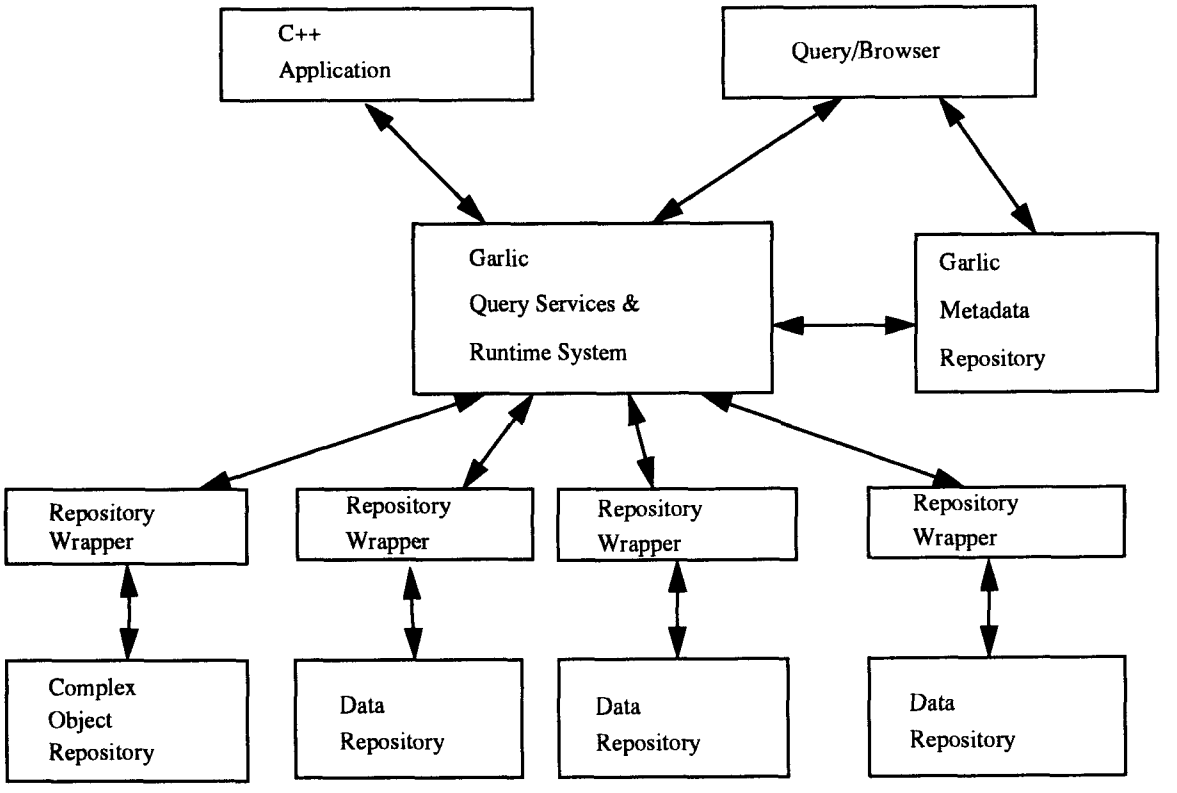


**Figure 1. Garlic System Architecture**

lying data in new and useful ways. For example, in an auto insurance application, Garlic complex objects could be used to link the images of a damaged car (stored in an image-specific repository) together with an accident report (stored in a document management system) and a customer's claim and policy records (residing in a relational database) in order to form a "claim folder" object to be dealt with by an insurance agent.

Query processing and data manipulation services, especially for queries where the target data resides in more than one repository, are provided by the Garlic query services and runtime system component shown in Figure 1. It is the responsibility of this component to present Garlic applications with a unified, object-oriented view of the contents of a Garlic database and to process users' and applications' queries, updates and method invocation requests against this data; queries are expressed in an object-oriented extension of the SQL query language. This component will also be responsible for dealing with transaction management issues. In the near term, Garlic will not provide any guarantees about consistency of legacy data that is operated on by legacy (as well as Garlic) applications. The new Exotica project at Almaden is examining a combination of workflow with transactions which may offer some help for environments such as Garlic's.

Finally, Garlic applications interact with the query services and runtime system through Garlic's object query language and a C++ application programming interface (API). Many applications will do this statically, in which case the Garlic schema will be presented to the application via a set of C++ classes that act as "surrogates" for the corresponding classes of the actual Garlic schema. Certain applications may require more dynamic access to the data, in which case they will use a portion of the C++ API that provides dynamic ac-

cess to information about the types and objects contained in a Garlic database and that enables objects to be manipulated without *a priori* (i.e., compiled) knowledge of the database schema. A particularly important example of such a dynamic application, which is also shown in Figure 1, is the Garlic query/browser. This component of Garlic will provide end users of the system with a friendly, graphical interface that supports interactive browsing, navigation, and querying of the contents of Garlic databases.

## 3.2 Repositories and Databases

As mentioned previously, the purpose of the Garlic system is to integrate and unify data managed by multiple, disparate data sources. Thus, virtually all Garlic databases will involve data, most likely including legacy data, that resides in several data repositories. Up to this point, we have been using the term "data repository" loosely, referring both to a particular kind of data management software (e.g., the DB2 C/S database system, the PixTex/EFS document manager, the QBIC image management library, etc.) and to the collections of data that it manages. However, to explain how we envision Garlic databases being created and used, it is useful to distinguish between the notion of a *repository type*, which is a particular kind of data management software, and a *repository instance*, which is a particular data collection that it manages. We will also refer to a *repository manager*, which is an instance of a repository type (for example, a particular DB2 C/S installation). In order for Garlic to provide access to data that resides in a given repository manager, someone must have written a repository wrapper for that repository type. The Garlic architecture is designed to be extensible in this dimension, and we plan to provide tools to ease the task of writing such wrappers; we will also write such wrappers ourselves for a set of interesting repository types. Given the existence of a wrapper for a repository type of interest, it is then possible to create Garlic databases that include one or more repository instances that are managed by repository managers of that repository type.

The contents of a Garlic database, as mentioned earlier, are presented to applications and to end users as a global schema expressed in terms of the Garlic data model. The Garlic data model, discussed further in Section 4, is based on a proposed object-oriented data model standard [21]. The global schema is in turn the integration of a number of local schemas, one per repository instance, that describe the data contents of each repository instance (including the Garlic complex object repository) that contributes data to the given Garlic database. Each of the local schemas, which are referred to in Garlic as wrapper schemas, is expressed in terms of the Garlic data model as well.[2] Enabling access to the contents of a repository instance through a Garlic database involves identifying the data types and collections that are to be visible as part of the Garlic database and then writing a wrapper schema that describes the target data in GDL (the Garlic Data Language, the syntactic form of the Garlic data model), as well as providing any code required to implement the types' behavior (normally these will form a thin layer on top of the repository type's programming interface). For important repository types that support type systems of their own, e.g., a relational DBMS such as DB2 C/S, we expect that automated schema mapping tools will be provided to assist Garlic database administrators (DBAs) with the task of defining a Garlic wrapper schema to represent the native data types and collections that they wish to export from a given repository instance.

---

2. It should be noted that, in the simplest of cases, the global schema may be nothing more than the union of the local schemas. However, it is more likely that the global schema will also involve views that serve to redefine, reshape, and hide some of the data definitions found in the underlying local schemas (see Section 4.4).

# 4 The Garlic Data Model

The past decade of research in the object-oriented database (OODB) area has produced a plethora of ideas and proposals for object-oriented data models. As yet, there is no single, widely accepted, standard object data model. However, a consortium of major OODB vendors recently proposed a candidate for such a model, the ODMG-93 Object Database Standard [21], which was developed by adding database concepts to the OMG Object Model. Our goal in Garlic is to understand the issues associated with providing an integrated, object-oriented view of data from disparate sources, and not to invent yet another object-oriented data model, so we have adopted the ODMG-93 object model as a starting point for the Garlic data model and the syntax of the ODMG-93 object definition language, ODL, as a base for the Garlic Data Language, GDL. Some aspects of the Garlic data model differ from those of the ODMG-93 model, however, as Garlic's heterogeneous environment poses certain problems not found in the logically centralized, OODB environment with which the ODMG-93 standardization effort is concerned. We begin this section by summarizing the ODMG-93 model; we then proceed to explain how Garlic extends and differs from the proposed ODMG-93 standard.

## 4.1 The ODMG-93 Object Model

In the ODMG-93 standard, the fundamental building blocks of the data model are *objects* and *values*. Each object has an identity that uniquely denotes the object, thus enabling the sharing of objects (by reference). Objects are strongly typed, and object types are expressed in the data model in terms of object *interfaces* (as distinct from *implementations*). The description of an object's interface includes the attributes, relationships, and methods (see below) that are characteristic of all objects that adhere to the interface. The model also supports an *inheritance* mechanism by which a new interface can be derived from one or more existing interfaces. The derived interface inherits all of the attributes, relationships, and methods of the interfaces from which it is derived, making the derived interface a subtype of those interfaces according to the usual notion of subtyping.

The *attributes* of an object are analogous to the fields of a record, and accessing an attribute returns a value. Values[3], in turn, can be *base* values (such as integers, strings, floating-point numbers, and object references) or structured values (which are, roughly, interfaces without identity). Unlike objects, values cannot be denoted by means of references and therefore cannot be shared (only copied). Built into the data model are several interfaces that provide for *collections* of values; these include interfaces for sets, lists, bags, and arrays. These collection interfaces are parameterized by their element type, as all elements must adhere to a specified interface (modulo inheritance). A *relationship* is a reference-valued attribute; the model also supports the notion of an *inverse relationship*, or bidirectional relationship, which is a relationship enhanced with a system-maintained constraint between the reference-valued attributes of two or more objects. The data model supports the modeling of one-to-one, one-to-many, and many-to-many relationships. Finally, *methods* are functions associated with an interface that take zero or more parameters with specified interfaces as arguments and may return a value with a specified interface.

3. Values are also referred to as *immutable objects* or *literals* in some sections of the ODMG-93 book.

## 4.2 Garlic ODMG-93 Extensions

The Garlic data model extends the concepts of the ODMG-93 object model in two significant ways. The first is the degree of support for alternative implementations of interfaces, and the other is related to type system flexibility. In addition, as Section 4.4 will discuss separately, Garlic further extends the ODMG-93 model by introducing an object-appropriate view definition facility.

Garlic makes a sharp distinction between an interface and its implementations. The type of an object is determined solely by its interface, and any number of implementations of a given interface are permitted. The contents of a given data repository in Garlic are exposed in terms of a set of interfaces for which one or more implementations are provided by that repository, and it is quite possible that several repositories may offer alternative implementations of an important multimedia data type (e.g., text or image). The ODMG-93 standard also makes an interface versus implementation distinction when first presenting the concepts of its object model; however, this distinction disappears in the concrete C++ realization of the model, where a one-to-one relationship is implicitly assumed between interfaces and implementations. Moreover, Garlic supports both the notion of a type extent, which is the set of all instances of a given interface, and an implementation extent, which is the set of all instances managed by a given implementation of an interface of interest. ODMG-93 supports only type extents in its abstract object model, and even these disappear in the standard's C++ realization of the model.

In terms of type system flexibility, Garlic extends the ODMG-93 model with the concept of conformity [22]. One interface is said to conform to another if the former defines a subtype of the latter under the standard definition of subtyping (including contravariance in the argument positions) -- even if the former interface was not derived from the latter interface by means of the data model's explicit inheritance mechanism. This notion results in an implicitly-specified type lattice that can provide additional flexibility when independently-defined schemas are merged later, which is clearly an important consideration for Garlic. However, to be compatible wtih ODMG-93 semantics, conformity in Garlic is provided as an option. The definer of an interface A may specify, when using the name of an interface B, whether that use of B should be restricted to mean instances of B and its explicitly derived subtypes, or whether any interface that conforms to interface B is acceptable. This is indicated by saying *conforms(B)* instead of *B* in the relevant part of the definition of A. For each interface known to the system, the Garlic data model actually defines two extents that can be enumerated and queried: one for all instances of the interface and its explicit subtypes, and one for all instances that conform to the interface (this latter extent is a superset of the first).

## 4.3 Garlic ODMG-93 Differences

Due largely to problems introduced by the heterogeneous nature of Garlic and a desire to provide object-oriented access to legacy data, the Garlic data model differs from that of ODMG-93 in its treatment of object references and integrity constraints.

**Object identity:** The ODMG-93 data model defines object identity in the traditional strong manner, guaranteeing that an object's identity be both unique and immutable. Unfortunately, some repositories, such as relational databases, do not provide this strong notion of identity for the data items that they manage. To enable such data items to be modeled as objects in a Garlic database, references in the Garlic data model are based on a notion that we call *weak identity*; this is simply a means of denoting an object uniquely but not necessarily immutably within the scope of a Garlic database. An object's weak identity is formed by con-

catenating a token that designates the object's implementation with an implementation-specific unique key. For instance, to view a table in a relational database as a collection of objects in Garlic, the identity of the object corresponding to a relational tuple could be formed by pairing an implementation identifier that (indirectly) specifies the tuple's source table together with the values of the key field(s) of the tuple. An important consequence of weak identity is that it allows Garlic to treat data items for which only weak identification is possible as objects without requiring that proxy (or surrogate) objects be maintained within Garlic for each such data item. Although some systems have employed a proxy-based approach to heterogeneity in object bases (e.g., [18]), we felt that such an approach would cause serious problems in terms of practicality (both space and the cost of maintaining consistency are at issue) and efficiency of access for large legacy databases.

**Legacy references:** Another problem related to identity and object references in Garlic is the fact that legacy data can contain legacy references. For example, many foreign keys in a relational database are essentially object references, and we would like to show them as such in the portions of the Garlic schema that correspond to the underlying relational data. Similarly, in a repository that stores and indexes documents using a traditional file system for document storage, file names for documents are essentially references to document objects, and they should be shown as such in the Garlic schema. Unfortunately, in both cases, the underlying repository represents and manipulates these legacy references in its own, repository-specific manner. Clearly, to provide access to legacy databases, Garlic cannot require such reference attributes to be converted into and stored using Garlic's full weak identifier format; in fact, Garlic must be careful not to attempt to store long-form references into the reference attributes of legacy data objects. The Garlic data model addresses this problem by providing the notion of an *implementation-constrained reference*, which is a reference that can only denote objects of a specified implementation. Returning to the example of a foreign key, a relational tuple that references another tuple by means of a foreign key cannot reference any arbitrary Garlic object that has the appropriate interface; rather, it can only reference tuples in the specified target table within the same relational database. Thus, when introducing an implementation of a given interface, Garlic supports the association of such implementation constraints with the reference attributes mentioned in the interface. Repository wrappers (see Figure 1) are responsible for converting such reference values between Garlic's object reference format and their repository-specific short forms as needed. Attempts to violate such constraints are detected by the relevant repository wrapper and result in a runtime error.[4]

**Other differences:** In addition to these reference-related differences, there are two other minor differences between the Garlic data model and the ODMG-93 model, owing to constraint specifications in ODMG-93 and legacy database support in Garlic. The first difference is that, according to the ODMG-93 object model, an interface definition can optionally specify the key attribute(s) for its objects; the system is then expected to enforce the uniqueness of the key across all object instances corresponding to the interface. Garlic does not support the specification of key attributes for interfaces, as it has no way to enforce such uniqueness constraints for legacy databases that may be accessed and updated by legacy applications (i.e., applications that operate completely outside the realm of Garlic). Interestingly, the concrete C++ realization of the ODMG-93 model does not support key attributes either. The second minor difference is that, according to

4. It should be noted that while most aspects of the Garlic data model are amenable to static type-checking, this solution is not. However, when a new interface or implementation with the potential to lead to such run-time errors is introduced into the schema, Garlic can provide a warning and even (optionally) prohibit the change.

the ODMG-93 object model, the definer of an interface has the option of specifying inverse relationships for any of the interface's relationship attributes. Garlic supports this option fully for Garlic complex objects, i.e., for non-legacy objects managed within the Garlic complex object repository, but such specifications are treated as purely advisory for objects that reside in other repositories. Again, such objects may be updated by legacy applications, so Garlic cannot possibly enforce such constraints.

## 4.4 Object-Centered Views

The most significant extension that Garlic makes to the ODMG-93 data model is the notion of views, the lack of which is seen by some as a significant shortcoming of the ODMG-93 model [34]. In their most general form, views can be problematic in object-oriented database systems ([23],[24], [25]), raising questions such as "where do views fit in the type hierarchy?", "what methods are available on the objects in a given view?", "are the elements in a view values or objects?", and "what is the identity of the objects in a view?" (if views contain objects). For Garlic, we have developed a view mechanism that is sufficiently restrictive in its expressive power to allow simple answers to these and other such questions; however, it is still sufficiently powerful to satisfy the needs that Garlic schemas are likely to have for views.

In Garlic, the primary purpose of a view is to enhance (extend, simplify, or reshape) a set of underlying Garlic objects, usually by adding or hiding some of their attributes and/or methods. Garlic introduces the notion of an *object-centered view* for this purpose. An object-centered view defines a new interface, together with an implementation of the new interface (usually written using Garlic's object query language), that is based on an existing interface that the view definer wishes to enhance. The existing interface is said to be the *center* of the view, as each element of the view is an enhancement of an object instance whose type is given by the center interface. In the global schema, the view assumes a role similar to a type extent, as its elements can be queried and enumerated just like the objects in a type extent. The elements of a view are considered to be objects, and the identity of each object in a view is derived from the identity of the center object that it enhances. In particular, the identity of an object in a view consists of the identity of its center object prepended with an implementation identifier that indicates the view from which the object was obtained; this allows Garlic to properly handle references to view objects when presented with them later. The methods available on the objects in an object-centered view are selected (or additionally provided) by the view definer, and a view can optionally be positioned in the type hierarchy through explicit placement when the view's interface is defined.

To make the idea of object-centered views clearer, consider the insurance scenario mentioned earlier, which had its customers' claim and policy records in a relational database, its images of damaged cars in an image-specific data repository, its accident reports in a document management system, and which used Garlic complex objects to create claim folders linking each customer claim record to the associated car images and accident report. In the global Garlic schema, it would be nice to hide from the agents the complexity of these extra claim folder structures. An object-centered view centered on claim folders could make claim objects appear to directly contain those attributes of a claim folder that are relevant for claims processing, hiding the extra level of indirection present in the underlying data objects and hiding any irrelevant attributes. Thus, it could make the text of the accident report and the pictures of the damage appear as attributes of the claim, and could provide a method to display the images of the damage that led to the claim.

**Relational DBMS**

```
create table Policy (
   pol_no integer,
   cust_name char(40),
   cust_address char(80),
   start_date date,
   primary key (pol_no)
)
create table Claim (
   pol_no integer,
   claim_no integer,
   date_filed date,
   handler char(40),
   status char(12),
   primary key (pol_no, claim_no),
   foreign key (pol_no) refs Policy)
)
```

**Text Repository**

```
class Document {
private:
   ....
public:
   char* text;
   int matches(char* search_expr);
   ....
}
make_doc_db /insur/documents
add_doc /insur/claim1.text
add_doc /insur/claim2.text
....
```

**Image Repository**

```
class ImagePredicate {
private:
   ....
public:
   ImagePredicate();
   ImagePredicate(char *);
   ....
}
class Image {
   ....
public:
   void display();
   int matches(ImagePredicate& pred);
   ....
}

mkdir /insur/images
qbic_make_db /insur/images/damage.dbm
qbic_import_image /insur/images/
     damage.dbm claim1.gif
qbic_import_image /insur/images/
     damage.dbm claim2.gif
....
```

**Figure 2. Data Repository Contents**

## 4.5 An Example

This section uses the Garlic data model to describe the schema for the auto insurance scenario. Figure 2 depicts the contents of the three data repositories involved in the example -- a relational database, an image database, and a collection of text documents. There are two tables in the relational database, with one containing information about customers' policies and the other containing records describing customers' accident claims. The image database contains one or more images of the damaged car associated with each claim, and there is an accident description stored in the document collection that corresponds to each claim record. Figure 3 shows the wrapper schemas that result from describing each repository's contents using GDL, the Garlic data language (Figure 3a). These wrapper schemas contain interface definitions for each type of data in the underlying data repositories. Also shown in Figure 3 is an interface definition for the complex objects, managed by the Garlic complex object repository, for linking claims in the relational database to their associated damaged car images and accident descriptions in the other two repositories (Figure 3b). To keep things simple, we have made several simplifying assumptions, e.g., that each policy covers exactly one car, that each claim involves only one accident description, and so on. Finally, Figure 4 shows the interface definitions to be included in the Garlic global schema for the insurance database; these interfaces can be provided by incorporating and defining views over the wrapper schemas of Figure 3. (Note that these are just the views' GDL interface definitions; the details of the view definitions are not shown).

**Wrapper Schema for Relational DBMS:**

```
interface Policy(extent Policy):persistent
{
   attribute Unsigned Long pol_no;
   attribute String cust_name;
   attribute String cust_address;
   attribute Date start_date;
}
interface Claim (extent Claim): persistent
{
   relationship Policy policy;
   attribute Unsigned Long claim_no;
   attribute Date date_filed;
   attribute String handler;
   attribute String status;
}
```

**Wrapper Schema for Image Repository:**

```
interface ImagePredicate: transient {
   void makeImagePredicate();
   void makeImagePredicate(in String);
   ....
}
interface Image (extent Image): persistent
{
   void display();
   Boolean matches(in ImagePredicate pred);
   ....
}
```

**Wrapper Schema for Text Repository:**

```
interface Document(extent Document):
      persistent {
   attribute String text;
   Boolean matches(String search_expr);
   ....
}
```

**Figure 3. (a): Repository Wrapper Schemas**

```
interface ClaimFolder (extent ClaimFolder): persistent {
   relationship Claim claim;
   relationship Set<Image> pictures;
   relationship Document report;
}
```

**Figure 3. (b): Complex Object Repository Schema**

# 5 Queries in Garlic

Given the schema for a Garlic database that a user or application wishes to utilize, Garlic provides access to the database in the usual way, i.e., through the provision of a high-level query language. Since the data model of Garlic is object-oriented, and since SQL is the dominant query language today for database application and tool builders, the query language of Garlic is an object-oriented extension of SQL. In this section we describe the general flavor of the extensions, mentioning both the object syntax extensions and the semantic extensions needed to accommodate the sorts of queries that can arise when querying multimedia types such as text or images. We then provide an overview of the Garlic query processing problem and approach and some of the implied research and architectural challenges.

## 5.1 The Garlic Query Language

To accommodate the object-oriented nature of the Garlic data model, Garlic extends SQL with additional constructs for traversing paths composed of inter-object relationships, for querying collection-valued attributes of objects, and for invoking methods within queries. These object-oriented SQL extensions are important, but are not particularly startling; they are essentially the same as the extensions provided in various other recent object query language proposals (e.g., [26], [27], [28], [29], [30], [31]), including the ongoing efforts of the SQL-3 committee [32], [33]. While we have adopted the ODMG-93 data model as a starting point for the Garlic data model, we have chosen not to adopt the OQL query language proposed by the

```
interface Policy (extent Policy): persistent {
    attribute Unsigned Long pol_no;
    attribute String cust_name;
    attribute String cust_address;
    attribute Date start_date;
    relationship Set<Claim> claims inverse Claim::policy;
  }
interface Claim (extent Claim): persistent {
    relationship Policy policy inverse Policy::claims;
    attribute Unsigned Long claim_no;
    attribute Date date_filed;
    attribute String handler;
    attribute String status;
    relationship Set<Image> pictures inverse Image::claim;
    relationship Document report inverse Document::claim;
  }
interface ImagePredicate: transient {
    void makeImagePredicate();
    void makeImagePredicate(in String);
    ....
  }
interface Image (extent Image): persistent {
    relationship Claim claim inverse Claim::pictures;
    void display();
    Boolean matches(in ImagePredicate pred);
    ....
  }
interface Document (extent Document): persistent {
    attribute String text;
    relationship Claim claim inverse Claim::report;
    Boolean matches(String search_expr);
    ....
  }
```

**Figure 4. Global Garlic Schema**

ODMG-93 consortium. There are currently too many ways in which OQL is unnecessarily different than SQL [34], and SQL (for better or for worse) is much too widely used to be ignored at this point. Instead, we are incorporating many of the nice ideas embodied in the ODMG-93 OQL language design into Garlic's object-oriented SQL extensions. The following query, written against the Garlic database schema of Figure 4, illustrates the flavor of the extensions:[5]

**select** P.cust_name, {C.pictures}
**from** Policy P, P.claims C, C.pictures X
**where** (C.date_filed - P.start_date) < 60 **and**
C.report.matches("fatality speeding") **and**
X.matches(["color", RED, .15])

This query could be used to find the customers who have a claim against their policy for an accident involving a red car, speeding, and a fatality, where the accident occurred within 60 days of the customer taking out the policy (perhaps to substantiate the hypothesis that red cars tend to get into lethal accidents as soon

5. We are still working out the exact details of our SQL extensions. This example is provided to give the reader a feeling for what we intend, and should not be taken too literally!

as they're insured). It also retrieves the pictures of the damage for each such claim. The query illustrates several of Garlic's object-oriented SQL extensions. First, it contains a number of path expressions involving the traversal of the claims relationship of Policy objects. Second, it contains invocations of the matches( ) methods associated with Document and Image objects. These methods pass a predicate specification to the appropriate repository for evaluation. In our example, the text repository is passed a simple vector of terms to match. The image repository is passed a structure indicating the type of match needed (color, in this case), the target for the match (we assume RED is a constant giving appropriate numeric values for some red color, which may have been chosen from a user interface widget such as a color wheel), and a number between 0 and 1 indicating the desired closeness of the match (roughly, the maximum distance from the picture's color to the target). Finally, the treatment of the pictures attribute of Claim objects in the query illustrates how set-valued attributes can be made to appear in their unnormalized form in query results.

Since the Garlic query language is intended for querying databases that contain data in a variety of repositories, including multimedia repositories with associative search capabilities, Garlic's SQL extensions must also take the needs of such repositories into account. Many of their needs can be accommodated simply through the use of the query language's object extensions, e.g., by making use of methods in query predicates and target lists, as illustrated above. However, the search facilities provided by repositories that manage multimedia data types such as text and images are often based on a somewhat different query model than the one used in most database systems -- rather than requesting the retrieval of every data item where a given predicate is true, text and image queries commonly request the ordered retrieval of the top N (or all) data items that come "close" (perhaps within a certain threshold) to matching a given predicate. An image-oriented example that could be satisfied using the search capabilities of the QBIC system [1] would be "find all reddish images that contain a yellowish, circle-like object in the middle". QBIC would respond to this query by returning a list of all images that approximately match the specified predicate, and would rank-order the results by their closeness to the query predicate, as measured by an appropriate similarity function.

Integrating approximate-matching query semantics with more traditional (exact-matching) database query semantics is an interesting problem, and we are currently developing a set of syntactic and semantic SQL extensions to support queries that involve both exact and approximate search criteria. This work involves introducing into SQL the notion of graded sets (the ordering of which can be modeled using lists in the Garlic data model). In such sets, each object is assigned a number between 0 and 1 for each atomic predicate; this number represents the degree to which the object fulfills the predicate, with 1 representing a perfect match. Boolean combinations of predicates can then be handled using the rules for combining predicates in fuzzy logic [35]. To enable query writers to specify the desired semantics, the syntax of SQL is extended to permit the specification of the number of matching results to be returned and whether or not rank-ordering (rather than an attribute-based sort order, or an arbitrary order) is desired for the query's result set.

## 5.2 Garlic Query Processing

As indicated in Figure 1 of Section 3, query processing in Garlic is the responsibility of the query services and runtime system component of the Garlic architecture. Since a Garlic database can span a number of data repositories, Garlic queries are also likely to span repositories. As a result, Garlic faces a query processing problem similar to that of heterogeneous distributed database systems and we will draw upon work in that area (e.g., [36], [39]) as well as the significant body of earlier work on distributed query processing (e.g., [37], [38]). Since Garlic queries are formulated in an object-extended SQL dialect, we will also draw upon

early results in the area of object query processing (e.g., [40], [41], [42]); this is still an active research area, so we also expect that the Garlic project will contribute to this area over time. In addition to the problems raised by coupling heterogeneous databases and supporting object queries, Garlic also faces significant query processing challenges that arise in accommodating a broad range of repository types and data types.

Given a query in Garlic SQL, we expect query processing to proceed roughly as follows. First, the query will be translated from Garlic SQL into an intermediate form suitable for manipulation by a query optimizer. This form will be based on extending the QGM query representation [50], which is used in Starburst and DB2 C/S, to accommodate Garlic's SQL extensions. We call Garlic's version of this representation OQGM. The query will then be handed to the Garlic "middleware" query optimizer, referred to as the Query Decomposition Facility (QDF), whose responsibility will be to turn the query into an execution plan that can be handed off to the Garlic runtime system for execution. Since the underlying data is stored in one or more repositories with associative search capabilities, this process involves decomposing the query into a plan containing a number of smaller queries, each runnable by an individual repository, as well as producing a subplan for combining the repositories' query results into the desired final form. The smaller queries that comprise this execution plan will also be expressed in OQGM, making it the job of each repository wrapper to convert its individual subplan -- containing one or more OQGM queries -- into an interpretable subplan whose queries are expressed in the repository's native query language (or its native search API, if it has no actual query language). The wrappers will then execute their subplans in a demand-driven fashion under the control of the Garlic runtime system, returning their results to the Garlic runtime system component for any additional processing needed before they are returned to the application.[6]

In order to accomplish its tasks, the Garlic QDF component will make use of several kinds of metadata that reside in the Garlic metadata repository. First of all, in order to successfully decompose the query, the QDF will utilize the database schema and the underlying wrapper schemas in order to rewrite the query, which may involve views, into a query that refers only to interfaces provided by wrapper schemas. It can then proceed to break the query into pieces that can be satisfied entirely within the scope of the individual wrapper schemas of the repositories whose data the query accesses. Secondly, in order to do a good job of query decomposition, producing an efficient execution plan, the QDF will have to utilize descriptions of the query processing power of each involved repository. The question of how to characterize the query power of a repository, in terms of the OQGM subset that its wrapper is capable of processing directly (i.e., without help from the Garlic runtime system), is an interesting question that the project is currently investigating. The QDF will also need to come up with cost and selectivity estimates in order to devise an efficient access plan. How to architect the system to make the necessary cost- and selectivity-related metadata available is another problem of interest to the project. Our initial prototype will likely side-step these important issues, striving only to produce a workable query plan of some sort for each query, but they are definitely issues that the project will address as it proceeds.

The result of each subquery, as well as that of the query as a whole, will be a collection of values or objects. The runtime system will utilize a stream-based approach to query processing that fits well with the planned demand-driven approach to query execution in Garlic. In terms of producing the final set of results, for tra-

6. In addition to the query path through the system, the Garlic C++ API will permit a C++ application program to invoke a method on an object for which the program has a reference. This can be viewed (and could even be processed) as a degenerate query, but a "fast path" through the query processing and runtime services component will be provided for such simple requests.

ditional database-like queries, the runtime system will simply have to join or otherwise combine the results from the various repositories in some order, making a suitable trade-off between performance and resource utilization while producing the final results. For queries involving approximate-matching, however, query processing becomes somewhat more complicated. In addition to the problems of performance and resource utilization, Garlic faces the interesting problem of producing the N best results efficiently (i.e., without materializing every intermediate result item that matches to any degree at all). This is another area where we expect interesting technology to result from our work.

# 6 Garlic Interfaces and Applications

Data in Garlic will be accessible through two primary interfaces. Application programs will be able to access data in the system via Garlic's C++ application programming interface. Within this C++ API, two major sub-interfaces will be provided -- one for compiled applications, written with *a priori* knowledge of the Garlic schema for the database of interest, and one for dynamic applications, written with no *a priori* knowledge of the types or objects that exist in the database. End users will be able to access data in the system via Garlic's query/browser interface, a dynamic (system-provided) Garlic application that will provide a friendly and intuitive means for users to query and browse the contents of any Garlic database. We briefly describe the nature of each of these interfaces here.

## 6.1 The Garlic C++ API

As just mentioned, the Garlic C++ API will be designed to support both compiled and dynamic C++ applications. For compiled applications, the C++ API will provide access to the data in a pre-selected Garlic database via a C++ class library that contains C++ "proxy" classes that correspond to each of the interfaces and other types defined in the schema of the database. To use this interface, a C++ programmer will invoke a tool to extract a set of C++ *.h* files corresponding to the schema of the database (and also to various system-defined template classes for collections, iterators, etc.). The application programmer will then be able to write a Garlic application using the classes and corresponding methods defined in the resulting C++ class library. This library will support both navigational (i.e., pointer dereferencing) and query-based access to the contents of the database; updates will be supported as well. It should be noted that the desired level of functionality is very similar to that proposed in the C++ chapter of the ODMG-93 standard [21], and we plan to adhere to that proposed standard to the maximum extent possible as we work out all of the details of the compiled C++ API for Garlic. Some differences are likely, however, due to Garlic's goal of permitting multiple implementations of an interface.

For dynamic applications, the C++ API will provide access to the contents of any Garlic database through the provision of (i) a set of Garlic interface definitions that provide access to the metadata (i.e., schema) of the database and (ii) a set of generic C++ functions and classes that support metadata-driven navigation and querying of the data residing in a Garlic database. To support metadata access, every Garlic database will contain a set of predefined GDL object types (interfaces) that are instantiated with object instances that describe the schema of the database. For example, an instance of the Garlic type *GInterfaceDef* would describe the user-defined interface named *Policy* for the insurance example discussed earlier. Its *hasAttributes* attribute would contain references to objects that describe the attributes of a *Policy* (e.g., the *cust_name* and *cust_address* attributes), and its *hasSubtypes* attribute would contain references to any interfaces derived

from *Policy* (e.g., *HighRiskPolicy*, if our example database were extended to include such policies). Access to these metadata objects will be provided via the static C++ API, as their GDL interfaces are known *a priori* (since they are the same for all Garlic databases). In addition, the dynamic C++ API will provide facilities that allow the regular (i.e., non-metadata) objects in a Garlic database to be accessed and manipulated dynamically. These facilities will support the runtime formation and execution of requests for manipulating Garlic objects, providing access to their attributes and relationships and the ability to formulate and execute method invocation requests for objects whose types only become known to the C++ program through runtime metadata accesses. The Garlic query/browser, described next, will be the first client of this Garlic API.

## 6.2 The Garlic Query/Browser

The purpose of the Garlic query/browser component is to provide end users of the system with an easy and highly visual way to access and manipulate the data in a Garlic database. As its name implies, the query/browser will provide support for two basic data access functions, namely querying and browsing. However, unlike existing interfaces to databases, the distinction between these two functions will be rather blurry -- the query/browser will allow users to move back and forth seamlessly between querying and browsing activities, using queries to identify interesting subsets of the database, browsing the subset, querying the contents of a set-valued attribute of a particularly interesting object in the subset, and so on.

The query/browser will support the exploration of a Garlic database by allowing users to browse through the contents of Garlic collections (via next/previous buttons or scrolling) and to traverse relationships by clicking on (selecting) objects' reference attributes. When multiple related objects are being simultaneously displayed, synchronous browsing will be implied (*a la* [43], [44]). For example, if a user is browsing through the contents of the *Policy* type extent, and then clicks on the *claims* attribute of the currently displayed *Policy* object, the query/browser will open a second window to allow the user to browse through the set of *Claim* objects related to the current *Policy* object; advancing to the next *Policy* object will advance the browse state to the next set of *Claim* objects as well. By clicking on reference attributes, the user is essentially able to arrange for the display of a subgraph of the object graph formed by the inter-object references in the Garlic schema, and synchronous browsing ensures that the user's display will always contain a consistent view of the subgraph of interest. Also, within this subgraph, the user will be able to indicate (by clicking) which attributes and relationships he or she wishes to see (or not see) on the current display. The browser will also support the browsing of multimedia objects and object attributes, allowing the user to request full-resolution displays of images, playbacks of video or audio clips, and so on.

As mentioned above, the query/browser will allow users to move seamlessly between querying and browsing to explore the contents of a Garlic database. While the query power of the query/browser will be bounded by the expressiveness of the underlying Garlic extension of SQL, users will not formulate their queries in this language. Instead, the query/browser will support querying via a "query-by-graphical-example" paradigm, extending the well-known query-by-example paradigm [45] for use in formulating queries over an object database. Returning to the browsing example of a moment ago, suppose that the user now wishes to find policies with outstanding claims made before 7/1/94 for customers named "Smith". To do so, the user could enter query mode (by clicking on the display's "query" selector), type "* Smith" into the *cust_name* attribute in the *Policy* window, type "< 7/1/94" into the *date_filed* attribute in the related *Claim* window, type "unresolved" into the *status* field of the *Claim* window, and then click on the "run query" button (which will have been created on the display when query mode was entered). The query/browser will respond by

filling in the *Policy* window's attributes with information about the first "Smith"-owned policy with such a claim and the *Claim* window's attributes with information about the first such claim associated with that policy. The user can then browse the results, with the query's constraints remaining active until explicitly cleared, in the same manner that the entire set of policies and related claims could be browsed in the example above. An interesting question, currently under investigation, is how much of the power of the Garlic query language the query/browser will be able to make available without sacrificing ease of use and intuitive semantics.

In addition to smoothly combining querying and browsing, the Garlic query/browser will also provide other useful features for exploring and manipulating the contents of a heterogeneous multimedia data collection. First, the objects on the display at any given time will be active objects -- the query/browser will remember their Garlic identities and will provide a graphical means of obtaining a list of their available methods and requesting that one of the methods be applied to the object of interest (prompting for method arguments if needed). Second, clicking on "query" followed by a multimedia (e.g., image, audio, video, or text) attribute of a displayed object will result in the display of a type-specific picker (or set of pickers) to support the construction of a media-specific predicate on that attribute of the object. The query/browser will contain a number of such pickers to support the graphical specification of content-based multimedia predicates. For image selection, for example, pickers along the lines of those that the QBIC user interface provides for specifying color, texture, shape, and other features for content-based image retrieval [1] will be provided in the query/browser. Finally, in time, the query/browser will become still more sophisticated, supporting the graphical definition of end-user views (thereby saving application developers from having to define views in the Garlic query language, and hopefully enabling end users to develop their own views). Ultimately, we believe that good support for customizing the browser's behavior with respect to a given Garlic database and Garlic user may lead to a new paradigm for visual application development, at least for applications of a "browsy" (i.e., navigational) nature.

# 7 Conclusions, Status and Future Work

We have presented an overview of the Garlic project at the IBM Almaden Research Center, the goal of which is to build a heterogeneous multimedia information system (MMIS) capable of integrating data from a variety of traditional and non-traditional data repositories. We described the overall architecture for the system, which is based on repositories, repository wrappers, and the use of an object-oriented data model and query language to provide a uniform view of the disparate data types and data sources that can contribute data to a Garlic database. As we explained, a significant focus of the project is the provision of support for repositories that provide media-specific indexing and query capabilities; this involves research challenges in the areas of middleware query processing and system extensibility. We also described the planned interfaces to Garlic, including support for compiled C++ applications, dynamic C++ applications, and exploratory access by end users via the Garlic query/browser.

The current status of the Garlic effort, which was initiated in early 1994, is that most of the high-level design work has been completed. Details of the various components are being worked out, and are in various stages of completion as of this writing. Our current target is to have an initial "proof of concept" prototype -- involving all of the components described here -- running (or at least limping) by the end of 1994. To speed development, we plan to use the ObjectStore object-oriented DBMS [46] to hold both the complex objects

and the metadata objects in the prototype. The initial prototype will be demonstrated by developing a simple application involving data that spans a relational DBMS (DB2 C/S), an image repository (based on QBIC), and a text repository (possibly IBM's Search Manager product). The data will be information about multimedia systems (a slightly incestuous example!) used by an imaginary industry consulting firm, the "Garlic Group", to produce reports on the state of the multimedia systems industry. The initial query/browser prototype will be the front end for this application, providing exploratory access to information about multimedia software and hardware vendors and their products. The main goal of the first prototype and application is to develop the system to the point where everything is at least starting to work, thereby providing clearer insights regarding the nature of wrappers, the challenges involved in query translation and processing, and the efficacy of the query/browser as an end-user window into a collection of multimedia data.

In the longer term, we expect the Garlic project to lead us into new research in many dimensions, including object-oriented and middleware query processing technologies, extensibility for highly heterogeneous, data-intensive environments, database user interfaces and application development approaches, and integration of exact- and approximate-matching semantics for multimedia query languages. There are also many interesting, type-specific issues, such as what predicates should be supported on image and video data, how to index multimedia information, how to support similarity-based search and relevance feedback, and what the appropriate user interfaces are for querying particular media types. We believe that significant challenges exist in each of these areas, and that solutions must be found if the field is going to move beyond the traditional boundaries of database systems and keep pace with the emerging demand for large-scale multimedia data management.

# 8 Acknowledgments

We would like to thank Rakesh Agrawal for his input in the start-up phase of the Garlic project; he contributed significantly to our vision for both the project as a whole and the query/browser in particular. John McPherson and Ashok Chandra have been particularly supportive of our efforts throughout; we thank them for their encouragement and many suggestions. Many others contributed to the definition of the Garlic project, including: Kurt Shoens, K.C. Lee, Jerry Kiernan, Peter Yanker, Harpreet Sawhney, David Steele, Byron Dom, Denis Lee and Markus Tresch.

# 9 References

[1] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasman, D. Petkovic, and P. Yanker: "The QBIC Project: Querying Images by Content Using Color, Texture and Shape", Proc. SPIE, San Jose, CA, February 1993.

[2] A. Pentland, R. Pickard, and S. Scarloff, MIT Media Lab: "Photobook: Tools for Content Based Manipulation of Image Databases", Proc. SPIE, San Jose, CA, 1994.

[3] B. McCune, R. Tong, J. Dean, and D. Shapiro, "RUBRIC: A System for Rule-Based Information Retrieval", IEEE Trans. on Software Eng. 11(9), Sept. 1985.

[4] G. Salton, "Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer", Addison-Wesley Publishers, 1989.

[5] B. Kahle and A. Medlar, "An Information System for Corporate Users: Wide Area Information Servers", Tech. Rep. No. TMC-199, Thinking Machines Corp., Cambridge, MA, 1991.

[6] K. Shoens, A. Luniewsi, P. Schwarz, J. Stamos, and J. Thomas, "The Rufus System: Information Organization for Semi-Structured Data", Proc. VLDB Conference, Dublin, Ireland, 1993.

[7] PixTex/Electronic Filing System (EFS) Reference Manual, Excalibur Technologies Corporation, San Diego, CA, 1993.

[8] PixTex/Visual Retrieval System (VRS) Reference Manual, Excalibur Technologies Corporation, San Diego, CA.

[9] A. Laursen, J. Olkin, and M. Porter, "Oracle Media Server: Providing Consumer Based Interactive Access to Multimedia Data", Proc. ACM SIGMOD Conference, Minneapolis, MN, May 1994.

[10] R. Haskin, "The Shark Continuous-Media File Server", Proc. IEEE COMPCON Conference, San Francisco, CA, 1993.

[11] R. Rosenberg and T. Landers, "An Overview of MULTIBASE", in Distributed Databases, H. Schneider, ed., North-Holland Publishers, New York, NY, 1982.

[12] A. Elmagarmid and C. Pu, eds., Special Issue on Heterogeneous Databases, ACM Comp. Surveys 22(3), September 1990.

[13] W. Kim and J. Seo, "Classifying Schematic and Data Heterogeneity in Multidatabase Systems", IEEE Computer, December 1991.

[14] D. Hsiao, "Federated Databases and Systems: Part I -- A Tutorial on Their Data Sharing", VLDB Journal 1(1), July 1992.

[15] Y. Brietbart, H. Garcia-Molina, and A. Silberschatz, "Overview of Multidatabase Transaction Management", VLDB Journal 1(2), October 1992.

[16] T. Conners, W. Hasan, C. Kolovson, M. Neimat, D. Schneider, and K. Wilkinson, "The Papyrus Integrated Data Server", Proc. 1st PDIS Conference, Miami Beach, FL, December 1991.

[17] M. Shan, "Pegasus Architecture and Design Principles", Proc. 1993 ACM SIGMOD Conference, Washington, DC, May 1993.

[18] D. Fang, S. Ghandeharizadeh, D. McLeod, and A. Si, "The Design, Implementation, and Evaluation of an Object-Based Sharing Mechanism for Federated Database Systems", Proc. IEEE Conf. on Data Eng., Vienna, Austria, April 1993.

[19] W. Grosky, "Multimedia Information Systems", IEEE Multimedia 1(1), Spring 1994.

[20] M. Ubell, "The Montage Extensible Datablade Architecture", Proc. ACM SIGMOD Conference, Minneapolis, MN, May 1994.

[21] R. Cattell, ed., "The Object Database Standard: ODMG-93 (Release 1.1)", Morgan Kaufmann Publishers, San Francisco, CA, 1994.

[22] A. Black, N. Hutchenson, E. Jul, and H. Levy, "Object Structure in the Emerald System", Proc. ACM OOPSLA Conference, September 1986.

[23] S. Abiteboul and A. Bonner, "Objects and Views", Proc. ACM SIGMOD Conference, Denver, CO, May 1991.

[24] E. Rundensteiner, "MultiView: A Methodology for Supporting Multiple Views in Object-Oriented Databases", Proc. VLDB Conference, Vancouver, Canada, 1992.

[25] R. Agrawal and L. DeMichiel, "Defining View Types Using Project Operations", Proc. EDBT Conference, Cambridge, England, March 1994.

[26] M. Carey, D. DeWitt, and S. Vandenberg, "A Data Model and Query Language for EXODUS", Proc. ACM SIGMOD Conference, Chicago, IL, June 1988.

[27] D. Beech, "A Foundation for Evolution from Relational to Object Databases", Proc. EDBT Conference, March 1988.

[28] F. Bancilhon, S. Cluet, and C. Delobel, "A Query Language for the O2 Object-Oriented Database System", Proc. DBPL Conference, Salishan Lodge, Oregon, June 1989.

[29] W. Kim, "A Model of Queries for Object-Oriented Databases", Proc. VLDB Conference, Amsterdam, the Netherlands, August 1989.

[30] M. Kifer, W. Kim, and Y. Sagiv, "Querying Object-Oriented Databases", Proc. ACM SIGMOD Conference, San Diego, CA, 1992.

[31] S. Dar, N. Gehani, and H. Jagadish, "CQL++: A SQL for a C++ Based Object-Oriented DBMS", Proc. EDBT Conference, Vienna, Austria, 1992.

[32] L. Gallagher, "Object SQL: Language Extensions for Object Data Management", Proc. ISSM CIKM Conference, 1992.

[33] K. Kulkarni, "Object-Oriented Extensions in SQL3: A Status Report", Proc. ACM SIGMOD Conf, Minneapolis, MN, May 1994.

[34] W. Kim, "Observations on the ODMG-93 Proposal", ACM SIGMOD Record 23(1), March 1994.

[35] H. J. Zimmermann, Fuzzy Set Theory and its Applications, Kluwer Academic Publishers, Boston, MA, 1990.

[36] W. Du, R. Krishnamurthy, and M. Shan, "Query Optimization in Heterogeneous DBMS", Proc. VLDB Conference, Vancouver, Canada, 1992.

[37] C. Yu and C. Chang, "Distributed Query Processing", ACM Comp. Surveys, December 1984.

[38] G. Lohman, C. Mohan, L. Haas, B. Lindsay, P. Selinger, P. Wilms, and D. Daniels, "Query Processing in R*", in Query Processing in Database Systems, W. Kim, D. Batory, and D. Reiner, eds., Springer-Verlag Publishers, Heidelberg, Germany, 1985.

[39] U.Dayal, "Query Processing in a Multidatabase System", in Query Processing in Database Systems, W. Kim, D. Batory, and D. Reiner, eds., Springer-Verlag Publishers, Heidelberg, Germany, 1985.

[40] S. Cluet and C. Delobel, "A General Framework for the Optimization of Object-Oriented Queries", Proc. ACM SIGMOD Conference, San Diego, CA, June 1992.

[41] J. Orenstein, S. Haradhvala, B. Marguiles, and D. Sakahara, "Query Processing in the ObjectStore Database System", Proc. ACM SIGMOD Conference, San Diego, CA, 1992.

[42] J. Blakely, W. McKenna, and G. Graefe, "Experiences Building the OODB Query Optimizer", Proc. ACM SIGMOD Conference, Washington, DC, May 1993.

[43] A. Motro, A. D'Atri, and L. Tarantino, "The Design of KIVIEW: An Object-Oriented Browser", Proc. 2nd Int'l. Expert Database Systems Conference, Tysons Corner, VA, April 1988.

[44] R. Agrawal, N. Gehani, And J. Srinivasan, "OdeView: The Graphical Interface to Ode", Proc. ACM SIGMOD Conference, Atlantic City, NJ, May 1990.

[45] M. Zloof, "Query-By-Example: A Data Base Language", IBM Systems Journal 16(4), 1977.

[46] C. Lamb, G. Landis, J. Orenstein, and D. Weinreb, "The ObjectStore Database System", Comm. ACM 34(10), October 1991.

[47] Understanding GIS -- The ARC/INFO Method, ESRI Inc. (1990).

[48] O. Guenther, "Efficient Structures for Geometric Data Management", Lecture Notes in Computer Science, Springer-Verlag 337 (1988).

[49] "SPANS: SPatial ANalysis System", TYDAC Technologies: Corporate Overview (1990).

[50] H. Pirahesh, J. Hellerstein, and W. Hasan, "Extensible/Rule Based Query Rewrite Optimization in Starburst", Proc. ACM SIGMOD Conference, San Diego, CA, June 1992.