Object-Oriented Heterogeneous Database for Materials Science

DAVID HANSEN, DAVID MAIER, JAMES STANLEY, and JONATHAN WALPOLE
The Oregon Graduate Institute of Science & Technology, Beaverton, OR 97006

ABSTRACT

As a part of the scientific database research underway at the Oregon Graduate Institute, we are collaborating with materials scientists in the research and development of an extensible modeling and computation environment for materials science. Materials scientists are prolific users of computers for scientific research. Modeling techniques and algorithms are well known and refined, and computerized databases of chemical and physical property data abound. However, applications are typically developed in isolation, using information models specifically tailored for the needs of each application. Furthermore, available computerized databases in the form of CDs and on-line information services are still accessed manually by the scientist in an off-line fashion. Thus researchers are repeatedly constructing and populating new custom databases for each application. The goal of our research is to bridge this gulf between applications and sources of data. We believe that object-oriented technology in general, and databases in particular, provide powerful tools for transparently bridging the gap between programs and data. An object-oriented database that not only manages data generated by user applications, but also provides access to relevant external data sources can be used to bridge this gap. An object-oriented database for materials science data is described that brings together data from heterogeneous non-object-oriented sources and formats, and presents the user with a single, uniform object-oriented schema that transparently integrates these diverse databases. A unique multilevel architecture is presented that provides a mechanism for efficiently accessing both heterogeneous external data sources and new data stored within the database. © 1993 John Wiley & Sons, Inc.

1 INTRODUCTION

The Scientific Database Group at the Oregon Graduate Institute of Science & Technology (OGI) is currently involved in a collaborative research effort with materials scientists at OGI to develop a comprehensive computational environment for materials science research.

Broadly stated, we are interested in exploring data models and database support for scientific domains. Specifically we are interested in:

1. Exploring data models that are capable of providing support for complex scientific data types
2. Examining database architectures based on these data models that are capable of integrating large scientific databases by provid-

Received October 1992
Revised November 1992

Scientific Programming, Vol. 1, pp. 115-131 (1992)
CCC 1058-9244/93/020115-17

ing transparent access to heterogeneous data sources
3. Using these data models to develop flexible and extensible interfaces between application programs and applicable sources of data

Currently, each application program typically has its own data model tailored to that application. Custom databases built using one program's data model are rarely useful to other applications [1]. Sharing data between applications is difficult and usually requires a user to convert data between the different file formats used by different programs [2].

2 DATABASE MANAGEMENT SYSTEMS

A database management system (DBMS) can be used to decouple data from the applications that use the data, allowing the data to be more easily shared by multiple applications. The interface to a DBMS has two parts, a data definition language and a data manipulation language. The data definition language is used to describe the data in the database. This description, or schema, defines a logical structure for the data as seen by clients of the database. (A client may be a person or an application program.) The data manipulation language is used by clients to access and manipulate the data in the database. Together, the schema constructed by the data definition language along with the data manipulation language provides an application-independent interface between data and applications. Applications access the database using the schema and are insulated from the physical layout of the data within the database. This insulation is called program-data independence and is an important benefit of using a DBMS.

A DBMS provides additional benefits such as managing concurrent access to the data in the database, protecting data through backup and recovery processes, controlling redundancy, checking the validity of data entered into the database, etc. [3].

The data definition language and the data manipulation language define the type of data model implemented by the DBMS. For a DBMS based on the relational data model, the data definition language will include constructs for describing relational tables. The data manipulation language (typically SQL) will provide the capability to insert, update, delete, and select data from the tables in the database.

A DBMS based on an object-oriented data model will use a data definition language that is capable of defining classes of objects. A class defines an abstract data type encapsulating attributes (i.e., data values) and methods (i.e., program codes) together in objects of that class. Classes are typically arranged in an inheritance hierarchy where subclasses "inherit" the structure and behavior of their superclasses.

The data manipulation language provides general query capabilities for selecting objects based on the value of its attributes as well as on the return value of a method invocation. (An object executes a method in response to a message [i.e., a procedure call] sent to the object.) The data manipulation language may also be used for writing the methods that are encapsulated in the class definition.

2.1 Benefits of the Object-Oriented Model

An object-oriented data model provides much richer data definition and manipulation languages than other data models such as the relational model. Where the relational model has only a single "data structure," the table, an object-oriented data model provides the ability to define an unlimited number of different data structures and types to support virtually any sort of data.

The object-oriented paradigm provides powerful tools for constructing complex data types common in scientific domains. Consider the task of storing matrices in a database. In a relation database, a matrix can only be represented by decomposing the matrix into a relational table that contains the data from the matrix. Operations over matrices such as multiplication and dot product are not supported by relational data manipulation languages. Thus an application using the database must retrieve the data and reconstruct the matrix using a representation defined by the data types within the application.

In an object-oriented database, the matrix could be represented intuitively as a two-dimensional array of rows and columns. The database and application programs can share a common representation for matrices. Furthermore, because the definition of a class in the database also includes the specification of the behavior of the objects in the class, the object-oriented database representation of a matrix can include methods for manipulating and testing matrices.

Packaging methods together with data as objects in an object-oriented database provides two important benefits. First, the development and maintenance of applications is simplified because a common code can be developed once and stored as part of the definition of a class. Second, these methods can be used as a part of any query over the data in the database. Thus the methods incrementally extend the query language by adding new functionality to the objects in the database. For example, a method called rank for objects of the matrix class could be used to compute and return the number of linearly independent rows of a matrix. A query could select fully independent matrices by searching for matrices where the rank of the matrix was equal to the number of rows in the matrix. In a relational implementation, this sort of query is not expressible. Instead, an application program would have to retrieve all matrices from the relational database, compute the rank of each matrix, and then compare the rank to the number of rows to determine which matrices were fully independent.

2.2 Currently Available Scientific Databases

Although an object-oriented database can be used to design and implement a database of scientific data, many databases containing relevant chemical and physical property data already exist. These general-purpose databases, in the form of CD-ROMs or on-line services, are designed to be accessed by interactive information retrieval software, not directly by user application programs. The interactive information retrieval software allows the user to pose queries and receive textual answers to those queries. The data of interest are then manually transferred by users into a local database for application.

The goal of our current research is to design and implement an object-oriented database for materials science that managers data stored locally within the database, as well as data external to the database. Our object-oriented database provides transparent access to the data managed by the database regardless of the format of the data or where that data physically reside as shown in Figure 1. The interface and data model are application independent, describing the data and data types in the domain terms of materials science rather than in the format of a particular application.

2.2.1 An Object-Oriented Data Model for Materials Science

We have chosen to use materials science as a test domain for our research. Materials scientists have been leaders in the use of computers for modeling and research, and computational models for materials science are well known and refined [2]. In addition, there are many computer-readable databases available for materials science [1, 4–7].

We have developed an application-independent object-oriented data model for materials sci-

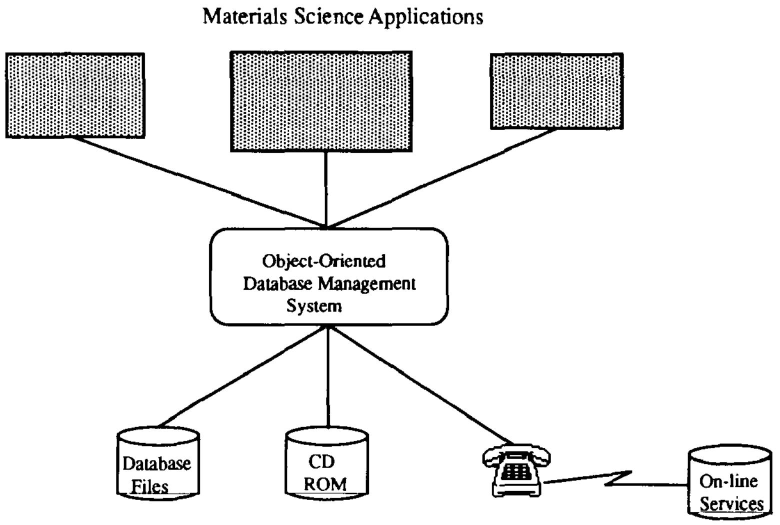
Materials Science Applications
Object-Oriented Database Management System
Database Files
CD ROM
On-line Services

FIGURE 1 A single programmatic interface between programs and sources of data.

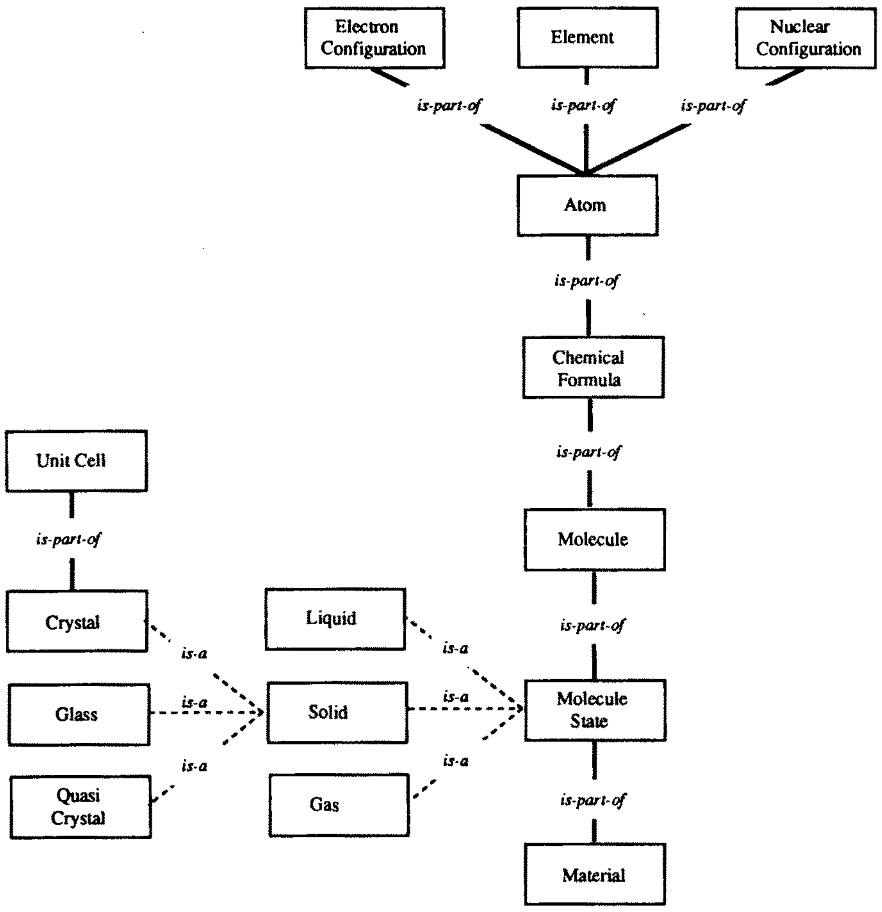

FIGURE 2 Object-oriented data model for materials science.

ence data. As shown in Figure 2, our data model currently supports a subdomain of materials science, known as structural analysis or crystallography [8]. Our data model is being incrementally defined as we seek to accommodate new areas of computational materials science.

The remainder of this paper describes the design and implementation of an object-oriented database that implements our data model and provides transparent access to data stored in external data sources.

3 HETEROGENEOUS DATABASES

The goal of providing access to multiple, diverse databases through a single interface has been a topic of research for over a decade and has become increasingly important due to the proliferation of incompatible databases and data available in a computer-readable format.

A heterogeneous database (HDB) is a mechanism for providing a single point of access to multiple databases, each of which may have a different data model and DBMS software. An HDB provides a single, common data manipulation language with which to manipulate the data it manages [9–11].

There are a number of reasons for building an HDB on top of existing component databases rather than attempting to homogenize them by incorporating them into a single monolithic database. One reason is that the component databases may be marginally related to one another. The databases may have very different implementations, each having been designed to meet the needs of its primary users. Furthermore component databases are often "owned" by different organizations that wish to retain control of their database. Another reason for integration over centralized homogenization is that the component databases may be geographically separated to promote efficient access by the local, primary users of the data.

In a scientific environment such as materials science, not only are databases of scientific property data not owned by the scientist, they are also quite large, some consisting of hundreds of megabytes of data. However, a researcher is likely to be interested in only a very small subset of the database at any one time. Thus the creation of a single, monolithic database is not a particularly practical or efficient solution, and may be impossible due to the sheer volume of the data sources.

3.1 Two Schools of Thought

There are currently two schools of thought on the development of HDBs, the integrated (federated) approach with a global unifying schema, as typified by Multibase [12], and the interoperable approach of integrating multiple databases without a globally integrated schema, as typified by the Multics Relational Data Store Multidatabase (MRDSM) [11]. The most significant difference between these two approaches is the degree of transparency provided.

3.1.1 Integrated HDBs

The integrated approach is the more ambitious of the two and attempts to maximize transparency by providing a single, unified schema for the users of the HDB. Component databases may be managed by other DBMSs, or may be nothing more than traditional files.

The architecture of Multibase, shown in Figure 3, is representative of the integrated HDB approach.

In Multibase, queries are posed against a global schema maintained by the Multibase software. Multibase translates the global query into subqueries over the local schemas of the component

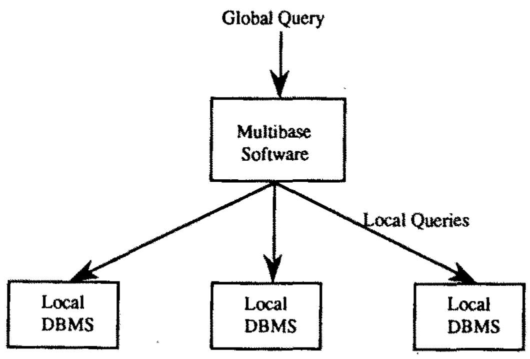

FIGURE 3 The Multibase architecture [12, p. 336].

databases. These local queries are then sent to the local DBMS for execution.

The major difficulty in developing an integrated HDB is in generating a unified, global schema and mapping the local, component schemas into this global schema.

Syntactic incompatibilities can include data type mismatches and conflicting names for the same data item in different databases. More difficult, however, are semantic incompatibilities that occur "when there is a disagreement about the meaning, interpretation, or intended use of the same or related data" [9, p. 187]. Multibase uses an integration database to resolve syntactic and semantic conflicts among its component databases [12, p. 339]. The integration database provides the information necessary to map component database schemas to the global schema.

3.1.2 Interoperable HDBs

Litwin and others [11, 13, 14] have noted that the task of creating a unifying global schema is difficult at best. Rather than develop a global schema, the interoperable approach focuses on providing access to the component databases while leaving the navigation of the component databases and the synthesis of data to the user. In contrast to the integrated approach, the interoperable approach assumes that the component databases are managed by some type of DBMS that is capable of executing queries posed by the user.

The MRDSM provides a data manipulation language, MDSL, for expressing queries and updates to data is separate databases. MRDSM also provides a measure of transparency by allowing for the definition of views in the relational tradition. These views can be used to transparently join data drawn from more than one database.

Litwin [15] has proposed that the relational model be adopted as a "standard (canonical) data model" for HDBs, thus providing a lowest common denominator among the component databases of an HDB.

3.2 The Current State of the Art

Most HDB implementations currently rely on relational technology. One reason for this relational bias is the adoption of SQL as a standard by most relational database vendors. This standardization provides a basic building block of HDBs using a relational database engine.

A number of commercial relational database management systems currently provide interoperable-style access to other vendor's databases through query translation mechanisms such as Ingres' gateways and the Open Server from Sybase [16].

In the last few years, a number of researchers have begun to suggest that the object-oriented model may hold promise for building HDBs [17–22]. The rich modeling constructs provided by an object-oriented database, along with the ability to encapsulate data (attributes) and programs (methods) together as objects in the database provide a powerful mechanism for building HDBs.

4 OBJECT-ORIENTED ARCHITECTURES FOR HDBs

The first issue to address in building an object-oriented HDB is to determine the granularity of "objectification," that is, what external entity will an object in an object-oriented database represent and encapsulate. We identify three levels of granularity:

1. Encapsulating databases where an object in the database encapsulates an entire external data source
2. Encapsulating collections where an object in the database encapsulates a homogeneous collection of external entities, possibly drawn from multiple data sources
3. Encapsulating single objects where each object in the database encapsulates a single entity drawn from one or more external data sources

4.1 Encapsulating Databases

Encapsulating databases as objects can be used to provide an interoperable HDB. Manola [17] has suggested such an approach: "The approach is basically to surround the DBMS with a layer of software that implements a common interface, using the object concept to encapsulate this software" (p. 129).

Queries are executed by sending messages to the database objects. Each database object performs the tasks of transforming the query into the native data manipulation language of the external database it encapsulates, passing the query to the native DBMS, and returning the result of the query. Figure 4 shows two encapsulated databases responding to the same "select" message. For the relational database, the message is simply passed along to the underlying database. The CODASYL database, however, must translate the query into an equivalent series of "find" messages.

However, encapsulating entire databases does not really exploit the power of the object-oriented paradigm. The only benefit is that all database objects are able to respond polymorphically to the same query, regardless of the underlying format of the external data source. Furthermore, although Manola [17] uses object-oriented technology to implement the HDB, the common query interface to the component databases remains relational (p. 129). Data retrieved from the encapsulated databases as the result of a query are presented to the user as relations. The data are not encapsulated as objects.

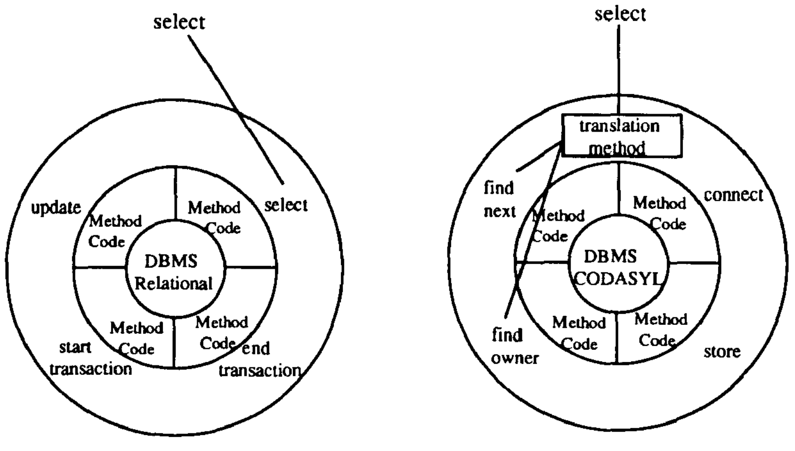

FIGURE 4 Encapsulated databases [17, p. 130].

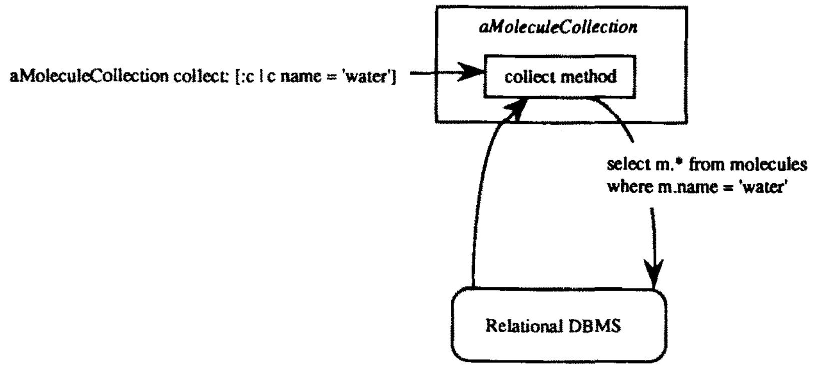
aMoleculeCollection
aMoleculeCollection collect: [:c | c name = 'water']
collect method
select m.* from molecules
where m.name = 'water'
Relational DBMS

FIGURE 5 Encapsulated collections.

4.1.1 Encapsulating Collections

Another solution is to encapsulate external data in collection objects (CO). Most object-oriented databases and languages have some notion of a collection class. Collections may be used to group objects together in a meaningful way. Collections need not be homogeneous, but a common use of collections is to provide a way of iterating over all the members of a class by creating a homogeneous collection containing those members.

Figure 5 depicts a CO of *Molecules* receiving an iteration message that "collects" objects with the name "water." In response to the iteration message, the CO, which stores no data itself, converts the query into the data manipulation language of an external data source and queries the external data source. A new collection of newly created *Molecule* objects matching the query selection criteria are returned.

This solution has an important advantage over encapsulating entire databases. By encapsulating homogeneous collections of entities, a single, consistent object-oriented definition is imposed on the data stored in the external data sources. So given a CO of objects of some class C, the CO can be queried based on the object-oriented structure and behavior of class C, regardless of the various formats of the entities in the external data sources.

Obviously a CO must be quite powerful. It must be able to translate queries over its members into queries over the external databases that contain relevant data, and cast the results of the external database queries into objects of the appropriate class.

Encapsulating classes of objects in this manner is very similar to the way other HDBs use relations. The difference is that instead of a global query translator, the CO does the work of translating queries and retrieving relevant data. This approach is similar to the object-oriented views described by Bertino [23]. Czejdo and Taylor [21] also describe an object-oriented integrated database that is built around encapsulated classes. Their approach is to use common superclasses to provide canonical "views" of data from different databases.

A disadvantage of this approach, however, is that interobject relationships, such as one object being a part of another object, are not maintained persistently and must be recreated each time a composite object is accessed.

4.1.2 Encapsulating Single Objects

The finest level of granularity is encapsulating an external entity as an individual object within the database. In this approach, an object in the database does not store any data, but is capable of responding to messages by retrieving data from external data sources. An encapsulating object "represents" an entity that is stored in external data sources. Tirri et al. [24] take this approach by using *federated objects*, which are objects composed of fragments drawn from various external data sources as shown in Figure 6.

The advantage of this approach is that the structural and behavioral definition of an object in the HDB inherently imposes an object-oriented interface on data drawn from external data sources. In addition, because each object in the database is now responsible for retrieving its data from external data sources, objects of different classes may be mixed together in heterogeneous collections.

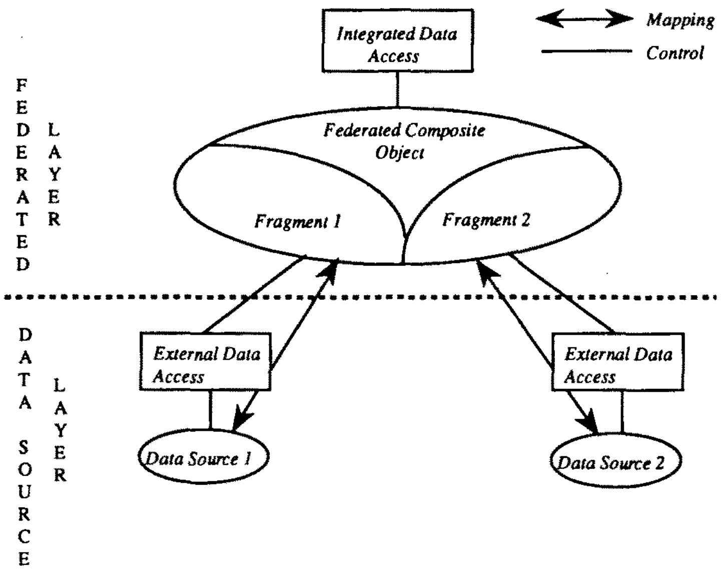

FIGURE 6 Federated complex objects [24, p. 295].

4.2 Our Approach

We have chosen to use an object-oriented database management system (OODBMS) to implement a HTB for materials science data. Our approach falls into the integrated HDB category with our object-oriented schema providing a single, unifying schema for diverse materials science data sources.

We have chosen the integrated approach for two reasons. First, we wish to provide a single object-oriented data model for materials science data. An integrated object-oriented schema is the only way to achieve this transparent integration of diverse external non-object-oriented data sources.

Second, the object-oriented paradigm and currently available commercial object-oriented databases provide mechanisms for addressing some of the syntactic and semantic heterogeneity problems that arise when integrating diverse data sources [22]. The most valuable mechanism for addressing these problems is the ability to encapsulate data together with methods for manipulating the object's data. This encapsulation provides an elegant way to programmatically handle problems due to heterogeneity by developing methods capable of resolving differences among different representations and providing a consistent external interface for the object. It should be noted that the ability to encapsulate data together with methods is not supported by all OODBMSs. We have chosen to use the GemStone OODMBS because it does provide the ability to store methods as well as data within the database. Many OODBMSs only provide storage for the data portion of an object; methods must be defined separately in the user applications that access the database.

Of the three object-oriented approaches above, our approach is similar to Tirri's federated object approach [24] where objects in the database encapsulate individual entities in external data sources. We call such an encapsulating object an object representative (OR).

Our approach differs from the approach taken by other researchers in a number of important ways.

First, in contrast to Tirri's federated object approach [24], our data sources are relatively static. Because the data sources change infrequently, we can initially scan each external data source and create an OR in the database for each entity of a particular class that is present in the external data source. This would be problematic in environments where external data sources were frequently being updated with new entities. However, large scientific databases of chemical and physical property data are slow to change so the need to rescan an updated database will be infrequent.

Second, as external data sources are scanned, ORs are added to the database, and static connections are made between related ORs. This allows complex objects to be constructed at scan time rather than attempting to relate objects during the execution of a query. For example, in adding a new OR for a *Molecule* object, that OR is connected to the *Atom* objects that are a part of the molecule's formula. This particular connection between a *Molecule* object and the Atom objects that form its formula provides an important performance enhancement for our database. Materials scientists often pose queries based on the chemical composition of a material. Consider a query to select all *Crystals* that contain copper and aluminum. The copper and aluminum *Atom* objects can be used to select related *Molecule* and *Crystal* objects by examining the static connections between a *Molecule* and its *Atoms*, and a *Crystal* and the *Molecule* on which it is based. Thus thousands of *Molecule* and *Crystal* objects may be searched without ever accessing external data sources. In contrast, encapsulating entire databases or collections of objects would require the query to be passed along to the external data sources for execution.

Third, the HDB will be used to integrate not only data from external data sources, but to store new objects created by applications that access the database directly. Thus our database is really a hybrid HDB, transparently managing both data stored within the HDB as well as external data sources.

5 AN OBJECT-ORIENTED HYBRID DATABASE

We have developed a hybrid database utilizing a three-level architecture with an external level consisting of the external data sources, a translational level providing an interface between ORs and the external data sources, and an object level that contains the ORs that populate the database (see Fig. 10). Both the object and translational levels reside within the database.

5.1 The External Level

The external level consists of the external data in its various native formats and software to access that data.

5.1.1 External Data Sources

We currently have three materials science data sources accessible via the database. The largest data source is the NBS Crystal Database distributed by the National Institutes of Standards and Technology [25]. This database is formatted as a single 128-megabyte ASCII file of structured data records and is distributed on CD-ROM. We have written a simple piece of data access software that is used to read a specified number of consecutive bytes beginning at a specified location in the file.

A second data source is composed of data files generated by a software tool called the Desktop Microscopist that was developed by Dr. James Stanley of OGI's Materials Science department. The Desktop Microscopist stores individual crystals, each in a single variable-length ASCII file.

The third data source is composed of data files generated by the Tektronix (TEK) Cache software for computer-aided chemistry. Like the Desktop Microscopist, the TEK Cache system stores individual molecules each in a single variable-length ASCII file.

Both the data files of the Desktop Microscopist and the TEK Cache system are self-describing and use a mixed format including both fixed-length fields and tabular fields similar to the tables of a relational database. The files from both systems are read in their entirety by the database when accessing the data contained in them.

Although these three data sources do not come with software for querying the data, we expect to be able to easily accommodate software-supported data sources such as databases managed by relational database management systems.

5.2 The Translational Level

Each OR in the object level has a corresponding object in the translational level of the architecture called a translational object (TO). Each TO contains two pieces of data:

1. Some unique key that allows us to access the record in the external data source containing it.
2. A data attribute that holds the data retrieved from the external data source.

For the NBS Crystal Database, the unique key used to access the data is a byte offset into the data file. This unique key allows us to access the data directly when required. Both the Desktop Microscopist and TEK Cache data are stored in single files so the name of the file provides the unique access key.

The primary function of a TO is to respond to messages from the OR requesting an attribute of

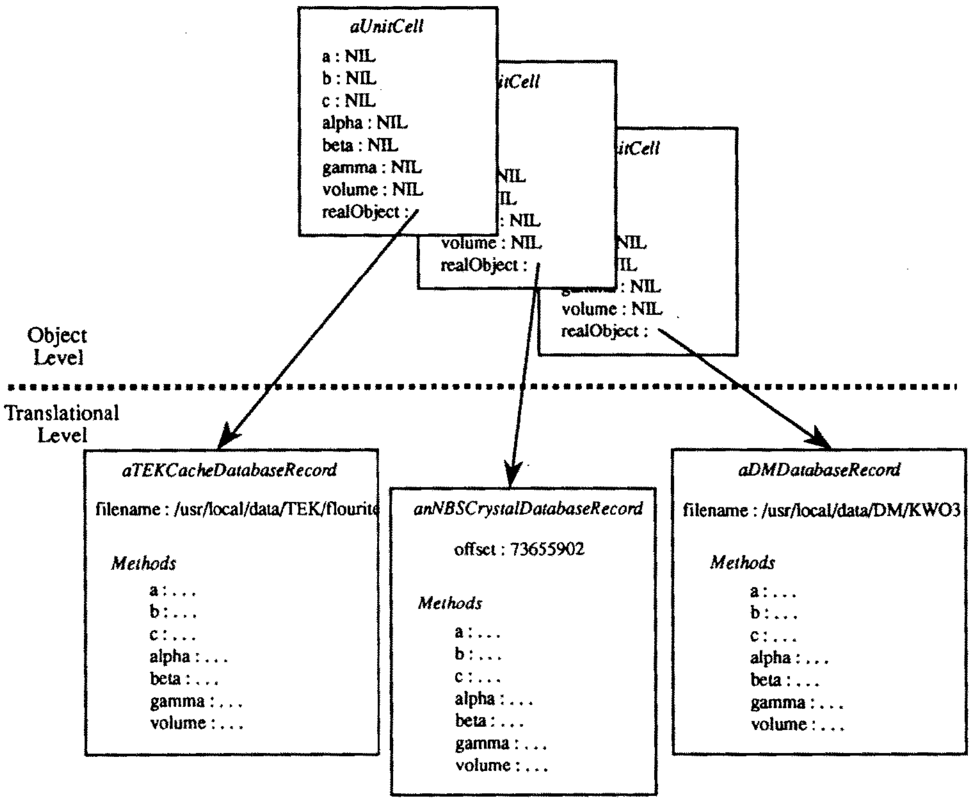

FIGURE 7 A Common interface between the Translational and Object levels.

the object by retrieving the data record from the external level, extracting the requested attribute from the formatted data, and performing any necessary datatype conversions.

5.2.1 Addressing Heterogeneity Problems

The translational level of the architecture provides a location for addressing syntactic and semantic problems that arise when attempting to integrate diverse data sources into a common data model.

As shown in Figure 7, different ORs of the same class (UnitCell in this case) may refer to TOs of different classes. However, the different TOs provide an identical interface to the ORs. The implementation of a particular method in a TO will depend on the format of the data in the external data source.

The interface between the object and translational levels of the architecture provides a mechanism to address syntactic as well as semantic heterogeneity problems. We are able to develop policies for addressing specific heterogeneity problems through the methods of the TOs. In the NBS Crystal Database, for example, some data items may have two values, one submitted by the author who researched the material, and a second added by the NBS Crystal Database editor. We have established a policy of using the value added by the editor if one is present, otherwise the value entered by the original author is used. Figure 8 shows how the unit cell volume method of the *NBSCrystalDatabaseRecord*, a TO, provides an external interface for the volume attribute by utilizing two internal methods, *authorsVolume* and *edVolume*, to decide what value to return to the OR at the object level.

5.3 The Object Level

It is the object level of the architecture that the users of the database see. It is the level where the unifying object-oriented schema presented in Figure 2 is implemented. Objects in the object level of the architecture may be ORs or native objects. An OR will refer to an object in the translational level and thus encapsulate data from an external data source. A native object is an object that is not derived from an external data source.

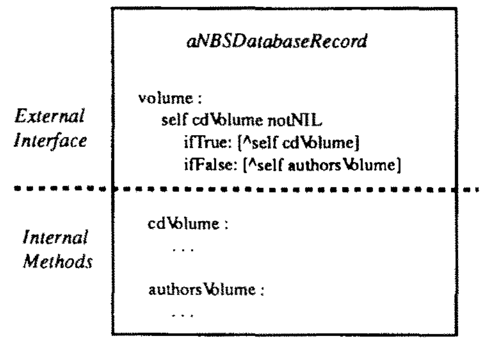

FIGURE 8 Handling semantic heterogeneity.

In order to provide a link between the object and translational levels of the architecture, a class has been defined from which all the object level classes are derived. This superclass contains a special attribute called the *realObject*. In an OR, the *realObject* attribute refers to the TO that encapsulates the source of the OR's data. In a native object, the *realObject* attribute is NIL. Figure 9 shows two *UnitCell* objects where one is an OR for unit cell data stored in the NBS Crystal Database and the other is a native object.

This demarcation between the object and translational levels of the architecture allows us to uniformly access data stored locally in the database (native objects) and data stored in external data sources. We are not aware of any other research providing this orthogonal view of data stored locally within the database and the external data accessible to the database.

5.3.1 Message Execution

Both native objects and ORs provide an identical behavioral interface. Figure 10 depicts an OR of the *Crystal* class responding to a *spaceGroup* message. If this *Crystal* object were a native object then the object would simply return the value of the *spaceGroup* attribute in response to the *spaceGroup* message. However, the crystal in Figure 10 is an OR and does not store any data locally. So the *spaceGroup* message is passed to the *NBSCrystalDatabaseRecord* object, a TO, in the translational level. The *NBSCrystalDatabaseRecord* object retrieves the data from the external data file, extracts the space group value from the formatted data, converts it to the proper data type (an integer in this case), and returns the value to the *Crystal* object, which in turn returns the value as the response to the original *spaceGroup* message it received.

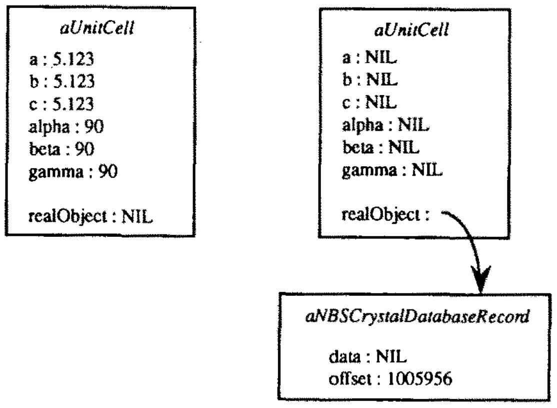

FIGURE 9 Unit cell objects.

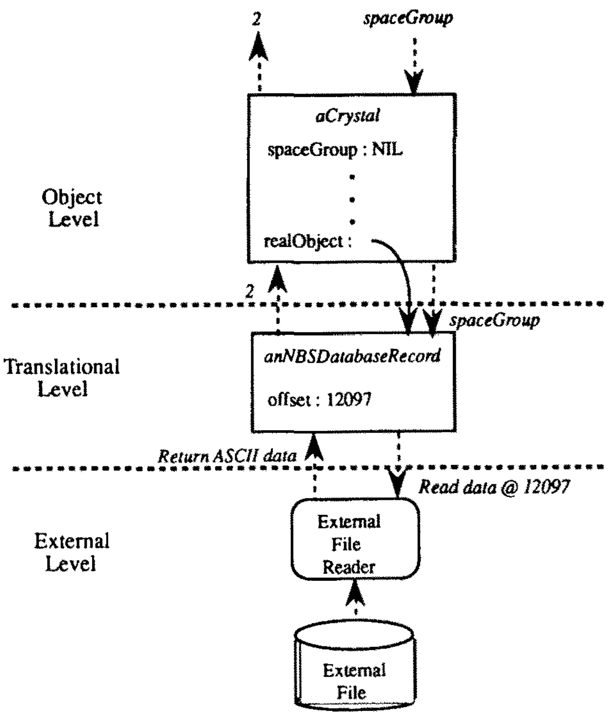

FIGURE 10 Message execution.

We expect each TO to be able to respond to any message sent by a corresponding OR. However, if a particular external data source does not contain the type of data requested, the TO accessing that data source simply returns a NIL value to the OR that requested the data.

5.4 Populating the Database

Because objects are created in the database that represent entities present in external data sources, an initial pass over the external data sources must be made to construct the ORs. This is a time-consuming process, but it has an important benefit, which is that the structure of complex objects can provide a good deal of information about the external entity without having to access the external data source. A *Crystal* object, for example, is a complex object that includes a *Molecule* object. The *Molecule* object is also a complex object that includes a *ChemicalFormula* object. The *ChemicalFormula* object is a complex object containing one or more *Atom* objects. During the pass over the external data source, not only is a new OR of the *Crystal* class added for each crystal encountered, that *Crystal* object is also "hooked up" to the proper *Molecule* object. These static connections between the objects at the object level of the architecture allow queries over *Crystal* objects based on their chemical composition by navigating the database at the object level without accessing the external data source. In this example, retaining a small amount of data within the database in the *Atom* objects allows the user to efficiently query a much larger number of *Crystal* objects that are stored in external data sources. This ability to trade small amounts of space for greatly increased query performance can be used throughout the architecture by making wise choices about what data to store within the database and what data to leave in external data sources.

5.4.1 The Identity Crisis

Unfortunately, making connections between objects drawn from external data sources is not trivial. Object-oriented databases rely on what is called "identity" to uniquely identify individual objects. Each object receives a unique, immutable object identifier when the object is created. This reliance on a unique system-generated object identifier is in contrast to value-based systems, such as relational databases, where the value of a data item determines its unique identity.

There are two potential problems in combining an identity-based system with a more traditional value-based database. The first problem occurs when an object in an identity-based system needs to refer to an entity stored in a value-based system. The object must use the "key" value of the entity to refer to it because that is the only way to uniquely identify the entity. The problems arises when some element of that key changes. Eliassen and Karlsen [26] point out that the link between an identity-based database and value-based external data sources is only as strong as the guarantee that the keys in the latter will not change.

This problem is only a small concern because the external data sources are primarily scientific property databases that will change slowly at fixed intervals.

The second problem is more subtle. The preceding section describes "hooking up" a *Crystal* object with the appropriate *Molecule* object as an external data source is initially scanned. The problem is that deciding which *Molecule* object a particular *Crystal* refers to is not always simple. For example, is the molecular formula H_2O "identical" to the formula *HHO*? The two formulas are not identical strings, however, they are equivalent representations of the same molecular formula. The most desirable solution is to recognize this equivalence and have a single chemical formula object representing this molecule in the database. The difficulty is that there is no "normal form" for representing chemical formulas. The NBS Crystal Database uses a highly structured and complex format for formulas. On the other hand, the TEK Cache files do not contain an explicit formula string; the formula must be inferred by enumerating the atoms that are a part of the molecule.

As a partial solution to this particular problem we have used the power of the object-oriented paradigm to develop a representation for chemical formulas. We construct *ChemicalFormula* objects from a string representation of the formula by parsing the string. *ChemicalFormula* objects present a canonical form for chemical formulas that we can use to more easily detect cases where two crystals are based on the same molecule. Figure 11 presents the canonical structure for both H_2O and *HHO*.

However, our strategy is not robust enough to match formulas that are not virtually identical. As a result, we have endowed *ChemicalFormula* objects with methods that allow us to compare different formulas for similarity and thus select crystals based on similar, if not identical formulas.

5.5 Space and Time Considerations

Our three-level architecture introduces a good deal of overhead to merely retrieve a single data value from an object stored in an external data source. However, performance can significantly be enhanced by caching data at the various levels of the architecture.

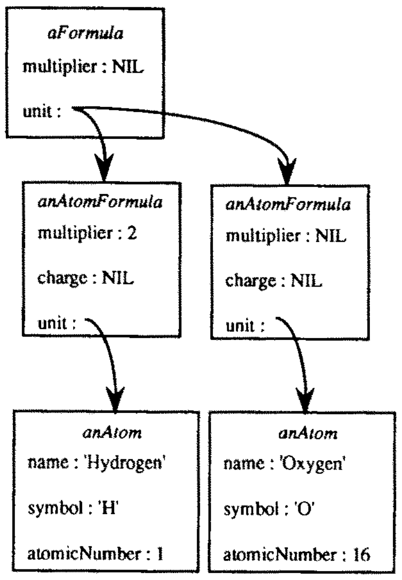

FIGURE 11 Canonical structure for H_2O and *HHO*.

5.5.1 Caching

Objects in the translational level access data from external data sources a whole record at a time. A simple optimization we can perform is to cache the data in the TO as it is read from the external data source. Caching the whole record is advantageous because queries are often predicated on more than one attribute of an object. By caching entire data records, a query over the a, b, and c attributes of a UnitCell object, for example, will result in only a single access to the external data source.

Likewise we can cache attributes in the OR at the object level as they are retrieved from the TO in the translational level. Caching attributes will accelerate repeated access to the same attribute.

5.5.2 Space Considerations

So far, tests of the system have been performed using a very small subset of the data currently available to us from external data sources. Our current disk capacity is more than sufficient to hold not only the data at the external level, but also the cached data at the translational and object levels as well. However, uncontrolled caching would quickly result in external data sources being completely duplicated in the translational and object levels of the database. If space were inexhaustible we would simply load entire external data sources into an object-oriented database. However, replicating data sources that are hundreds of megabytes each is impractical if not impossible. In addition, we expect only a small subset of these data sources to be relevant at any given time.

Hence the goal is to cache relevant data, while flushing irrelevant data as soon as possible. Initially we considered simply flushing the data cached in TOs immediately, resulting in no caching at that level. However, if data are not cached at the translational level, a query predicated on two different attributes of the same object results in us reading the data in to satisfy the message for the first attribute, flushing the data, and then re-reading and flushing the data again to satisfy the message for the second attribute.

Our current solution is to implement a flush method in each translational level class. The advantage of an explicit flush method is that we have total control over when and what objects are flushed. The disadvantage is that a single query that ranges over all instances of a class must use the flush message otherwise hundreds of megabytes of data will be cached, possibly exceeding available disk capacity.

An explicit flush message may also work at the object level. However, caching is not as severe a problem at this level because we are caching single attributes instead of entire data records. For example, caching an NBS Crystal Database Record at the translational level consumes approximately 1200 bytes of space. A single attribute such as the gamma attribute of a corresponding *UnitCell* object at the object level consumes only the space necessary for the floating-point object.

Because we have yet to examine the pattern of use at the object level, we are reluctant to propose solutions for flushing data cached in that level. Our plan is to periodically detect the objects at the object level that are unlikely to be accessed, flush the attributes that have been cached from external data sources, and leave untouched the attributes in objects with which users are currently working. We expect to be able to keep track of which objects users are working with by maintaining private collections of objects for each user.

5.5.3 Performance

We have performed some preliminary tests of our architecture using small subsets of the three data sources we have described earlier. The tests were conducted on a Sun SPARC with 48 megabytes of memory. Our database has been implemented us-

Table 1. Access Time (in seconds)

	String	Float
Nonresident	695	715
Partially resident	15	35
Resident*	5	5

* These times represent the performance of GemStone version 2.5 with no optimization. We have discovered simple optimizations that result in times more than one order of magnitude faster for larger collections.

ing version 2.5 of the GemStone OODBMS. (Our preliminary tests of GemStone 3.0beta indicate approximately a doubling of performance over version 2.5 for querying large collections.)

For the purposes of testing, the object-level database consists of 1200 *Crystal* and associated *UnitCell* and *Molecule* objects. About 1156 of these *Crystal* ORs represent a subset of the NBS Crystal Database (representing approximately 1% of that database), 40 represent files from the Desktop Microscopist, and 4 represent files from the TEK Cache system. All of the external data sources, including the subset of the CD-ROM-based NBS Crystal Database, reside on a local hard disk.

The test consisted of using the GemStone-Smalltalk Interface to interactively access a single attribute from all instances of the *Crystal* class with the database in a nonresident, partially resident, and resident state. In the nonresident state, no data are cached in either the translational level or the object level and all queries must access the external level. In the partially resident state, entire data records required. In the resident state, attribute values are cached at the object level, providing performance equal to the basic performance of the OODBMS we are using.

Table 1 presents the time to retrieve the value of an attribute from each of the *Crystal* objects in database with the database in a nonresident, partially resident, and resident state. The times given are average times for five trials for each of the three states. We performed this series of tests for two attributes, one a string attribute, which required no conversion from its external data format, and the other a floating-point attribute, which required additional conversion at the translational level from the external ASCII string format to an internal floating-point representation.

As the data in Table 1 demonstrate, caching provides better than one order of magnitude increase in performance between the external and translational levels, and from three to five times the performance of the translational level at the object level. As expected, the conversion of the floating-point value added substantially to the time required to access data at the translational level.

We hope to substantially improve the performance between the external and translational levels with version 3.0 of GemStone. The only access to the operating system currently provided by GemStone is through spawning a new shell. In accessing the data from the NBS Crystal Database, TOs must spawn a new shell for each record accessed! GemStone 3.0 will add support for interprocess communications (IPC), which will allow us to develop separate "server" processes to access data in external data sources. These server processes will reduce the overhead of accessing a record from spawning a shell to sending and receiving a few IPC messages.

5.5.4 Query Performance and Optimization

Our architecture introduces some potential inefficiencies in querying external data sources.

One source of inefficiency is the object-by-object strategy of accessing entities stored in external data sources. Data are only retrieved from external data sources when an individual object in the database is queried for a value that is not currently stored in the database. As depicted in Figure 10, when data are not present in an OR, the OR forwards the query on to its related TO, which then retrieves the data from the external level. Although this is a reasonable data access mechanism for data sources with no query support such as the NBS Crystal Database and the files of the Desktop Microscopist and TEK Cache systems, it is not an efficient mechanism for accessing an external relational database.

A relational database management system is designed to provide efficient retrieval of sets of records rather than single records. Our current mechanism will require data to be retrieved from a relational database one entity at a time by generating an SQL query for each unique object in the database. However, the data sources we currently access, or have plans to incorporate, are not managed by relational database management systems.

Another potential inefficiency may be the order in which external databases are accessed. We currently have no control over the order in which messages are sent to objects in the translational level. In the event that we are accessing multiple, large CD-ROM data sources, a random access pattern could result in a user being forced to perform an endless series of CD-ROM swaps. Even with a planned CD-ROM "jukebox" this thrashing between data sources would unacceptably degrade performance.

The solution to this lack of ordering may be as simple as ordering the collections based on the class of the *realObject* attribute of the OR in the collection. Queries over such an ordered collection would then follow a more desirable access pattern, completely accessing each data source in turn.

6 CONCLUSION

We have presented the design and initial development of a hybrid object-oriented HDB for integrating diverse data sources supporting materials science research.

The object-oriented data model has proven more than adequate for modeling crystallographic data.

Our unique three-level architecture with its persistent object representatives appears to be a suitable architecture for integrating data sources that remain relatively static. The architecture provides efficient access to external data by providing a static structure for queries based on components of complex objects. In addition, the architecture provides opportunities for enhancing performance by caching data within both the object and translational levels of the architecture.

We have found the database relatively easy to extend to new data sources, and currently provide transparent access to three structurally diverse data sources.

Future Work

The development of a more efficient mechanism for accessing external data sources is critical to the usefulness of the database. The IPC capabilities of version 3.0 of the GemStone OODBMS will allow the performance at the external level to be improved significantly.

In addition, two important issues remain to be addressed by our database. First, it must be possible to associate meta-data with data retrieved from the external data sources. The minimum meta-data requirement is simply to know the data source from which the data are taken. This is critical because the architecture allow researchers to substitute their own value for an attribute retrieved from an external data source if they wish. Thus assumptions cannot be made about the source of an attribute simply by the determining the data source from which the data are expected to have been retrieved.

The second issue is the development of a tunable data retention mechanism for flushing little-used data that have been cached from an external data source. Once we have had an opportunity to examine and understand patterns of data access between the levels of our architecture we will be able to develop space management policies that will minimize space consumption and maximize performance.

6.1.1 Application Interface

Finally, we will soon begin the process of modifying materials science application programs to access the database. The Desktop Microscopist will be modified to use the database for all data storage and retrieval rather than the ASCII files it currently uses.

In addition, the materials scientists with whom we are collaborating are investigating other subdomains of materials science for inclusion in the database. We hope to provide support for applications that calculate phase diagrams by incorporating external data sources containing thermodynamic data into the database.

REFERENCES

[1] J. R. Rumble. "Socioeconomic barriers in computerizing materials data." In J. S. Glazman and J. R. Rumble. Eds., *Computerization and Networking of Materials Data Bases*. Philadelphia, PA: ASTM. 1989, pp. 217–226.

[2] S. R. Hall. "The design, development and implementation of program systems." In G. M. Sheldrick, C. Kruger, and R. Goddard, Eds., *Crystallographic Computing 3: Data Collection, Structure Determination, Proteins, and Databases*. New York: Oxford University Press, 1985, pp. 147–156.

[3] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*. Redwood City, CA: Benjamin/Cummings, 1989.

[4] C. Hightower and R. Schwarzwalder. "A comprehensive look at materials science databases," *Database*, vol. 14, pp. 42–53. 1991.

[5] M. E. Williams. "Electronic databases," *Science*, vol. 228, pp. 445–456, 1985.

[6] G. Bergerhoff. "The inorganic crystal structure

database," In G. M. Sheldrick, C. Kruger, and R. Goddard, Eds., *Crystallographic Computing 3: Data Collection, Structure Determination, Proteins, and Databases*. New York: Oxford University Press, 1985, pp. 85–95.

[7] S. V. Meschel, "Numeric databases in the sciences," Online Rev. vol. 8, pp. 77–103, 1984.

[8] D. M. Hansen, "An extensible data model for materials science," May 1992.

[9] A. P. Sheth and J. A. Larson, "Federated database systems for managing distributed heterogeneous, and autonomous databases," *ACM Comput. Surv.* vol. 22, pp. 183–236, 1990.

[10] A. K. Elmagarmid and C. Pu, Guest editors' introduction to the special issue on heterogeneous databases," *ACM Comput. Surv.* vol. 22, pp. 175–181, 1990.

[11] W. Litwin and A. Abdellatif," Multidatabase interoperability," *IEEE Comput.* vol. 19, pp. 10–18, 1986.

[12] J. M. Smith, P. A. Bernstein, U. Dayal, N. Goodman, T. Landers, K. W. T. Lin, and E. Wong, "Multibase-integrating heterogeneous distributed database systems," In *AFIPS Proceedings of the National Computer Conference*, vol. 50, Reston, VA, 1981. American Federation of Information Processing Societies, Inc., AFIPS Press.

[13] W. Litwin, L. Mark, and N. Roussopoulos, "Interoperability of multiple autonomous databases," ACM Comput. Surv. vol. 22, pp. 267–293, 1990.

[14] C. Batini, M. Lenzerini, and S. B. Navathe, "A comparative analysis of methodologies for database schema integration," ACM Comput. Surv. vol. 18, pp. 323–363, 1986.

[15] W. Litwin, *Exploring Technology: Today and Tomorrow*. Dallas, TX, 1987, pp. 751–752.

[16] G. Thomas, G. R. Thompson, C. W. Chung, E. Barkmeyer, F. Carter, M. Templeton, S. Fox, and B. Hartman, "Heterogeneous distributed database systems for production use," *ACM Comput. Surv.*, vol. 22, pp. 237–265, 1990.

[17] F. Manola, "Applications of object-oriented database technology in knowledge-based integration in formation systems," In A. Gupta, Ed., *Integration of Information Systems: Bridging Heterogeneous Databases*, Selected Reprint Series. New York: IEEE Press, 1989, pp. 126–134.

[18] F. Saltor, M. Castellanos, and M. Garcia-Solaco, "Suitability of data models as canonical models for federated databases," In A. Segev, Ed., *SIGMOD Record—Special Issue: Semantic Issues in Multidatabase Systems*, vol. 20. ACM Press, 1991.

[19] J. L. McCarthy, "Information systems design for material properties data," In J. S. Glazman and J. R. Rumble, Eds., *Computerization and Networking of Materials Data Bases*. Philadelphia, PA: ASTM, 1989, pp. 135–150.

[20] F. J. Smith and M. V. Krishnamurthy, "Integration of scientific data and formulae in an object-oriented system," In H. Hinterburger and J. C. French, Eds., *Proceedings of the Sixth International Working Conference on Scientific and Statistical Database Management*, Zurich, June 1992, pp. 110–122.

[21] B. Czejdo and M. Taylor, "Integration of database systems using an object-oriented approach," In *First International Workshop on Interoperability in Multidatabase Systems*, Kyoto, Japan, April 1991. New York: IEEE Computer Society Press, 1991, pp. 30–37.

[22] Q. Li and D. McLeod, "An object-oriented approach to federated databases," In *First International Workshop on Interoperability in Multidatabase Systems*, Kyoto, Japan, April 1991. New York: IEEE Computer Society Press, pp. 64–70.

[23] E. Bertino, *First International Workshop on Interoperability in Multidatabase System*, Kyoto, Japan, April 1991. New York: IEEE Computer Society Press, 1991, pp. 22–29.

[24] H. R. Tirri, J. Srinivasan, and B. Bhargava, "Integrating distributed data sources using federated objects," In *Pre-Proceedings of the International Workshop on Distributed Object Management*. Edmonton, Canada, August 1992, pp. 292–306.

[25] NIST standard reference data products catalog, 1992.

[26] F. Eliassen and R. Karlsen, "Interoperability and object identity," In A. Segev, Ed., *SIGMOD Record-Special Issue: Semantic Issues in Multidatabase Systems*, vol. 20. ACM Press, 1991, pp. 25–29.

[27] Y. Brietbart, P. L. Olson, and G. R. Thompson, "Database integration in a distributed heterogeneous database system," In A. Gupta, Ed., *Integration of Information Systems: Bridging Heterogeneous Databases*, Selected Reprint Series. New York: IEEE Press, 1989, pp. 221–230.

[28] M. Castellanos, F. Saltor, and M. Garcia-Solaco, "A canonical model for the interoperability among object-oriented and relational databases," In *Pre-Proceedings of the International Workshop on Distributed Object Management*, Edmonton, Canada, August 1992, pp. 365–370.

[29] J. Chomicki and Witold Litwin, "Declarative definition of object-oriented multidatabase mappings," In *Pre-Proceedings of the International Workshop on Distributed Object Management, Edmonton*, Canada, August 1992, pp. 307–325.

[30] U. Dayal, "Query processing in a multidatabase system," In W. Kim, D. S. Reiner, and D. S. Batory, Eds., *Query Processing in Database Systems*. New York: Springer-Verlag, 1985, pp. 81–108.

[31] A. K. Elmagarmid, *Exploring Technology: Today and Tomorrow*. Dallas, TX, 1987, p. 746.

[32] A. Goldberg and D. Robson. *Smalltalk-80: The Language and Its Implementation.* Reading, MA: Addison-Wesley. 1983.

[33] S. Heiler. *Exploring Technology: Today and Tomorrow.* Dallas, TX. 1987. 749–750.

[34] D. K. Hsiao and M. N. Kamel. "Heterogeneous databases: Proliferations, issues and solutions." IEEE Trans. Knowledge Data Eng. vol. 1. pp. 46–62. 1989.

[35] S. Iwata. "Expert systems interfaces for materials data bases." In J. S. Glazman and J. R. Rumble. Eds.. *Computerization and Networking of Materials Data Bases.* Philadelphia. PA: ASTM. 1989. pp. 175–184.

[36] T. Kaehler and D. Patterson. *A Taste of Smalltalk.* New York: W. W. Norton. 1986.

[37] W. Kent. "The breakdown of the information model in multi-database systems." In A. Segev. Ed.. *SIGMOD Record—Special Issue: Semantic Issues in Multidatabase Systems*, vol. 20. ACM Press. 1991. pp. 10–15.

[38] W. Kent. *Proceedings of the 17th International Conference on Very Large Data Bases.* Barcelona. Spain. September 1991. pp. 147–160.

[39] W. Kim. *Introduction to Object-Oriented Databases. Computer Systems.* Cambridge, MA: The MIT Press. 1990.

[40] H. S. Lipson. *Crystals and X-rays.* The Wykeham Science Series. London: Wykeham Publications. 1970.

[41] P. Machin. "Programming aspects of crystallographic data files: Interactive retrieval from the Cambridge database." In G. M. Sheldrick. C. Kruger. and R. Goddard. Eds.. *Crystallographic Computing 3: Data Collection, Structure Determination, Proteins, and Databases.* New York: Oxford University Press. 1985. pp. 107–118.

[42] G. Maini. A. Napoli. D. Colnet. D. Léonard. and K. Tombre. *Object-Oriented Languages*, vol. 34 of A.P.I.C. San Diego, CA: Academic Press Inc.. 1991.

[43] F. Manola and S. Heiler. "An approach to interoperable object models." In *Pre-Proceedings of the International Workshop on Distributed Object Management*, Edmonton, Canada, August 1992. pp. 326–330.

[44] F. C. Phillips. *An Introduction to Crystallography*, 3rd ed. London: Longmans, Green and Col. Ltd.

[45] J. R. Rumble and D. R. Lide. "Chemical and spectral databases: A look into the future." J. Chem. Information Comput. Sci. vol 25. pp. 231–235. 1985.

[46] M. Rusinkiewicz. *Exploring Technology: Today and Tomorrow.* Dallas. TX. 1987. p. 753.

[47] A. P. Sheth. *Exploring Technology: Today and Tomorrow.* Dallas. TX. 1987. pp. 747–748.

[48] A. Shoshani. F. Olken. and H. K. T. Wong. "Characteristics of scientific databases." In *Proceedings of the Tenth International Conference on Very Large Data Bases*, 1984. pp. 147–160.

[49] R. Taylor. "The Cambridge data file." In G. M. Sheldrick. C. Kruger. and R. Goddard. Eds.. *Crystallographic Computing 3: Data Collection, Structure Determination, Proteins, and Databases.* New York: Oxford University Press. 1985. pp. 97–105.

[50] S. B. Zdonick and D. Maier. *Readings in Object-Oriented Database Systems.* San Mateo, CA: Morgan Kauffman. 1990.