Object-Oriented Databases with Applications to CASE, Networks, and VLSI CAD

Object-Oriented Databases with Applications to CASE, Networks, and VLSI CAD

Editors:

Rajiv Gupta
GE Corporate Research and Development

Ellis Horowitz
University of Southern California

Prentice Hall Series in Data and Knowledge Base Systems
Dennis McLeod, Series Editor

Prentice Hall, Englewood Cliffs, NJ 07632

Library of Congress Cataloging-in-Publication Data

Object-oriented databases with applications to CASE, networks, and VLSI CAD / edited by Rajiv Gupta and Ellis Horowitz.
p. cm.
Includes bibliographical references and index.
ISBN 0-13-629833-8
1. Object-oriented data bases. 2. Computer-aided software engineering. 3. Computer networks. 4. Integrated circuits--Very large scale integration--Design and construction--Data processing. 5. Computer-aided design. I. Gupta, Rajiv. II. Horowitz, Ellis.
QA76.9.D3026 1991
005.75--dc20 90-39302

Editorial/production supervision and
interior design: Jennifer Wenzel
Manufacturing buyers: Linda Behrens and Patrice Fraccio

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN 0-13-629833-8

Prentice-Hall International (UK) Limited, *London*
Prentice-Hall of Australia Pty. Limited, *Sidney*
Prentice-Hall Canada Inc., *Toronto*
Prentice-Hall Hispanoamericana, S. A., *Mexico*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Simon & Schuster Asia Pte. Ltd., *Singapore*
Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*

Preface

Object-oriented has become one of the most important buzzwords in the computer science arena. This is no accident. To a practitioner in software engineering the term promises a substantial body of proven concepts such as data abstraction, encapsulation, inheritance, polymorphism, extensibility, generic programming, information hiding, code reusability, modularity, exception handling, and so on. The list goes on, if only because there is no clear consensus on "what" is object-oriented programming.

One stream of research in the object-oriented field is object-oriented *databases*, (abbreviated here as OODBs). OODBs, which are a radical departure from conventional data structuring paradigms, are the primary focus of this book. Since language, database, and software engineering issues are intricately intertwined, a few chapters in the book are devoted to object-oriented languages. However, the overall view is distinctly database oriented.

Relational database technology has dominated the database field for the past decade, and more. There are now many commercially available relational systems. Researchers have long recognized the limitations of relational systems in managing data for applications such as CAD CAM, office information systems, and CASE. The incorporation of the object paradigm in database management systems is a direct consequence of this realization.

Many journal articles have been published that deal with numerous aspects of this new class of data management software. However, "real" object-oriented databases are just now appearing and several others are about to make their debut. In the absence of any torch bearers when it comes to actual software, and any book that presents the big picture, buzzwords abound and the Tower of Babel syndrome is clearly discernible. It is our ambition in this book to take some of the "buzz" out of these buzzwords that are so prevalent in this fast-breaking field and expose the concepts behind them.

The purpose of this book is fourfold.

1. It provides the reader with a perspective on various object-oriented concepts. These concepts, which have been under development in a diverse set of fields such as artificial intelligence, database theory, programming languages and compiler theory, form the backbone of the object paradigm. Illustration of their power, implementation, and use is a primary objective of this book. This book presents several issues of practical importance which are still in research/development stages. Issues such as benchmarking, schema evolution through learning, and incorporating and promoting the object paradigm in a corporation, are covered.
2. The book provides the reader an overview of existing OODBs, including examples of their use and comparison of their strengths and weaknesses. Besides a survey, three OODBs are selected for closer scrutiny. We believe that these three databases adequately represent the three evolutionary tracks taken by OODB research. As the reader will find out, the core concepts embodied in most OODBs are very similar. Instead of restating the same concepts over and over, the book presents three different facets of these databases.
3. The book provides a series of real-world examples and shows how they are mapped onto an object-oriented framework. The idea is to show the concepts in action. A diverse set of application areas is chosen so as to exemplify the universal applicability of the object paradigm.
4. C++, even though it is not an OODB, has been a prime mover in this field. Several yet-to-be-released OODBs have either adopted C++ as their primary data manipulation language, or are implementing object persistence directly into the language. The book presents a brief overview of the language, its power and limitations. It then presents two quite different approaches to *making a database out of* C++.

A number of object-oriented systems are covered in this book. These include Vbase (TDL and COP), GemStone and OPAL, Statice, ONTOS, IRIS, ORION, ODE, SIM, and C++. We do not expect the reader to be familiar with any of these systems. The overall coverage, though intentionally detailed at places, is quite self-contained. The book, however, does assume a degree of maturity about the "software-in-the-large" and knowledge of any one high-level language.

Advice to the Reader

The reader of this book will doubtlessly be someone who is a computer professional. Let us first assume that the reader has had no exposure to object-oriented concepts, but has an interest and knowledge of databases. He is advised to "begin at the beginning". Part I introduces the main concepts of object-oriented programming and object-oriented databases. For a reader just starting out in this field, the lay of the land is important. For such a reader, of special merit would be the articles by Gupta and Horowitz, McLeod, and Berre as they assume a beginning reader. In Part II the "Overview" article by Horowitz and Wan also would be appropriate for this person. Afterwards the reader might either move into the more in-depth material on Statice, Vbase and GemStone, or move to the applications of Part III.

Alternately, let us assume that our reader is interested in using an OODB for an application. Readers vho have a specific interest, such as VLSI CAD, CASE or network management might begin with the surveys in Part I and II and then go to the article in Part III that deals specifically with their application. Here he can read about several serious attempts to use OODBs, the successes and the failures. He might then go to Part IV and examine the latest attempts to add persistence to C++.

Acknowledgments

This book would not have been possible without the wisdom and cooperation of the contributing authors. Special thanks go to the staff at Ontologic, Inc. for providing us with software and documentation which is eminently useful. Thanks also go to our editors at Prentice Hall. Special thanks to Dan Olson for providing help with PHtex macros. The University of Southern California has been a stimulating environment for writing this book.

Rajiv Gupta and Ellis Horowitz

To my wife, Lupe,
and to Samantha—the greatest thing that happened to us since this book was begun—with a request.

"Child! do not throw this book about;
Refrain from the unholy pleasure
Of cutting all the pictures out!
Preserve it as your chiefest treasure"
[Hilaire Belloc, Bad Child's Book of Beasts]

Rajiv Gupta

To Ira Horowitz,
the object of my parental affection.

Ellis Horowitz

About the Editors

RAJIV GUPTA is a computer scientist at the General Electric Corporate Research and Development Center, Schenectady. Prior to his position at GE, he was a Research Assistant Professor of Electrical Engineering—Systems at the University of Southern California, Los Angeles from 1988 to 1990. Dr. Gupta served as a Post Doctoral Research Fellow in the VLSI Test Group at USC from July 1987 to June 1988, and as a Junior Software Engineer at Tata Burroughs Ltd, Bombay from March 1982 to July 1982.

Dr. Gupta received his B.E. (Hons.) degree in Electrical and Electronics Engineering and his M.Sc. (Hons.) degree in Physics from Birla Institute of Technology and Science, Pilani, India, both in 1982. He received his M.S. and Ph.D. degrees in Computer Science from the State University of New York at Stony Brook, where he was a Research Assistant from August 1982 to July 1987.

Dr. Gupta's primary research interests include object-oriented frameworks for VLSI CAD, VLSI design and testing, system-level fault diagnosis and reconfiguration, and fault-tolerant computing. He is currently investigating databases for organizing and integrating CAD information. As an experimental testbed for integrating VLSI CAD tools, he has designed and implemented a prototype framework called Chase. Dr. Gupta has published numerous research articles on subjects ranging from system reconfiguration, fault diagnosis, VLSI testing, and logic programming to object-oriented databases for VLSI CAD.

Dr. Gupta is a member of IEEE Computer Society and a recipient of the National Science Talent Search Scholarship awarded by the Government of India.

ELLIS HOROWITZ received his B.S. degree from Brooklyn College and his Ph.D. in computer science from the University of Wisconsin. He was on the faculty there and at Cornell University before assuming his present post as Professor of Computer Science and Electrical Engineering at the University of Southern California. He has also been a visiting Professor at M.I.T. and the Israel Institute of Technology (Technion). Dr. Horowitz has held numerous administrative jobs including Associate Chairman of Computer Science at the University of Wisconsin and at U.S.C. He was also acting chairman of the Computer Science Department at U.S.C. He is the author of six books and over sixty research articles on computer science subjects ranging from data structures, algorithms, and software design to computer science education. Dr. Horowitz has been a principal investigator on research contracts from NSF, AFOSR, ONR, and DARPA. He is a past associate editor for the journals *Communications of the ACM* and *Transactions on Mathematical Software*. He currently serves on the technical advisory boards of several software companies and is an IBM scholar.

Contents

1 A Guide to the OODB Landscape

Rajiv Gupta and Ellis Horowitz

1.1 INTRODUCTION

In the wake of the Falkland Islands war it was reported that the British destroyer HMS Sheffield was sunk on high seas because the database in its warning system was incorrectly programmed. Gemini V landed miles away from its designated landing point because of a flaw in its guidance program. Stories about false alarms from North American Aerospace Defense Command because of situations unanticipated by the C^3 software are disquietingly common in software engineering folklore. What is described as the "software crisis" is quite real. The above mishaps, which can all be traced to buggy software, would convince even the most optimistic practitioner of the craft that present day large-scale programs can at best be regarded as fragile contraptions [Lin85].

Development of high-quality, large-scale software is widely acknowledged as a tough undertaking. Design methodologies that promise to improve the quality of software—notwithstanding the debate over how to measure software quality in terms of correctness, robustness, extensibility, ease of integration, reuse and maintenance—are of great interest to the design community.

Of the current software design methodologies, top-down decomposition of the function to be implemented, with stepwise refinement, is by far the most widely practiced. This design methodology starts with a very high-level description of the problem or task, and gradually refines it into a set of subtasks until the problem is reduced to a sequence of manageable pieces. This leads to a tree-like decomposition of the original problem.

Experience over the past three decades has shown that this paradigm for software construction leads to quick development of *a* program for the task at hand as it concentrates on *what* needs to be done. However, the resulting programs are less maintainable as the methodology concentrates on a very chameleonic aspect of the program, viz., its function. Over the lifetime of the program, the original function invariably changes and becomes one of the many functions provided by the program. Also, the tree-like decomposition, based on a premise that will change over time, divides the program along procedural lines with data structures and future reusability being second-order concerns. A number of empirical studies have shown (see, for example [Boehm81]) that in such an environment the effort required in constructing large-scale software increases rapidly with the size of the program.

1.1.1 Object Paradigm in a Nutshell

The object-oriented paradigm for program construction builds on the simple premise that software organized along modular, self-contained "objects" is more maintainable, extensible and reusable than the conventional "action-oriented" approach where software is divided around procedural lines. This is a radical shift from the traditional top-down approach. Instead of concentrating on what functions need to be performed, the focus is shifted to the entities on which functions have to be performed. The object-oriented design methodology leads to an architecture that is based on objects every system or subsystem manipulates.

There are several key advantages of shifting the emphasis from actions to objects. First, the basic objects in any application, in general, change much less frequently than the functions the application is required to perform. This is especially true of database programs. Even when the basic data entities do change, the change either introduces new types of objects or is localized to a few object types. Ease of software integration is another argument in favor of the object-oriented approach. It is difficult to combine actions if the data structures used by them are incompatible. In this respect, pre-agreed object modules hold a distinct advantage over pre-agreed function modules. The same argument holds for reusability of software. The users of the system-provided objects do not have to reinvent the wheel every time a new facility is needed.

Object-oriented program construction begins with the identification of objects that the applications will manipulate. Thus the key question the designers need to ask is, "What are the basic entities that the program will need?" rather than "What function will the program perform?" Once the set of objects and their characteristics are identified, *classes* of objects sharing similar characteristics are defined. These classes are typically organized in an *inheritance* hierarchy or lattice to describe interrelationships among them.

A class should be viewed as a self-contained modular unit that specifies what can be stored with objects of this class, and provides *operations* or *methods* that can be performed on these objects. Thus the notion of class is quite close to that of an *abstract data type*. In fact the object-oriented design of software systems can be thought of as a collection of well-integrated abstract data types providing a core of functionality around which complex systems can be built.

The basic tenets of the object paradigm are summarized below.

1. Make the system modules correspond to the data structures to be used. These modules should be self-contained as far as possible.

2. Implement each module as an abstract data type. Every module should correspond to a class of data objects. The interface to the object implemented by a module should be explicit and concise. In addition, the user of the module should be given access to minimal information about the object, preferably through some layers of abstraction (information hiding).
3. Organize the modules in a hierarchy/lattice reflecting the commonality among the object classes they correspond to.
4. Provide a set of generic operations which can act on objects of various classes.

One look at the above steps for object-oriented software construction would confirm that they can be applied just as effectively to a database management system. The only difference is that in the latter case the objects in question are *persistent* and outlast the invocation of any individual program. This, however, lends further credibility to the basic premise of the object paradigm that the data structures change less frequently than the programs that manipulate the data.

On the surface the object paradigm for software construction in general, and database management in particular, appears to be a marriage of software engineering principles that have been known for a long time. Most of the concepts mentioned above have been discussed at great length in the research literature in the past quarter century. All these concepts, in one form or another, were present in early efforts such as Simula 67 (1966), Smalltalk (first version in 1972), CLU (mid-1970s), Mesa (1979), Modula-2 (1982), Ada (1983), and several others.

Is object orientation an old wine in a new bottle? Even though some in the community view it that way, we differ with this characterization. Even if the parts of the solution existed—and that should be no surprise as we are addressing the same problems in large software design as those that existed 25 years ago—it is this coming together of these disparate concepts that have given some a glimmer of hope. What makes the object paradigm powerful and exciting is the symbiotic fusion of many basic solution strategies that promises a concerted attack on the large software development problem. From this point of view, many vintage wines, aged to perfection and harmoniously blended, in a new bottle is perhaps a better description of the object paradigm.

1.1.2 Object-Oriented Databases

The last decade has witnessed the emergence of practical, general-purpose, relational database management systems (DBMSs) and associated fourth-generation database languages as state-of-the-art technology. Commercial DBMSs based on relational and pseudo-relational concepts are now widely utilized in a variety of application environments, on computers ranging from large-scale mainframes to small personal computers.

Recent research and development efforts have resulted in yet another generation of database technology: the so-called semantic and object-oriented database (OODB) systems. These advanced systems, now beginning to appear as commercial products, provide further database management capabilities and address some of the limitations of relational and other record-oriented DBMSs (e.g., those based on hierarchical and network data models).

This book is about this new generation of database technology that incorporates the object paradigm in a database management system. Several issues are important in the study of OODBs. What object-oriented concepts can be applied to databases? How are these concepts incorporated in an OODB? How are OODBs constructed and used? Can database management capabilities be merged into an object-oriented programming language? The chapters that follow will provide answers to these and many other questions.

1.2 TOPICS IN THE STUDY OF OODBs

Our brief introduction to the object-oriented concepts pointed out that applying these concepts in a database setting is what this book is all about. In this section we discuss the key object-oriented concepts as they appear in the database context. Various issues pertinent in the study of OODBs are outlined and an attempt is made to show how the various chapters of the book address these issues. This section is as much an introduction to the book as it is to the field. We hope that by highlighting the salient points of the various chapters the reader will be pointed in the proper direction.

1.2.1 The Basics

Part I of this book is a collection of chapters which talk about the basic OODB concepts in general. Starting with an evolutionary history of OODB concepts, the steps involved in logical design of an object-oriented schema are covered. This part then presents the issues pertinent in schema evolution and benchmarking. Finally, guidelines for harnessing this new technology in an environment dominated by conventional data structuring paradigms are discussed.

The evolution of OODB concepts. The concepts that OODBs bring together have been under development in diverse fields such as programming languages, compiler theory, database theory and artificial intelligence. How these concepts evolved from early semantic networks and relational models into object-oriented programming languages, and then into OODBs, makes a fascinating story.

The book begins with an overview of the field from an evolutionary perspective. In the chapter "A Perspective on Object-Oriented and Semantic Database Models and Systems," Dennis McLeod, a pioneer in the field of semantic data models, traces the evolutionary history of OODBs. The chapter is of a general nature. It requires no prior knowledge of OODBs and is suitable for a wide audience.

Object-oriented schema design. A key step in the design of OODBs is the derivation of classes that describe the data objects required by the applications and the specification of interrelationships among them. This activity is referred to as the logical design of an object-oriented database schema. Most applications require a wide variety of changes before converging to an acceptable schema. In general, the users of OODBs arrive at the desired schema for objects through trial and error.

Schema design has not been thoroughly addressed in the database literature to date even though a considerable body of research exists in the areas of AI knowledge representation, dependency theory, AI theorem proving, and graph algorithms to solve some of the most fundamental problems in schema design. A unified framework for logical design of an OODB schema that synthesizes existing research results from these areas is essential for deriving schemata that are efficient, consistent and non-redundant.

The third chapter by Hyoung-Joo Kim, "Algorithmic and Computational Aspects of Object-Oriented Schema Design," looks at the issues and problems involved in logical schema design in an OODB. The basic steps involved in schema design are discussed. Three fundamental problems in the process are identified and their computational complexity studied. The latter portion of this chapter is quite detailed and readers may wish to skip it in the first reading without any loss in continuity.

Schema evolution. Large applications such as weather simulation software, embedded computer systems for real time flight guidance, office information systems, and payroll and accounting packages are never written in their final form the first time around. An important issue in this respect is the ability to make a wide variety of changes to the database schema dynamically. This process is called *schema evolution*. For practical applications of object-oriented databases, which evolve over time, some form of schema evolution is indispensable. Unfortunately, changes to the schema, which are common in many application environments, are in general inadequately supported by existing database systems.

This fundamental problem concerning changes to the conceptual structure (meta-data/knowledge) of a database is addressed in Chapter 4, "Conceptual Database Evolution through Learning." In this chapter, Li and McLeod explore methods for automatic object flavor evolution using machine learning techniques.

Performance and benchmarking. For any new technology that claims to solve some of the most nagging problems of an existing one, good performance is a crucial condition for survival. If modeling power and methodological elegance are the jewels in the OODB crown, performance, at least so far, has been its Achilles' heel. Improving performance is the central issue facing OODBs today. Another important need is a method for unbiased evaluation in an environment for which OODBs are targeted. Since most object-oriented databases are aimed to meet the needs of engineering applications such as CAD and CASE, an application-oriented approach is more suitable. The chapter entitled "The HyperModel Benchmark" describes such an approach to database evaluation. In this chapter, a generic benchmark that can be used to measure the performance of any OODB is described at a conceptual level.

Transition to the object paradigm. In the past few decades very large systems have been built using the relational and other conventional database management systems. These systems represent an astronomical investment in terms of effort, time, and money. From this point of view a transition to object-oriented development and promoting this new paradigm in the relational world is an important issue. In fact, for a very large system, it is not clear what exactly is meant by the word "transition." Should the new applications embrace the new paradigm and the old ones be left untouched? Is it possible for the two to co-exist in the first place? How does one go about convincing the upper-level management that a change to object-oriented technology is warranted?