Intermedia: A Case Study of the Differences Between Relational and Object-Oriented Database Systems

Karen E. Smith
Institute for Research in Information and Scholarship

Stanley B. Zdonik
Department of Computer Science

Brown University
Providence, Rhode Island 02912

ABSTRACT

This paper compares two approaches to meeting the data handling requirements of Intermedia, a hypermedia system developed at the Institute for Research in Information and Scholarship at Brown University. Intermedia, though written using an object-oriented programming language, relies on a traditional relational database management system for data storage and retrieval. We examine the ramifications of replacing the relational database with an object-oriented database. We begin by describing the important characteristics each database system. We then describe Intermedia and give an overview of its architecture and its data handling requirements. We explain why and how we used a relational database management system and the problems that we encountered with this system. We then present the design of an object-oriented database schema for Intermedia and compare the relational and object-oriented database management system approaches.

1. INTRODUCTION

The Institute for Research in Information and Scholarship (IRIS), as part of a research project designed to enhance education and research, has developed powerful hypermedia software that allows professors and students to create and follow links between electronic documents [Conk86, Enge68, Nels80, Yank85]. The system, called Intermedia, provides a framework for a collection of editors written in an object-oriented language, each capable of allowing sophisticated connections between pieces of information in its documents [Garr86a, Meyr85a]. Currently these editors include a word processor, a structured graphics editor, a timeline editor that allows users to organize information interactively in temporal sequences, a scanned-image viewer, and a viewer that displays and rotates three-dimensional models.

To provide persistence, concurrency, and access control to the data that represent the links and to some of the document information itself, we use the INGRES relational database management system [Ston76]. At times, however, using INGRES is awkward as the information is made up of complex, hierarchical data structures that have to be flattened into relations.

We contend that an object-oriented database would provide a better facility for modelling these complex structures [Bane87, Maie86a-b, OBri86, Oren87, Zdon84]. One such database is the Encore object-oriented database management system that is under development at Brown's Computer Science Department. This paper explores several ways in which the use of Encore could improve the structure of the Intermedia code and the performance of the resulting system[1]. To do so we present several of the relations that Intermedia uses and the code for an action that accesses these relations. We then describe how Encore might be used by Intermedia and provide the code that would perform the same action. Our intention is to illustrate some of the fundamental differences between relational and object-oriented database systems. Although Intermedia's use of Encore is currently only hypothetical, we do intend to use Encore in a future version of Intermedia.

2. OVERVIEW OF THE DATABASE SYSTEMS

In the commercial world, relational database systems [Codd79] have recently become popular for data processing applications. The relational model is more flexible and easier to use than previous database models. It is more flexible because inter-record relationships do not have to be pre-defined. The relational join operator allows a user to relate records dynamically based on attribute values. The relational model is easier to use because of its more intuitive metaphor of tabular data and because of fourth-generation query languages, which allow a user to pose ad hoc retrieval requests to the system.

[1] In this paper, all statements about object-oriented database systems refer to Encore.

Conventional database management systems have been successful at supporting data-intensive record-processing applications. Although these applications often require a large amount of code to produce many varied reports, the level of complexity as measured by the interactions between modules is relatively low. We believe that there exists a large class of applications for which relational database systems and other more conventionally-structured database systems are too limited. These applications can be characterized as complex, large-scale, data-intensive programs. One example is a computer-aided design system in which the designer creates hierarchy of inter-related design objects. Intermedia is another example in which the user creates a network of connected pieces of information. This class of applications needs a database model that is more expressive and flexible than the relational model. Furthermore, this class of applications needs database technology that is designed to facilitate programming-in-the-large, i.e., designing, constructing, and maintaining large, complex programs.

Object-oriented databases are being developed to meet the data handling needs of such applications. An object-oriented database stores class definitions and instantiations of these classes (i.e., objects). All objects have a unique identity [Khos86], and other objects can refer to this identity. Furthermore, the database can invoke a method on an object directly because it stores the methods for each class.

An object-oriented database differs from a relational database in the following ways:

- class extensibility

  With a relational database, there is only a single parameterizable type, Relation. The operations on all relations are limited to get-field-value and set-field-value (Figure 1). In the object-oriented approach, each object is associated with a class. User defined types, or classes, are at the same semantic level as the built-in types. The interface to each object is customized to that object. A user can create new classes from existing classes by supplying a specification and an implementation of that specification. The specification is expressed in terms of method signatures and property declarations.

- data abstraction

  In an object-oriented database, the behavior of an object is described by a class definition. Each of these classes is defined by a data abstraction. By incorporating data abstractions at the level of the database, it is possible to make changes to the way a database class is implemented without any effect on other classes in the database that make use of the abstraction.

- ability to store active objects

  The data abstractions include a description of the methods that can be invoked on instances of the class. Because these methods are stored as objects in

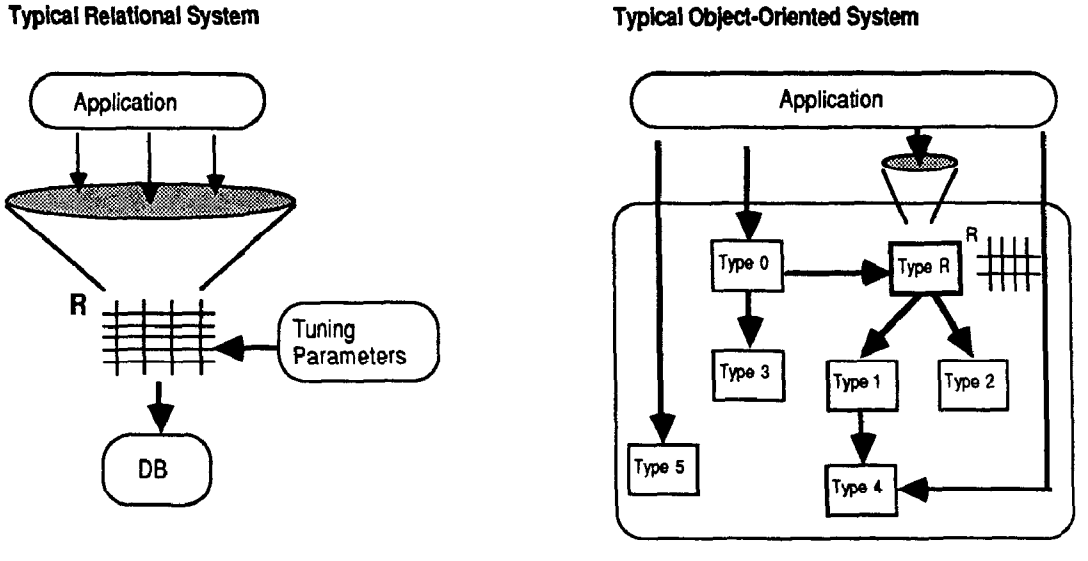

Figure 1: An overview of the two database architectures. With a relational system the application represents everything with relations. One can interact with the underlying storage structures in very limited ways by specifying tuning parameters (e.g., indices on attributes). The application, however, cannot get past the wall of abstraction to change how the database stores tuples. With an object-oriented system, one can build a class called, "relation," that behaves exactly as the INGRES interface. It is also possible to make use of lower level types within the same application program. Note that this does not violate data abstraction.

the database, an application can ask the database to invoke a method on an object. Since methods are expressed in terms of a general-purpose programming language, any operation can be constructed and stored in the database. This is in contrast to relational query languages that are not computationally complete. Furthermore, by storing methods as objects, the programming environment can be made more uniform [Zdon86]. As a result, the bulk of the application code resides in the database and can be managed by all of the database facilitates (e.g., concurrency, recovery, and version control).

- no need to copy data to virtual memory

  Because the application can send messages to the database and have it invoke methods on objects, the application does not need to copy these objects into its virtual memory. This does not mean the data is never moved to the workstation. For example, the Encore system kernel resides on the workstation, and the application communicates with it by sending messages. Encore actually manipulates the data in its own local buffers. This scheme allows caching of data on the workstation, avoids memory-to-memory copies, and enforces data abstraction.

- automatic type checking at the point of use

  Since methods are executed locally, the object-oriented database itself can perform type checking on the arguments to method calls. Type failures can be detected at the time of invocation instead of at the end of a transaction when the new values are checked back in, as is done in a relational database.

We will now examine Intermedia to explore the differences between relational and object-oriented database systems. Before explaining the architecture and data handling needs of Intermedia, we need to describe the system from the user's perspective.

3. DESCRIPTION OF INTERMEDIA

With Intermedia, users can browse through linked information in a non-sequential but controlled manner. Anything that can be selected in a document can become a block, an end point of a link. There can be many blocks in a document, and any number of links can emanate from a block.

To create a link, the user selects a source block in a document, chooses the command to start a link, selects a destination block, and chooses the command to complete the link. When the operation is finished there is a semantic tie – a navigational link – between the source and destination blocks (Figure 2a-b). This link is represented to the user by markers embedded in the documents. Whenever the user selects a marker and picks the command to follow it, the system displays the explainers of the links that emanate from the marker (Figure 2c-d). When the user picks which link to follow, the document containing the other end of the link is opened and the appropriate information is presented in another window (Figure 2e).

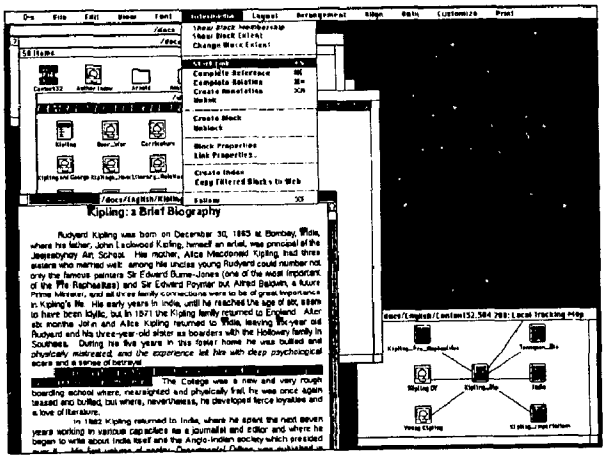

Figure 2a: The user has selected, "Between 1878 and 1882...," and is about to pick the Start Link command.

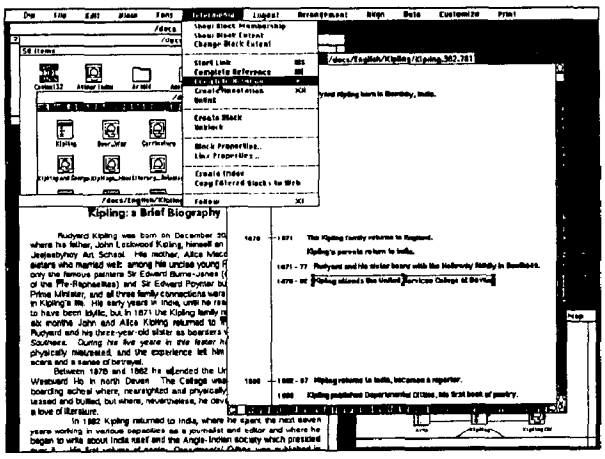

Figure 2b: The user has opened another document, selected an event on a timeline, and is about to choose the Complete Relation command, thereby completing the link.

Users can attach any number of keywords to blocks and links. In the future users will be allowed to apply predicates, or filters, to documents so that only the markers meeting their filtering requirements are shown. Webs present users with contexts in which to collect a set of links that are superimposed upon a set of documents. A document can exist within the context of many webs, but only those markers in the current web are shown in the document. Visual representations of webs, or maps, are provided to users to help them explore and understand the linked context.

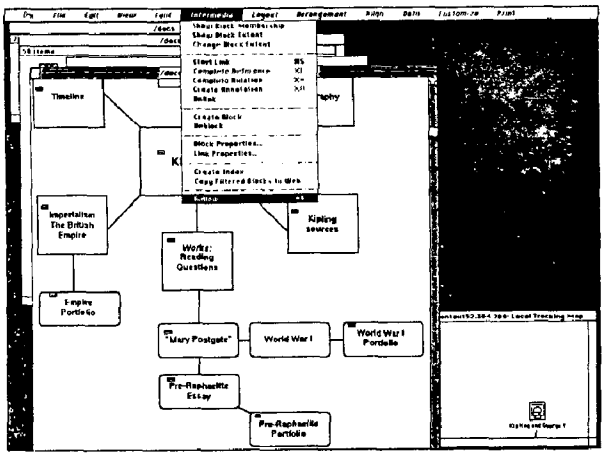

Figure 2c: Once the link is established, Intermedia displays a marker at each end. Here the user has selected the marker near "Pre-Raphaelite Portfolio" and is about to choose the Follow command.

Figure 2e: The document at the other end of the link opens with the other link marker selected and with the destination, "The Awakening Conscience," highlighted.

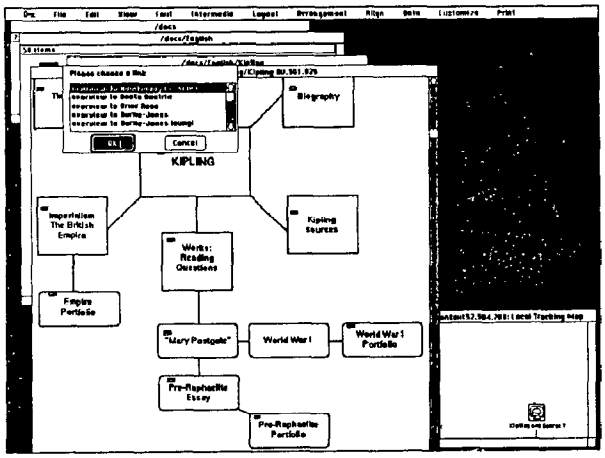

Figure 2d: Because more than one link emanates from the selected marker, Intermedia displays a list of the link explainers. Here, the user is about to choose the first link in the list.

4. CURRENT SYSTEM ARCHITECTURE

To facilitate program development, to enforce user-interface consistency, and to impose a standard architecture upon the different applications, we chose to use object-oriented programming to build Intermedia. We wanted not only an object-oriented language, but also a set of classes that defined the user interface and behavior of standard applications. In particular, we were interested in using MacApp [Doyl86, Schm86], a set of classes that define behaviors common to all Macintosh applications. Under special agreements with Apple Computer, Bolt Beranek and Newman (BBN), and Cadmus Computer, we were able to use a C/UNIX 4.2 version of MacApp on the IBM RT PC. BBN provided Inheritance C [Meyr85b, Nodi85, Smit87], an object-oriented extension to C that is supported by a pre-processor and a set of macros. The port of MacApp to C was partially done at BBN and was completed at IRIS. Cadmus wrote Cadmac, a C version of the Macintosh Toolbox that runs on UNIX; we ported Cadmac to the RT under a special licensing agreement.

Because our applications share functionality beyond that provided by the standard MacApp classes, we extended the set of classes by creating the Intermedia Layer. This layer covers the functionality needed by all Intermedia editors, such as management of blocks and links and an interface to a Macintosh-style "Finder" for file management. The Intermedia layer contains subclasses of some MacApp classes, such as IntAppl which adds data fields and methods for managing more than one type of editor to the standard MacApp FileAppl class, IntDoc which adds saving, storing, and retrieving block and link information to the MacApp Document class, and IntView which adds link marker rendering to the MacApp View class. The Intermedia layer also adds new classes, like Block, which contains information concerning the location of the block in a document and the links which emanate from it and Link, which models the link between two block objects. Therefore, where the MacApp layer provides functionality for a standard Macintosh application, the Intermedia layer provides functionality for a standard Intermedia application. For a more thorough discussion of the architecture of Intermedia, see [Meyr86] and [Garr86b].

5. DATA HANDLING REQUIREMENTS

Because Intermedia is a multi-user application, with users simultaneously examining and creating links between the same documents, concurrency and access control were important concerns. Appropriate levels of control assure the integrity of information, allow individuals working together to have full access to their shared data, and prevent those without permission from viewing or modifying such data. Intermedia provides three categories of access to

documents. Readers may examine the content of a document and can follow links. Annotators, in addition to having readers' privileges, can add links but cannot alter the content of the document. Writers have all of the capabilities of annotators and can also modify the content of a document.

In the current release of Intermedia, only one user at a time can have write access to a document, while multiple users can have annotate or read access. This model for access rights implies that the level of granularity for concurrency control is not the entire document but is the individual blocks and links between the documents. We chose to use a relational database management system to store the block and link information and to provide the concurrency control needed for referencing and updating. With the concurrency control provided by a DBMS, no two individuals could change explainers or keywords or delete the same block or link simultaneously. However, the concurrency control would not prevent multiple users from adding and/or following links at the same time.

list of blocks, each block has a list of keywords and a list of links that emanate from it, and each link has a list of keywords. When the user creates a link between two blocks (Figure 2a-b), a block object is created for each end point (assuming that the block had not already been allocated by a previously-made link). Each block object is added to the list of blocks of its associated document object. A link object is created and added to the list of links for both the source and destination blocks.

When a user selects a link marker and picks the command to follow it, the system opens the document containing the other end of the link and retrieves the block and link information for that document from the database (Figure 2c-e). Internally, the system attaches the data structure for a user-level marker to a system-level block object. When a marker is selected, its block becomes active. When the user picks the follow command, the system examines the active block's list of links. If there is only one link, the system will follow it to the destination block. If there is more than one link, the system will display a dialog box

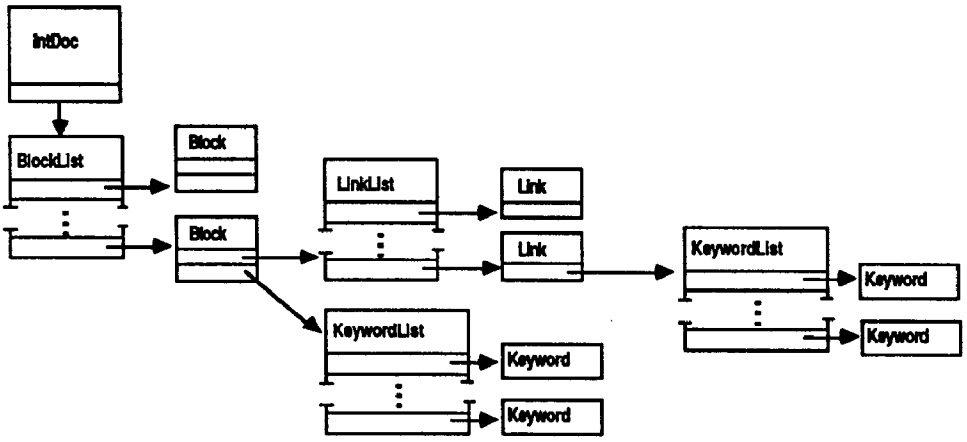

Figure 3: The object ownership hierarchy of documents, blocks, links, and keywords.

A document's block list contains the set of blocks that correspond to the web that the user currently has open. The web object represents this open web, and all interactions with the database are done through the web object. Therefore, it provides the interface to the database through which the document, block and link object hierarchies are recreated. When a document opens, the web object receives a message to retrieve the blocks in this document. The web then sends a message to retrieve the links for each block. We explain these interactions with the database in section 6.1.

The object hierarchy that needs to be stored in the database consists of documents' lists of blocks and links (Figure 3). As stated previously, each document has a

containing the explainers of all the block's links. The user can then select which link to follow.

6. COMPARISON OF THE TWO DATABASE MANAGEMENT SYSTEMS

To compare the two systems, we will examine the code necessary for accessing the links in a document that a user has just opened. We first look at the relational schema and the code for retrieving the link information from these relations. Next, we present how we would store and access the same information in Encore. Given these examples, we then compare the two database systems.

6.1 The Relational System

In choosing an appropriate database management system, we were guided mostly by what was available. Our target machine was the IBM RT PC running 4.2 BSD UNIX, and in the fall of 1985, there were no commercially-available databases for the RT. We therefore turned to the university version of the INGRES relational database management system from UCB [Ston76] which runs on 4.2 UNIX and only had to be ported to the RT[2].

In designing our relational schema, we had a requirement to store data for blocks and links. For each type of object, we created a series of normalized relations to avoid duplication of information. Each of these relations contains an ID field that serves as the database key. Therefore, when the application needs to create a new object, it allocates the memory for it, requests a unique ID from the database, and initializes the object's instance variables. In Intermedia, storing an object in the relational database means copying those instance variables that must be saved in a database record. This copying is performed with the query language. Retrieving an object means allocating its memory and then using the query language to reconstruct the object. This involves assigning some of the object's instance variables to values from the database while initializing all other instance variables to default values.

To provide concurrency control, INGRES has a locking mechanism which, depending on the scope of lock, acts either on entire relations or on individual records. When a record is appended to a relation, the query language requests a write lock on the relation. Such a lock prevents others from reading or writing any information in the relation. The lock is freed when the database has finished appending the record. When an individual record is updated, the query language requests a write lock on that record. This kind of lock will be deferred if there is a write lock on its relation. Once granted, it prevents other write locks on the same record or on the relation and any read locks on that record. It is freed after the record is updated. When a record is being retrieved, INGRES attaches a read lock to the record, reads the information, and frees the lock. A read lock will be deferred if another user has requested a write lock on the same record or on its relation. A read lock prevents another user from writing the record or relation. These locks provide concurrency control to the information contained in the relational system.

The link class described below illustrates the kind of information that is stored in INGRES. It is defined in Inheritance C in a format that first describes the instance variables and then the methods, with a syntax that is similar to C:

```
CLASS Link (Object)          /* Link is a subclass of the generic Object          */

bool    fNew;                /* is this a new link?                               */
bool    fChangedKeyword;     /* has any keyword been added or deleted?            */
bool    fChangedExplain;     /* has the explainer been changed?                   */
bool    fDeleted;            /* is this link deleted?                             */
long    fCreateDate;         /* date the link was first made                      */
long    fModifyDate;         /* date the link was last changed (keywords, etc.)   */
char    *fAuthor;            /* Author user ID                                    */
short   fExplainExtend;      /* flag if explainer > 30 characters                 */
TExplain fExplainer;         /* pointer to the explainer string object for link   */
TList   fKeywordList;        /* pointer to list of keywords assigned to this link */
short   fLinkId;             /* INGRES key for retrieving this link               */
short   fType;               /* type of link - Reference, Relation, Annotation    */
short   fSrcBlockId;         /* INGRES key for source block                       */
short   fSrcDocId;           /* INGRES key for document containing sourceblock    */
short   fDestBlockId;        /* INGRES key for destination block                  */
short   fDestDocId;          /* INGRES key for doc containing destination block   */
TBlock  fSrcBlockHand;       /* pointer to source block object                    */
TBlock  fDestBlockHand;      /* pointer to destination block object               */

PROCEDURES

        lLink();             /* Initialize the link object                        */
        CompleteLink();      /* Complete the link                                 */
        UnLink();            /* Remove the link                                   */
TObject Clone() OVERRIDE;    /* Copy the link                                     */

END
```

To model this class in INGRES, we used six relations: Link, LExtend, Keylink, Keyword, Linkfree, and Keyfree (Figure 4). The Link relation stores most of the persistent information of the link object (i.e., Link ID, Type, Block ID1, Block ID2, Explainer, CreateDate, ModifyDate, and Owner). To optimize retrieval, this relation also includes the blocks' document IDs so that all of the links for a document can be retrieved without having to do an expensive join between the link and block relations. Because INGRES only provides for fixed length character strings, the explainer field only holds the first thirty characters of the explainer. Any overflow is signalled by incrementing the eextend field. The actual overflow characters are stored in the LExtend relation in a sequence of thirty-character records. This limitation, although not an intrinsic property of a relational system, makes dealing with variable length objects very cumbersome.

The link class also contains a list of keywords that have been applied to the link. This list is stored in two relations: the Keylink relation stores the link ID and the keyword ID and the Keyword relation matches the keyword ID to its actual keyword character string.

Because links and keywords need unique identifiers, we need relations to maintain the IDs that have not yet been assigned. The Linkfree relation keeps the free IDs for links while the Keyfree relation has the unused IDs for keywords.

We could not represent the Link class in just one relation for a number of reasons. Because INGRES does not allow variable-length lists to be stored in a single record, we needed to store each keyword in a record in another relation. Also, INGRES does not allow for variable-length character strings. We,

[2]From this point on, all references to INGRES will imply the university version.

LINK RELATION					
Link ID	Type	Doc ID1	Block ID1	Doc ID2	Block ID2
63	2	399	28	180	35
64	1	399	28	352	40

Explainer	EExtend	CreateDate	ModifyDate	Owner
connecting a scanned reproduct	87	861008	870219	kes
Biography to photo	0	861115	861115	kes

LEXTEND RELATION		
Link ID	Sequence	Explainer
63	1	ion of Titian's "Venus and Ado
63	2	nis" to Titian's biography.

KEYLINK RELATION	
Link ID	Key ID
64	20
64	39

LINKFREE RELATION	
Start	End
19	19
65	32767

KEYWORD RELATION	
Key ID	Keyword
20	Titian
21	Venetian

KEYFREE RELATION	
Start	End
120	32767

Figure 4: The six INGRES relations needed to model the link class.

therefore, had to store each variable-length link explainer as a series of records in yet another relation. Finally, because INGRES does not generate unique IDs for keys, we needed to create relations to generate unique IDs for both keywords and links.

When the user opens a document, the document object needs to be created and initialized with values that are stored in these and other relations. The program sends the GetBlockList message to the web object to query INGRES for the information on the blocks contained within the document. It takes one query to retrieve this information, and from it the web builds a list of block objects. It then needs to build up each block's hierarchy. Part of this hierarchy is represented in three relations (one for its keywords, one for its explainer overflow, and one for its extent in the document). Therefore, retrieval of the block list takes another three queries per block. The other part of the hierarchy, each block's list of links, is represented in the Link relation. The obvious way would be for each block to retrieve all of its links, but this would mean accessing the Link relation n times, where n is the number of blocks in the document. Instead, for performance, we retrieve all of the link information in the link relation for this document in one query, initialize link objects, and attach the links to their blocks (Figure 5). On a per link basis, the web must then query the database to retrieve the explainers and the keywords (two queries per link). Therefore, to recreate a document's block and link hierarchy from information stored in INGRES, the web must make 2 + (3 * number of blocks in the document) + (2 * number of links attached to these blocks) queries.

The following section of code is an example of querying the relational database. It is the method that retrieves the links.

In this code, the web is retrieving all of the links that are attached to blocks in the specified document. It accesses the Link relation and retrieves all the tuples that have a docid field equal to the specified document's ID. As each tuple is retrieved, its fields are

```
webdoc_GetLinkList (self, doc, blockList)
TWebDoc     self;       /* the current web object                          */
TWebDocObj  doc;        /* the document object being opened                */
TBlockList  blockList;  /* list of blocks already queried from INGRES      */
{
        /* All QUEL statements are preceded by "##" */
    ...
# #  range of l is relname   /* retrieve from link relation where docID1 or docID2 is doc's ID  */
# #  retrieve (lid = l.linkid,  /* retrieve these fields from link relation and store temporarily */
# #      ltype = l.ltype,    /* in QUEL variables. These variables will eventually be          */
# #      author = l.owner,   /* passed to the initialization method for a link object.         */
# #      bid1 = l.blockid1,
# #      doc1 = l.docid1,
# #      bid2 = l.blockid2,
# #      doc2 = l.docid2,
# #      cdate = l.createdate,
# #      mdate = l.modifydate,
# #      explain = l.explainer,
# #      ext = l.eextend)
# #  where (l.docid1 = did or l.docid2 = did)
# #      {
        tLink = NEWOBJ (TLink,go_Link);                  /* create a new link object */
        /* initialize this object with the values retrieved from INGRES */
        (MESSAGE(tLink,ILink),lid, bid1, doc1, bid2, doc2, ltype, author, cdate,
                        mdate, explain, ext);
        /* add this link to the list and repeat */
        ...
# #      }
    ...
    /* cycle through the list of links - if a link's explainer extension instance variable is set, then its
        explainer is longer than 30 characters and part of it needs to be retrieved from another
        relation - the lextend relation */
    ...
}
```

copied into temporary variables that are then passed to the link's initialization method. Before this method was called, the document's list of blocks was built in a similar manner by copying tuple fields into temporary variables and then into the instance variables of block objects.

Once a document's object hierarchy is in place, the application uses it to respond to the user's actions. For example, when the user selects a marker and picks Follow, the follow command object is created, and it sends a message to the current selection to find the selected block object. It then sends a message to the block object to display, in a dialog box, a list of the explainers of all the links that emanate from it, and the user selects the one to follow. A message is then sent to the block to follow the selected link by opening the document at the other end of the link and highlighting the extent of the other block.

```
...
theLink = (MESSAGE(aBlock, ChooseLink));
(MESSAGE(aBlock, Follow), theLink);
...
```

When the other document is opened, it recreates its object hierarchy that the application will then use to respond to user-initiated requests.

In this section we have seen how an application must copy data from the relational database into its data structures to manipulate it. We will now compare this approach with how an application interacts with an object-oriented database.

6.2 The Object-Oriented System

Encore is an object-oriented database developed at the Computer Science Department at Brown University [Zdon86]. A single-user prototype of Encore that runs on Apollo workstations configured with 4.2 BSD UNIX has been developed, and a multi-user version that uses a shared object server, called ObServer [Skar86b], is under development.

An Encore database is based on a set of programmer-defined classes. Each class exports an interface consisting of a set of methods and a set of properties. Both properties and methods are objects in their own right and therefore have behavior that is described by a

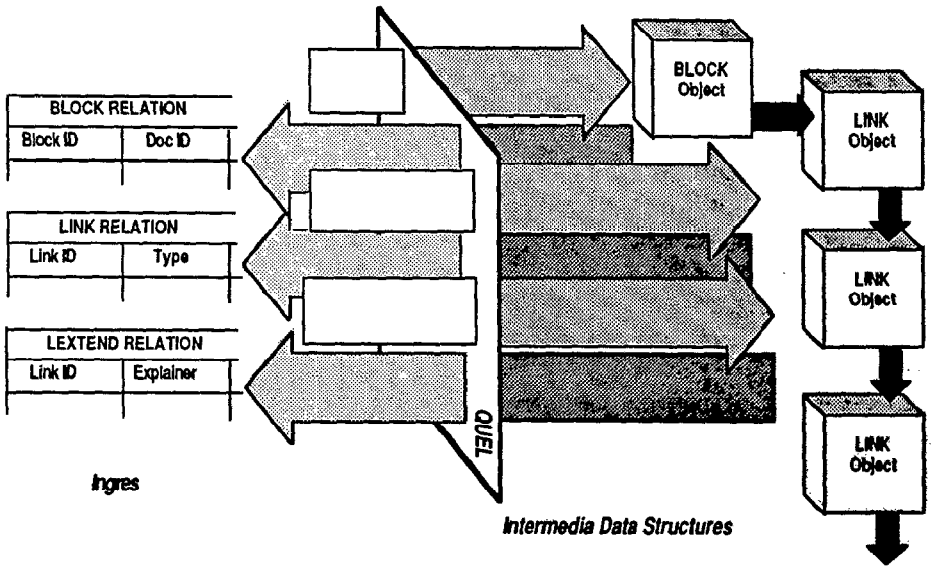

Figure 5: The data stored in INGRES relations must be copied into objects in the Intermedia data structures to be manipulated. When it is time to save the changes, the data must be copied back into the relations. The arrows represent the application program copying pieces of data to and from the database.

class. Properties are different from the instance variables of an object-oriented language like Smalltalk [Gold83]. Properties are the public view of the state of an object while instance variables are private. In many cases, a property might correspond to an instance variable, and the methods of accessing and setting the value of a property might map directly onto manipulating the instance variable. However, this need not be the case, since getting the value of a property might involve applying some function to one or more instance variables.

The following example of a Car type illustrates the general notation that is used for class definition. Notice that this is an abstract specification and, therefore, does not mention the instance variables. They would be described as a part of the implementation.

```
Define Class Car
Superclasses: Vehicle
    Properties:
        vin: INTEGER
        color: COLOR
        owner: PERSON
        location: POSITION
    Methods:
        drive (C: CAR, P: POSITION) /* where the first parameter, C, represents self */
```

In this example, we define a class Car which is a subclass of Vehicle. Car defines four properties. The uppercase tag to the right of the colon is a type name that indicates the range of values allowed for the given property. Car also defines a single method, called drive, that takes a car and a position as arguments and changes the car's location.

Encore stores the objects of an application directly in the database, and the application maintains only references to the objects in the database, not the objects themselves nor copies of them. Reference variables are declared to be of the type of object to which it will be referring. In a traditional object-oriented programming environment, reference variables are handles to objects that reside in the application's virtual memory. With Encore, reference variables are handles to objects that reside either in the memory on the database server or in the application's memory. The database determines where an object will reside; the application does not know, or need to know, where an object is located. All objects are treated as though they are in the application's virtual memory. A reference variable, therefore, provides a transparent way of accessing an object in the database.

For concurrency control, Encore provides locking at the object level. Locks can also be attached to a programmer-defined set of related objects, called a segment, or to an ownership hierarchy. Locks are attached by invoking methods on the objects. If a method will change the makeup of a segment or an ownership hierarchy, it requests a write lock on the collection of objects to prevent others from reading or writing any part of the collection. Encore will defer the lock if any read or write locks are attached to the collection or any of its objects. If a method will change the value of a single object's property, it requests a write lock for that object. Encore will defer the write lock if there are any other write or read locks attached to the object. If a method is only reading an object's properties, then the method requests a read lock for that object. When the method is completed, it frees any locks that it requested.

Both the application program and objects in the database can send messages to the objects in the database. Each object is stored as a whole, not as multiple tuples spread across several relations. Such atomic modelling would be helpful in Intermedia where, for example, the information contained in a link object is currently stored in six relations: link, lextend, Keylink, Keyword, Linkfree, and Keyfree. With Encore, each link object could be stored once, as a unit. Furthermore, because the object exists in the database, complete with its data and the methods to act on that data, the object's data does not need to be copied into the application's data structures.

Because the data does not need to be duplicated in the application, initialization of objects is not necessary as it is when using a relational database. Instead, initialization might consist of setting up hierarchical lists of references to objects in the database (Figure 6). These lists of references are set up by sending a message to an object in the database which returns one or more object references. For example, when a document opens in Intermedia, the web object queries INGRES a number of times to retrieve the information for the blocks and links contained in that document. It uses this data to reconstruct the object hierarchy shown in Figure 3. With Encore, however, the application would send a message to the web object to return a list of references to the document's block objects, and behind the scenes in the implementation of the GetBlockList method in the Encore database, the web object would send a message to each block object to return a list of references to their links. With such a schema, the application would obtain a hierarchical list of references to the document's block and link structure. The references would then be used when the user needed to access any of this information. Initialization, therefore, is simpler because the pieces of information do not need to be copied from the database into the application's data structures.

Furthermore, the initialization process can be even simpler because an application does not have to extract complete reference hierarchies from the database and cache them in main memory for performance and computation reasons. Instead it only needs to maintain references to objects which are accessed frequently, and these references are only needed to improve performance. To act on the hierarchies already in place in the database, the application can send messages to "parent" objects. For example, in Intermedia, when a document is opened, the application would receive a reference to the document object. It would not immediately retrieve a hierarchy of references to the document's block and link objects. Instead, it would use the document reference whenever it needed to access some part of the hierarchy. Therefore, when the user selects a marker, a message would be sent to the document object to return a reference to the corresponding block object. The application would then maintain this block's reference because, as a selected block, there is a chance that the user will want to act on it. If the user then chooses to follow from this marker, a message would be sent to the block object to display a dialog box with the explainers of all the links that emanate from it. The user could then select which link to follow. This example is illustrated in the following Encore class definitions and application code:

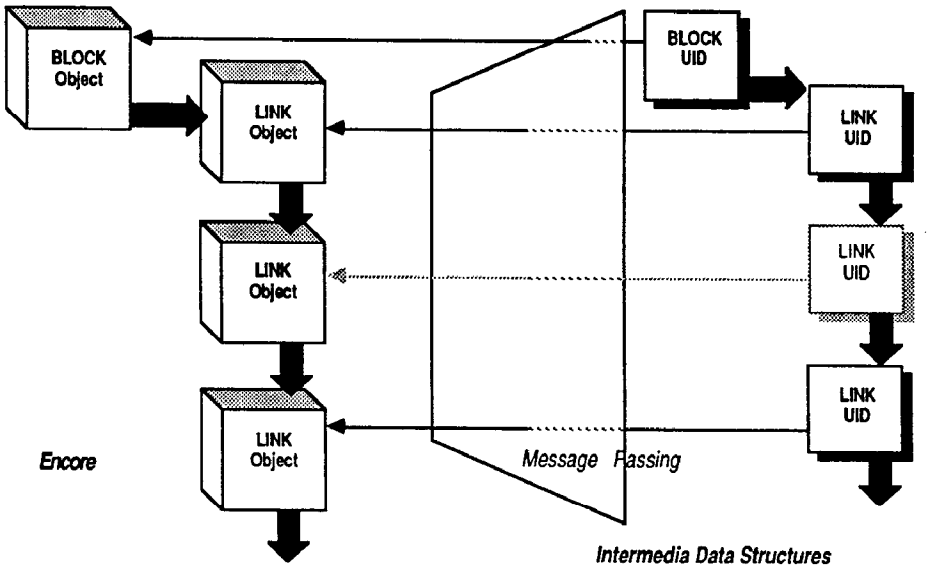

Figure 6: The data stored in Encore can be manipulated directly in Encore. Intermedia maintains references (unique IDs, or UIDs) only to those objects in the database that it needs to reference. The grayed link UID shows that the corresponding link object does not need to be referenced, but that it could be referenced if Intermedia needed to use it.

```
Define Class Block
Superclasses: Object.
    Properties:
        content : set of Object
        container_doc : Document
        marker : Marker
        explainer : Text
        create_date : Date
        modify_date : Date
        owner : User_ID
        key_words : set of Text
        links : set of Link
    Methods:
        initialize (B: Block)        /* initialize this block                                    */
        highlight (B: Block)         /* highlight the extent of this block in a document         */
        delete (B: Block)            /* delete this block and all of the links which emanate from it */
        choose_link (B: Block)       /* let the user select one of the links which emanate from B */

Define Class Link
Superclasses: Object
    Properties:
        block1 : Block
        block2 : Block
        explainer : Text
        create_date : Date
        modify_date : Date
        owner : User_ID
        key_words : set of Text
    Methods:
        initialize (L: Link)         /* initialize this link                                     */
        follow (L : Link, B : Block) /* follow this link from B to the block on the other side   */
```

In the following application code, assume that CurrentBlock is a reference to the block object corresponding to the selected marker (Figure 7a). CurrentBlock is sent the choose_link message (Figure 7b) to display a dialog box with the explainers of all the links in its list. Choose_link allows the user to select one of these links and returns DesiredLink, a reference to that link object. The follow message is then sent to DesiredLink (Figure 7c). This message finds the block at the other end of the link, causes its document to open, and returns a reference to the other block. OtherBlock is then sent a message to highlight (Figure 7d) so that the user's attention will be focused on the block at the other end of the link.

```
...
DesiredLink = choose_link (CurrentBlock);          /* Figure 7b */
OtherBlock = follow (DesiredLink, CurrentBlock);   /* Figure 7c */
highlight (OtherBlock);                            /* Figure 7d */
...
```

Note that the code for following a link in the application that uses Encore is similar to code for following a link in the application that uses INGRES (see section 6.1). The code is similar because both applications perform the command by sending messages to objects in the document's hierarchy. The difference is in the initialization of this hierarchy. With INGRES, the information contained in the hierarchy needs to be retrieved from the database, and that hierarchy then needs to be constructed. On the other hand, with Encore, this hierarchy does not need to be initially set up and "read in" to act on some member of that hierarchy. Instead, given a reference to an object, messages can be sent to it to act on the hierarchy already in place in the database. This style of interaction between an application and a database is quite different from the relational system. We will now further compare the two database systems.

6.3 Comparison

One difference between the two systems involves how much information is sent between the application and the database management system. In the relational world, an application sends a query to database which then returns a number of values. These values are typically stored in the application's data structures, manipulated in these data structures, and then shipped back to the database to be stored. In the object-oriented database world, however, a program sends a message to an object in the database to manipulate its data, compute some value, and/or to return something, such as a reference to another object in the database.

One problem that we encountered with the relational system is that the flat nature of relations made it difficult to model our complex object networks. Because the object-oriented ownership hierarchies could not be represented directly, we needed to mimic the hierarchies with identification keys instead of pointers or direct references. As illustrated earlier, objects have to be represented in a series of relations to keep the information in normal form so that there are no embedded structures.

Furthermore, because complex objects are stored in a series of relations, a number of queries need to be

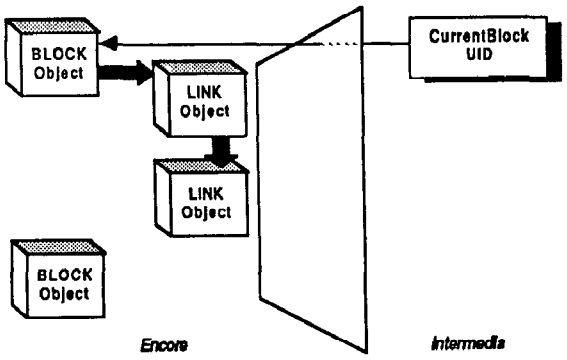

Figure 7a: There is a reference to the current block object.

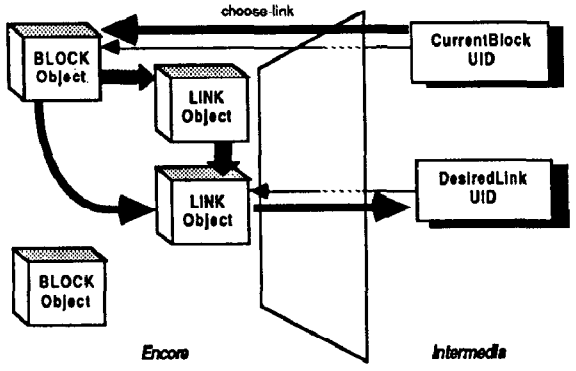

Figure 7b: The choose-link message is sent to the object referenced by CurrentBlock. CurrentBlock displays the link explainers to the user and returns DesiredLink, a reference to the link selected by the user.

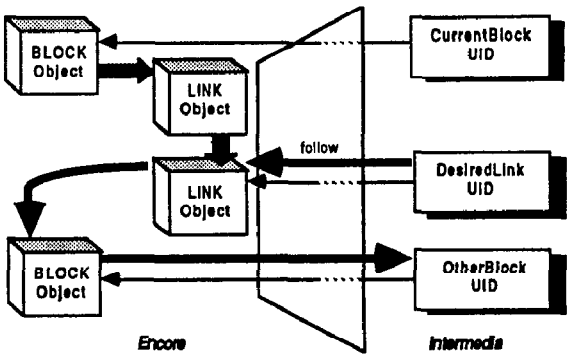

Figure 7c: The follow message is sent to the object referenced by DesiredLink. It performs the follow command and returns OtherBlock, a reference to the block object at the other end of the link.

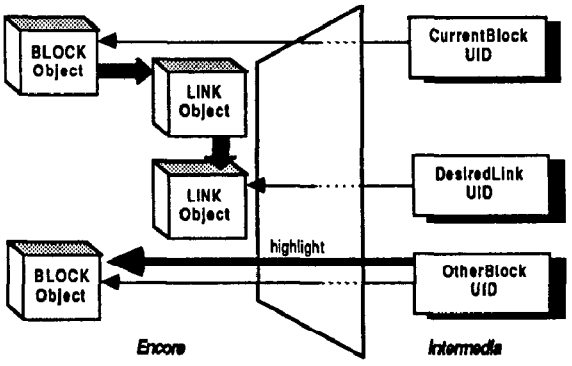

Figure 7d: The highlight message is sent to the object referenced by otherBlock.

applied to retrieve a single object from the database. These queries are applied sequentially, with one query depending on the result of a previous query. In fact, when an Intermedia document object is initialized, 2 + (3 * number of blocks in the document) + (2 * number of links attached to these blocks) queries are performed. The database does not know about the global request, only about the individual queries, and therefore cannot reorder the queries to optimize them. Also, interactions with the database system are expensive, especially when the database resides on a shared server in a distributed environment. Because of communication costs and inherent database overhead, performing a number of queries is slower than doing just one query.

With an object-oriented database, sending one message can take the place of performing many relational database queries. A message can perform computations and send messages to other objects before returning a result. In other words, messages can invoke other messages, thereby alleviating the problem of sequential queries found in the relational world.

By extracting values from INGRES tuples to store in instance variables of objects, at times we encountered the problem of impedance mismatch between the programming language data structures and the database data structures. For example, Intermedia explainers are variable-length character strings, but INGRES only allows for fixed-length character strings. We had to introduce another relation, the LExtend relation, to store a sequence of thirty-character substrings for each explainer. The object-oriented system helps avoid the problem of impedance with the style of interaction it provides between the database and the application program. Because Inheritance C and Encore use an object-oriented type system, there would be much less of a mismatch. Ultimately, we believe that object-oriented databases will evolve into object-oriented database programming languages. In such a language, there would be no difference between the two type systems.

With relational systems, type checking is only performed when changes are committed to the database. Illegal values and types can be assigned to the application program structure representing the database objects, and these problems will not be caught until much later when the changes are committed to the database. While the application software can perform data integrity checking, it is easier and less error-prone to let the database take care of it. For example, to save a document for the first time, a user

must type a file name into a dialog box. Because the user might type illegal characters into the name, we need to explicitly perform value checking on it. Furthermore, we should perform such checking every place we assign a value to the document's fTitle field (i.e., every place that the user can change the document's name). However, if we did not want to do checking on the fly, we could let INGRES perform checking when the changes were committed to the database. Unfortunately, many computations could be performed with an illegal value before the database could catch the problem. In contrast, an object-oriented database can automatically perform integrity checking when the change occurs because the change takes place in the database. An object-oriented database can perform run-time type checking at each point of use (i.e., each method call).

The two systems provide a similar concurrency control scheme. Each provides read and write locking at the granularity of either a record or an object and write aggregate locking at a granularity of either a relation or a segment. However, there is a difference between the two systems when it comes to locking information contained in an ownership hierarchy[3]. In a relational system, because hierarchies are represented across a number of relations, the application needs to explicitly lock each record in each relation to lock the hierarchy. In Encore, however, the application can lock a pre-defined containment hierarchy with a single operation.

7. FUTURE RESEARCH

We feel that an object-oriented database can successfully meet the data handling requirements of Intermedia that are currently met by INGRES. Furthermore, we might be able to use an object-oriented database in ways that we were not able to use the relational system. For example, we might want to use Encore to store all of the persistent data, like text and graphics objects, as opposed to just the block and link objects. One of the reasons we chose not to store the editor-specific persistent data in INGRES was that we did not want to have to flatten the complex document data model into relations. However, with Encore, this flattening process would not have to take place. Persistent objects would be stored directly in the database as objects, not as a series of relations. The objects would retain their direct references to other objects, and the hierarchies would be stored in a form that could be understood by the database. Facilitating the storage and access of object hierarchies would reduce program development time and would provide a consistent storage mechanism for Intermedia's editors.

In addition to traditional database storage activities, Encore provides versioning of the objects that it contains. Also, because classes are objects, Encore stores version histories that track the evolution of classes. A scheme for minimizing the impact of type changes on existing applications has been proposed [Skar86a]. In the future we would like to investigate how Intermedia might benefit from version management. One possibility is to allow users to open previous versions of documents or webs. Intermedia developers might also benefit by using Encore as a version control mechanism for classes and methods. In this case, Encore would be an integral part of the program development environment.

8. CONCLUSION

The relational database model was created to provide a simple abstraction that allowed the representation of large classes of data using a small set of principles. Likewise, the object-oriented model of programming was designed to allow the creation and representation of complex data structures in a coherent and uniform way. We have attempted to look at how the relational database model fits into this object-oriented programming paradigm by examining the data-handling needs of Intermedia, a large object-oriented system. We explored replacing a relational database with an object-oriented database, which much more closely models the hierarchies present in the Intermedia system. Certainly, this study does not presume to have covered all the potential uses of relational systems, nor all the comparisons such systems have with their object-oriented counterparts. Rather, we hope that this study has shed some light in a concrete fashion on the fundamental differences between the relational and object-oriented database architectures.

9. ACKNOWLEDGEMENTS

We would like to thank the past and present members of the Scholar's Workstation Group at IRIS who have worked long and hard at designing and implementing Intermedia. In particular, Page Elmore designed and implemented most of the initial relational database support. Thanks also to the members of the Systems Group at IRIS for providing the necessary operating system and hardware support for Intermedia. The Intermedia project was sponsored in part through grants by the Annenberg/CPB Project and a joint study contract with IBM. In addition, we thank Apple Computer, Cadmus Computer, and Bolt Beranek and Newman for their assistance in making available key software.

We would also like to thank Andrea Skarra, Mark Hornick, and the other members of the Encore development team for their help in designing and implementing Encore. The Encore project was supported in part by the National Science Foundation under grant DCR 8605597, by IBM under contract 55917 and amendment contract 643513, by the Office of Naval

[3]The university INGRES does not provide transactions and, therefore, does not provide locking. However, for comparison purposes, assume that this INGRES does provide transactions and locking.

Research under contract N00014-86-K-0621, and by DARPA under ONR contract N00014-83-K-0146.

Special thanks to Helen DeAndrade for designing the figures in this paper and to Norm Meyrowitz, Nan Garrett, and Tim Catlin for their extensive comments and guidance.

10. REFERENCES

[Bane87] J. Banerjee, H. Chou, J. Garza, W. Kim, D. Woelk, "Data Model Issues for Object-Oriented Applications," ACM Transactions on Office Information Systems, April, 1987.

[Codd79 E. Codd, "Extending the Database Relational Model to Capture More Meaning," ACM Transactions on Database Systems, December, 1979.

[Conk86] J. Conklin, "A Survey of Hypertext," MCC Technical Report, No. STP-356-86, Austin, TX, October, 1986.

[Doyl86] K. Doyle, B. Haynes, M. Lentczner, L. Rosenstein, "An Object Oriented Approach to Macintosh Application Development," Proceedings of the 3rd Working Session on Object Oriented Languages, Paris, France, January, 1986.

[Enge68] D. Engelbart and W. English, "A Research Center for Augmenting Human Intellect," Proceedings of FJCC, Vol. 33, No. 1, AFIPS Press, Montvale, NJ, Fall 1968.

[Garr86a] L. Garrett, K. Smith, N. Meyrowitz, "Intermedia: Issues, Strategies, and Tactics in the Design of a Hypermedia Document System," Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW '86), Austin, TX, December, 1986.

[Garr86b] L. Garrett and K. Smith, "Building a Timeline Editor from Prefab Parts: The Architecture of an Object-Oriented Application," Proceedings of OOPSLA '86, Portland, OR, September, 1986.

[Gold83] A. Goldberg and D. Robson, Smalltalk: The Language and Its Implementation, Addison-Wesley, 1983.

[Khos86] S. Khoshafian and G. Copeland, "Object Identity," Proceedings of OOPSLA '86, Portland, OR, September, 1986.

[Maie86a] D. Maier, et al, "Development of an Object-Oriented DBMS," Proceedings of OOPSLA '86, Portland, OR, September, 1986.

[Maie86b] D. Maier and J. Stein, "Indexing in an Object-Oriented DBMS," Technical Report CS/E-86-006, Oregon Graduate Center, Beaverton, OR, May, 1986.

[Meyr85a] N. Meyrowitz, et al, "The Intermedia System – A Software Framework and Applications for Education and Research: Requirements, User Interface, and Systems Design," Institute for Research in Information and Scholarship, Providence, RI, September, 1985.

[Meyr85b] N. Meyrowitz, "Inheritance C Report," IRIS Technical Report 85-7, Institute for Research in Information and Scholarship, Providence, RI, July, 1985.

[Meyr86] N. Meyrowitz, "Intermedia: The Architecture and Construction of an Object-Oriented Hypermedia System and Applications Framework," Proceedings of OOPSLA '86, Portland, OR, September, 1986.

[Nels80] T.H. Nelson, "Replacing the Printed Word: A Complete Literary System," Information Processing 80, S.H. Lavington (ed.), North-Holland Publishing Company, IFIP,1980.

[Nodi85] M. Nodine, personal communication, Bolt Beranek and Newman, Cambridge, MA, 1985.

[OBri86] P. O'Brien, B. Bullis, and Craig Schaffert, "Persistent and Shared Objects in Trellis/Owl," 1986 International Workshop on Object-Oriented Database Systems, Pacific Grove, CA, September, 1986.

[Oren87] J. Orenstein, D. Goldhirsh, F. Manola, Computer Corporation of America, Internal Report, Cambridge, MA, 1987.

[Schm86] K. Schmucker, Object-Oriented Programming for the Macintosh, Hayden Book Company, 1986.

[Skar86a] A. Skarra and S. Zdonik, "The Management of Changing Types in an Object-Oriented Database," Proceedings of OOPSLA '86, Portland, OR, September,1986.

[Skar86b] A. Skarra, S. Zdonik, S. Reiss, "An Object-Server for an Object-Oriented Database System," 1986 International Workshop on Object-Oriented Database Systems, Pacific Grove, CA, September, 1986.

[Smit87] K. Smith, "A Developer's Guide to Intermedia," Institute for Research in Information and Scholarship, Providence, RI, 1987.

[Ston76] M. Stonebraker, E. Wong, P. Kreps, G. Held, "The Design and Implementation of INGRES," ACM Transactions on Database Systems, September, 1976.

[Yank85] N. Yankelovich, N. Meyrowitz, A. van Dam, "Reading and Writing the Electronic Book," IEEE Computer, October, 1985.

[Zdon84] S. Zdonik, "Object Management System Concepts," Proceedings of the Conference on Office Information Systems, ACM/SIGOA, Toronto, Canada, June, 1984.

[Zdon86] S. Zdonik and P. Wegner, "Language and Methodology for Object-Oriented Database Environments," Proceedings of the Nineteenth Annual Hawaii International Conference on System Sciences, January, 1986.