AN IMPLEMENTATION OF A MULTIMEDIA OBJECT-ORIENTED DATABASE MANAGEMENT SYSTEM

Milko Marinov, Dimitar Radev

Abstract: *The object-oriented paradigm represents one of the most successful paradigms in many areas of computer science. It has gained wide acceptance as a unifying paradigm for the design of database systems, programming languages, and artificial intelligence area over the last decade. In this work the characteristics of the object model of data are presented. The general characteristics are described of the object-oriented database management system CODAS implemented as a library of C++ classes. The major requirements to the file architecture are formulated. The organisation of the database nodes and the structures of the object and index files are presented in details.*

Key words: *Object data model; Database management system; File architecture; Persistent objects.*

INTRODUCTION

Object-oriented programming does not change the traditional requirements to the database management system (DBMS). It just augments programming approaches with better ways of representing data inside certain applications [2,6].

The relational model represents data in the form of relations or tables with rows and columns. Those relations are interpreted as files in the database. The rows are the separate records of the file and the columns - the fields of the records. The relational model assumes that the file contains records of fixed size and format. Data elements of variable length are not allowed as well as repeating data elements or abstract data types in the record definition [1,3,4]. This imposes and alternative to the relational data model to be searched for. One of the attractive aspects of the object-oriented paradigm is the possibility for creating abstract data types. Besides, a lot of applications use databases with objects of variable size (CAD/CAM systems, geographic systems, multimedia sound and text data flows, electronic mail, some text processing systems) [2,5,6,7].

These applications make many demands on conventional database technology, including the ability to model very complex data and the ability to evolve without disruptive effects on the current application base. Object-oriented multimedia databases address both sources of complexity by including facilities to manage the software-engineering process (data abstraction and inheritance) and features for capturing more directly some of the interconnections and constraints in the data [2,3,7]. Objects in a multimedia database have various properties and participate in a number of relationships with other objects. In a typical database the bits and bytes usually represent numbers and text which are interpreted by the database engines. Most database mechanisms do not have a built-in means to display graphics, video or play sound.

The present paper considers an architecture and implementation of a multimedia object-oriented database management system. The paper is structured as follows. Section 2 provides some specifications of the general features and architecture of the CODAS object-oriented database management system. Section 3 describes the organization of data. Section 4 presents the management of persistent objects in CODAS. Section 5 summarises the authors' contributions and their future research intentions.

GENERAL CHARACTERISTICS AND CODAS ARCHITECTURE

The object model is a first attempt to relate the program model to the database model since the object-oriented design represents abstract data types in ways that no traditional data model supports well. The object-oriented design needs its own database model. The objects can be stored into and then restored from the area of their storage which survives the existence of the objects.

CODAS is an object-oriented database management system implemented as a class library of C++. It supports an object database with a relatively simple interface to the classes. In some aspects its data model includes the characteristics of the relational model while in others it adopts the properties of objects in C++. The first objective of a persistent object database manager is the identification of a method to describe the using class's cooperation in a way that is easy for programmers.

The traditional databases are less sophisticated than multimedia databases. The differences between a traditional database and multimedia database are not only the structure represented in the organization of the data, but also the tools that are part of the database. Modern databases act more like containers, allowing a programmer to impose a structure on data and they also provide the tools to manipulate that data. A multimedia database should contain tools and a structure. The CODAS multimedia database management system interprets certain kinds of data and can invoke the appropriate mechanism to allow a viewer to see a particular kind of media. This kind of structuring mechanism allows for a particularly rich and powerful way to traverse large amounts of data with relative facility. A true multimedia database should have retrieval capabilities for all of the media types that it supports. A technology for carrying out picture, sound and video searching should be developed. The basic characteristics defining a multimedia database are as follows:

- ***Data types to be stored:*** numeric, text, formatted text, graphics, sound files, video files;
- ***Container types:*** pages, forms, frames, records;
- ***Viewer for:*** numeric data, text data, formatted data, graphics files, sound files, video files, pages, frames, records;
- ***Links between container types***;
- ***Move between container types***;
- ***Searchers for container types***;
- ***Report generator for multimedia database elements***.

A database is a kind of container for data. The page container is one that is largely unstructured and provides little organisation to the data contained in it. Records in a traditional database consist of fields containing data. A record contains a group of related data. Records in multimedia database will hold sequences of multimedia information. Most multimedia databases include a tool called a report generator. It is the means by which data is retrieved from the database and organised in various desirable forms. A report generator in a multimedia database would allow the display of sequences of multimedia information in various formats. The format could be a hypermedia format (traversing links between containers), a sequence of multimedia or an overview of the database contents. Each media type has an appropriate viewer associated with it - a mechanism that handles the display or the rendering of a medium on a particular output device. A multimedia report is defined as a static representation of multimedia information contained in the media database. In a traditional report generator a user will prepare a layout template. This template is a visual representation of what the report eventually produced by the system will look like. The report generator allows for complex presentation sequences that are appropriate for multimedia. The report could be interactive. An interactive report is one that would configure itself according to events occurring as a result of viewing the report. Such events include mouse click or responses to queries during the report presentation. In this mode the system could be used to present certain information sequences as desired by the author.

To make use of the sequences of records in the multimedia database two kinds of mechanisms are defined: viewers and loaders. A viewer is used to display a particular kind of media. For each media type there is a viewer to display the media. In an object-oriented paradigm the media type (text, graphic, video or sound) is an object and the object can

receive a message whose name is ***view***. In this sense the viewer belongs to the data type. Retrieving specific kinds of media in a large multimedia database can be a complex task. This task can be facilitated by a viewer giving the user a view of the complete multimedia database. A media type must be loaded before it is viewed. The purpose of a loader is to prepare a media type for viewing or manipulation. A specific loader is associated with each media type. Loaders move media files from their particular sources into the multimedia database. A loader may not be physically separate from a viewer. In fact it may be part of a viewer. The difference between the loader and the viewer is that a loader will process the compressed media, expanding it into a viewable form. Another function of loaders is to act as a means to store media in a multimedia database.

The architecture of the CODAS database management system is shown in figure 1.

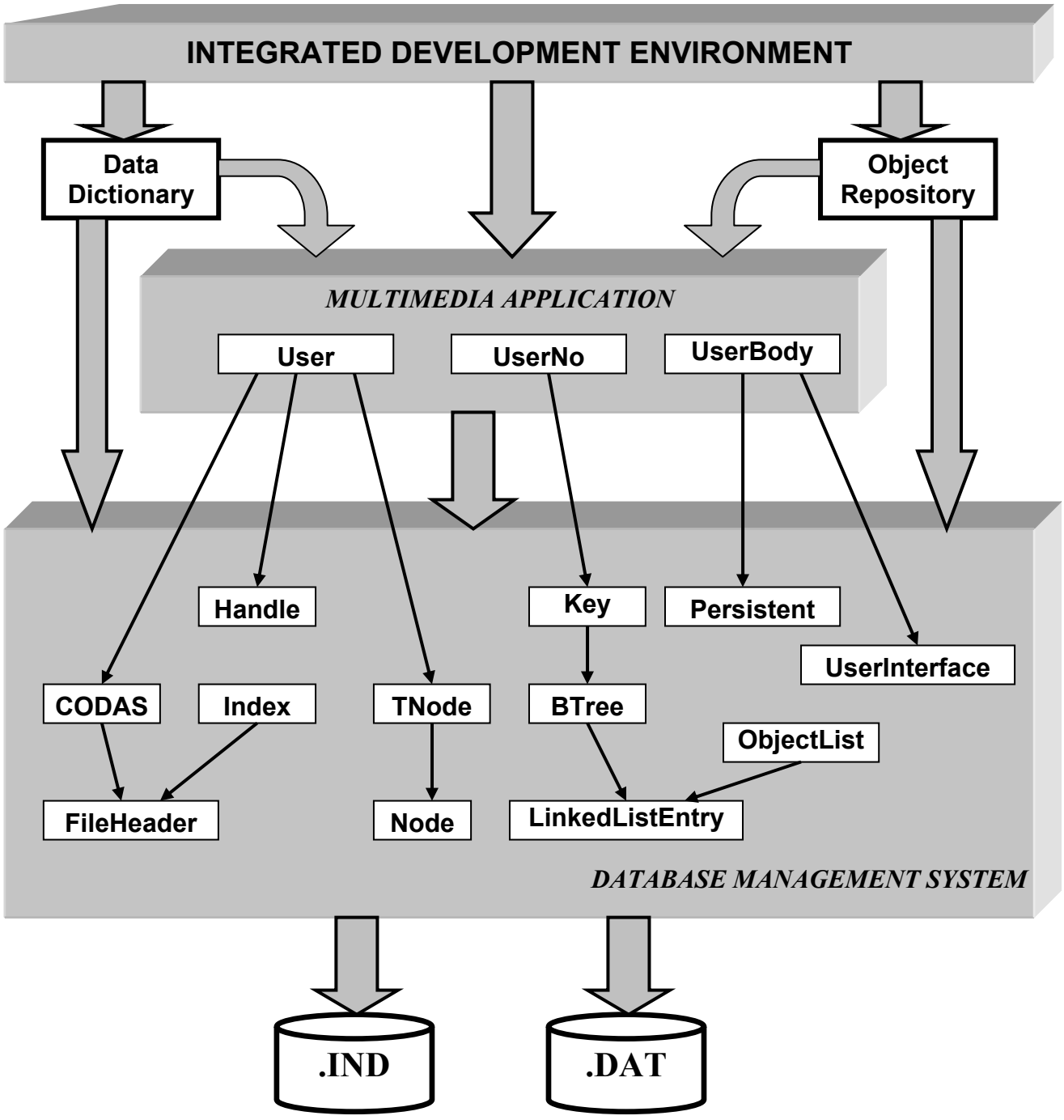

Fig. 1. The architecture of CODAS database management system

ORGANISATION OF DATA

The definition of multiple related and persistent classes is in fact equivalent to developing the database schema which describes the contents of the files. Most of the traditional DBMS use a special language for defining the data in describing the schema of the database. CODAS makes use of the class definitions in C++. Thus not just the description of the data members is encapsulated in each class but also the integration of the class with the CODAS DBMS. The database supported by CODAS consists of two files: the data file comprising the objects and an index file, comprising the key indexes to the objects. The data file has a name extension ".DAT", while the index file has an extension ".IND". The file architecture supports:

- persistent objects of variable length;
- indexes into and objects of multiple classes;
- persistent objects which can increase or reduce their lengths or can be deleted in the time interval between their declaration and destruction in the program;
- repeated usage of the freed file space.

To support those objectives CODAS uses a system of nodes in the file.

Both files in the CODAS database are managed through a system for including/excluding nodes into or out of the file. The nodes have a fixed length and are linked in a chain. The file contains a header record followed by records of the nodes (Fig.2).

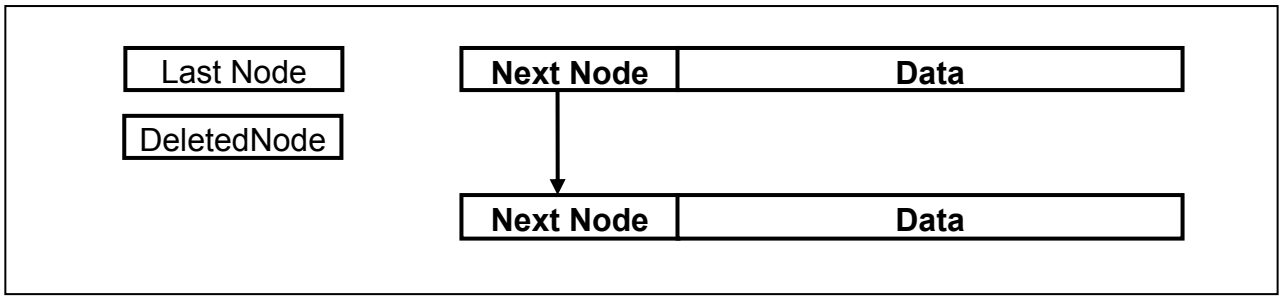

Fig. 2. The structure of nodes in the database

The nodes are logically addressed by numbers starting from 1. CODAS uses the system of nodes in two ways. The index files are organised from nodes of fixed length which inherit the behaviour of the *Node* class. The objects are unformatted streams of variable length data which can be extended beyond the boundaries of the node. The header record contains the numbers of two nodes. The number one node points to the node with the highest number being allocated. The number two node points to the first node in the list of the deleted nodes. The header records are implemented as objects of the *FileHeader* class. Each node record is a fixed length record on the disk, containing the number of the node at the start of the record followed by the data of the record. The number of the record is a chain pointer. It points to the next node in the chain of logical nodes. The nodes are implemented as objects of the class *Node*.

CODAS stores the objects in a chain of nodes. Each object starts at a new node and is addressed through the node number. The address is comprised of two integers, the first identifying the class of the object and the second - the logical number of the node. In case the object cannot fit into one node, the object overflows into the next node. Thus new nodes are added to the chain until the entire object is recorded. If needed, the last node of the chain is completed with zeros. Figure 3 presents the connection between the index file and the object file. The objects of all the different classes of the database are contained in

physically the same file. Every class may produce objects of different size and format, respectively the objects of one class may have a variable length.

The object file does not make any sense without the definitions of the class and its methods. No information is contained in the file to identify the types of objects in the data base or their format. In order to find the exact location of an object, CODAS needs the address of the first node of the object. This address is provided by the index file.

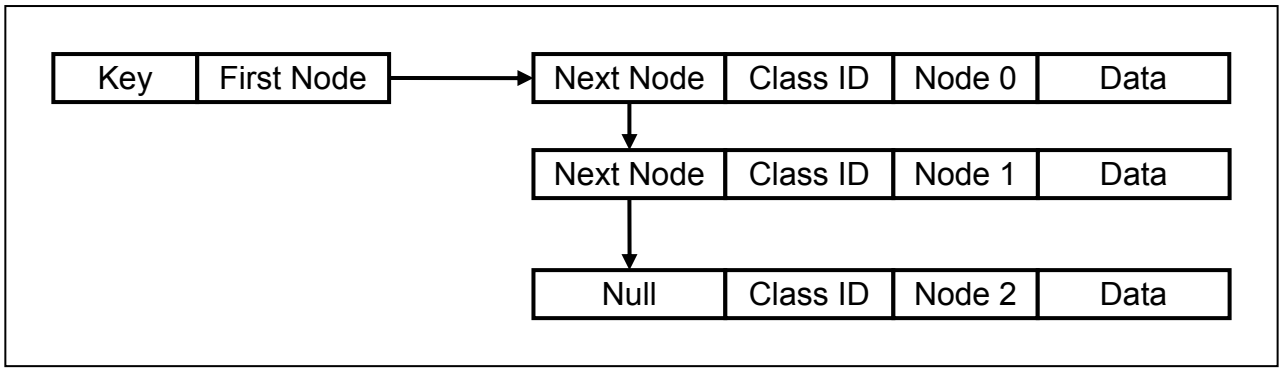

Fig. 3. The connection between the index file and the object file

The index file of the database supports the indexes which give the link between the key values and the address of the object node in the object file. For every primary and secondary key in the persistent classes one logical index table is stored in an index file. The classes are represented in the file by a table of index header records. Each successive node in the chain is the header table for a class of related identification, starting from zero. CODAS uses B-trees to implement the indexes.

MANAGEMENT OF PERSISTENT OBJECTS

The design and implementation of the mechanisms which support persistence must address issues such as how to name an existing (persistent) object, from both a user and system-level point of view, and how to ensure that the state of an object is stored in a stable storage. Any class may be made persistent by inheriting from the persistent library classes. The persistent object and key member classes derive from abstract base classes defined in the CODAS class library. Objects of the persistent class types cooperate in their own persistence by calling CODAS member functions at specific times. They also provide member functions that CODAS can call to perform tasks that require knowledge of the content and format of the persistent object. The flow of function calls to load and save an object is presented in figure 4.

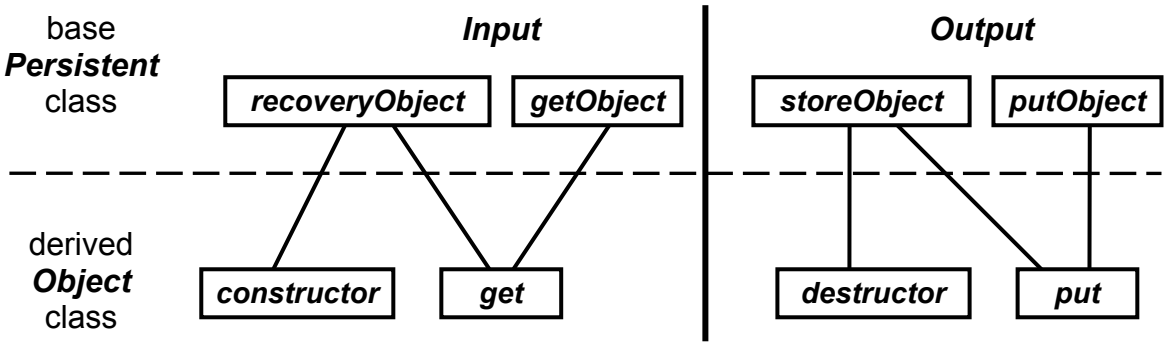

Fig. 4. The maintenance of persistent objects in CODAS

To load an object, the derived class constructor

- calls the ***recoveryObject*** function in the base ***Persistent*** class,
- the base ***Persistent*** class calls the ***get*** function in the derived class;
- the derived class makes several calls to the ***getObject*** function in the base class.

To save an object, the derived class destuctor

- calls ***storeObject*** in the base class;
- the base class calls the ***put*** function in the derived class;
- the derived class makes several calls to ***putObject*** in the base class.

The user responsibilities in persistent class design are to provide the derived class constructor and destructor and its ***get*** and ***put*** functions.

CONCLUSION

The suggested file architecture of the CODAS system combines the advantages of the relational and the object data model. It supports a persistent object database through a comparatively simple interface to the classes. Through the efficient organisation of the index file a fast access to the data is provided, as well as storing objects of variable length and rational utilisation of the disk space. CODAS uses B-trees in implementing the index file. This leads to a relative retardation of the system performance in updating the database.

The implementation of the CODAS class library provides flexibility for use and maintenance of persistent objects. Persistent objects can be added, retrieved, changed and deleted. CODAS maintains the integrity of the interclass relationships and its actions can be monitored.

Further efforts of the authors will be aimed at refining the user interface and automating the activities for the development of applications by means of CODAS.

REFERENCES

[1] Alhajj, R., F. Polat, Rule-based schema evolution in object-oriented databases, Knowledge-based systems, Vol. 16, pp. 47-57, 2003.

[2] Graham, I. Object-oriented methods. Addison-Wesley, 1993.

[3] Nierstrasz, O., S. Gibbs, D. Tsichritzis, Component-oriented software development, Communication of the ACM, Vol. 35, No 9, pp. 160-165, 1992.

[4] Preuner, G., St. Conrad, M. Schrefl, View Integration of Behavior in Object-oriented Databases, Data & Knowledge Engineering, Vol. 36 (2), pp.153-183, 2001.

[5] Soutou, C., Modeling Relationships in Object-relational Databases, Data & Knowledge Engineering, Vol. 36, pp.79-107, 2001.

[6] Stevens, Al, C++ database development, MIS:press, 1992.

[7] Yoshitaka, A., T. Ichikawa, A Survey on Content-based Retrieval for Multimedia Databases, IEEE Transactions on Knowledge and Data Engineering, Vol. 11, No 1, pp. 81-92, 1999.

ABOUT THE AUTHORS

Assist. Prof. Milko Marinov, PhD, Department of Computer Systems & Technologies, University of Rousse, Phone: (+359 82) 888 356, E-mail: MMarinov@ecs.ru.acad.bg.

Assoc Prof. Dimitar Radev, PhD, Department of Communication Equipment & Technologies, University of Rousse, Phone: (+359 82) 888 841, E-mail: DRadev@abv.bg.