The Object Database Standard: ODMG 2.0

"BOOK Extract"

Edited by

R.G.G. Cattell
Douglas K. Barry

Contributors

Dirk Bartels
Mark Berler
Jeff Eastman
Sophie Gamerman
David Jordan
Adam Springer
Henry Strickland
Drew Wade

Morgan Kaufmann Publishers
San Francisco, California
415 / 392 - 2665
1997 -- 288 pages -- paper
$36.95
ISBN 1-55860-463-4
http://www.mkp.com/books_catalog/1-55860-463-4.asp

Table of Contents

Preface

This book defines the ODMG standard for object database management systems. It should be useful to engineers, managers, and students interested in object database systems. Although product documentation from ODMG member companies covers similar information, this book represents the definitive reference for writing code that will work with multiple products. The book has also proven useful in courses on object databases, and as an overview of object database programming.

Release 2.0 differs from the previous Release 1.2 in a number of ways. With the wide acceptance of Java, we added a Java persistence standard in addition to the existing Smalltalk and C++ ones. We made the ODMG object model much more comprehensive, added a meta-object interface, defined an object interchange format, and worked to make the programming language bindings consistent with the common model. We made changes throughout the specification based on several years experience implementing the standard in object database products.

With Release 2.0, the ODMG standard has reached a new level of maturity. Products now available comply reasonably well with the C++ binding, Smalltalk binding, OQL, and even the new Java binding. As with Release 1.2, we expect future work to be backward compatible with Release 2.0. Although we expect a few changes to come, for
example to the Java binding, the standard should now be reasonable stable. For any changes since publication of this book, please check the contact information at the end of Chapter 1, particularly the web site www.odmg.org.

ODMG created a working group for each chapter of this book. The authors listed on the cover of this book are the elected chairs and editors for those working groups. Work on standards is not always recognized as important to companies whose livelihood depends on next quarter's revenue, so these authors are to be commended on their personal dedication and cooperation in improving the usability and consistency of their technology. In addition, other people have made important contributions to the ODMG working groups and to previous releases of this standard. These people are acknowledged in Chapter 1.

Rick Cattell, March 1997

Chapter 1

Overview

1.1 Background

This document describes the continuing work on standards for object database management systems (ODBMSs) undertaken by the members of the Object Database Management Group (ODMG). This specification represents an enhancement to ODMG-93, Release 1.2.

We have worked outside of traditional standards bodies for our efforts in order to make quick progress. Standards groups are well suited to incremental changes to a proposal once a good starting point has been established, but it is difficult to perform substantial creative work in such organizations due to their lack of continuity, large membership, and infrequent meetings. It should be noted that relational database standards started with a database model and language implemented by the largest company involved
(IBM); for our work, we have picked and combined the best features of implementations we had available to us.

1.1.1 Importance of a Standard

Before ODMG, the lack of a standard for object databases was a major limitation to their more widespread use. The success of relational database systems did not result simply from a higher level of data independence and a simpler data model than previous systems. Much of their success came from the standardization that they offer. The acceptance of the SQL standard allows a high degree of portability and interoperability between systems, simplifies learning new relational DBMSs, and represents a
wide endorsement of the relational approach.

All of these factors are important for object DBMSs, as well. The scope of object DBMSs is more far-reaching than that of relational DBMSs, integrating the programming language and database system, and encompassing all of an application's operations and data. A standard is critical to making such applications practical.

The intense ODMG effort has given the object database industry a "jump start" toward standards that would otherwise have taken many years. ODMG enables many vendors to support and endorse a common object database interface to which customers write their applications.

1.1.2 Goals

Our primary goal is to put forward a set of standards allowing an ODBMS customer to write portable applications, i.e., applications that could run on more than one ODBMS product. The data schema, programming language binding, and data manipulation and query languages must be portable. Eventually, we hope our standards proposal will be helpful in allowing interoperability between the ODBMS products, as well, e.g., for heterogeneous distributed databases communicating through the OMG Object Request Broker.

We are striving to bring programming languages and database systems to a new level of integration, moving the industry forward as a whole through the practical impetus of real products that conform to a more comprehensive standard than is possible with relational systems. We have gone further than the least common denominator of the first relational standards, and we want to provide portability for the entire application, not just the small portion of the semantics encoded in embedded SQL statements.

The ODMG member companies, representing almost the entire ODBMS industry, are supporting this standard. Thus, our proposal has become a de facto standard for this industry. We have also used our specification in our work with standards groups such as the OMG and the ANSI X3H2 (SQL) committee.

We do not wish to produce identical ODBMS products. Our goal is source code port-ability; there is a lot of room for future innovation in a number of areas. There will be differences between products in performance, languages supported, functionality unique to particular market segments (e.g., version and configuration management),

accompanying programming environments, application construction tools, small versus large scale, multithreading, networking, platform availability, depth of functionality, suites of predefined type libraries, GUI builders, design tools, and so on.

Wherever possible, we have used existing work as the basis for our proposals, from standards groups and from the literature. But, primarily, our work is derived by combining the strongest features of the ODBMS products currently available. These products offer demonstrated implementations of our standards components that have been tried in the field.

1.1.3 Definition

It is important to define the scope of our efforts, since ODBMSs provide an architecture that is significantly different than other DBMSs — they are a revolutionary rather than an evolutionary development. Rather than providing only a high-level language such as SQL for data manipulation, an ODBMS transparently integrates database capability with the application programming language. This transparency makes it unnecessary to learn a separate DML, obviates the need to explicitly copy and translate data between database and programming language representations, and supports substantial performance advantages through data caching in applications. The ODBMS includes the query language capability of relational systems as well, and the query language model is more powerful; e.g., it incorporates lists, arrays, and results of any type.

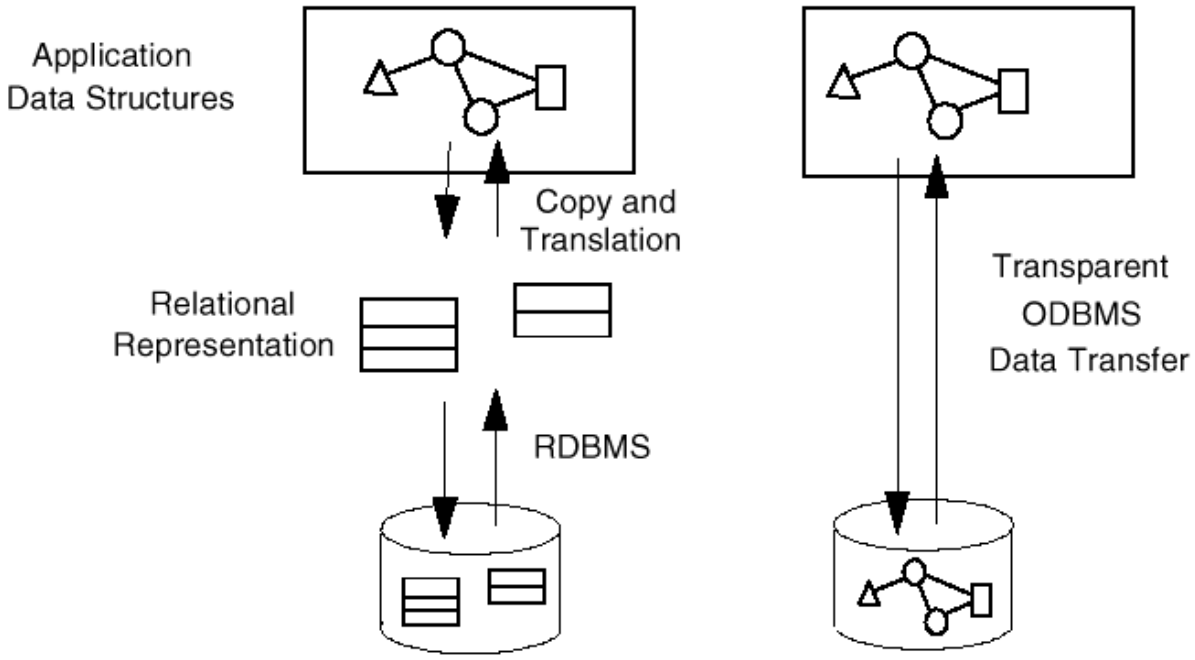


Figure 1-1. Comparison of DBMS Architectures

In summary, we define an *ODBMS* to be a DBMS that integrates database capabilities with object-oriented programming language capabilities. An ODBMS makes database objects appear as programming language objects, in one or more existing programming languages. The ODBMS extends the language with transparently persistent data, concurrency control, data recovery, associative queries, and other database capabilities. For more extensive definition and discussion of ODBMSs, the reader is referred to textbooks in this area (e.g., Cattell, *Object Data Management*).

1.2 Architecture

In order to understand the chapters of this book, it is necessary to understand the overall architecture of ODBMSs.

1.2.1 Major Components

The major components of ODMG-2.0 are described in subsequent chapters of the book:

Object Model. The common data model to be supported by ODBMSs is described in Chapter 2. We have used the OMG Object Model as the basis for our model. The OMG core model was designed to be a common denominator for object request brokers, object database systems, object programming languages, and other applications. In keeping with the OMG Architecture, we have designed an ODBMS *profile* for their model, adding components (e.g., relationships) to the OMG core object model to support our needs. Release 2.0 introduces a meta model in this chapter.

Object Specification Languages. The specification languages for ODBMSs are described in Chapter 3. One is the object definition language, or ODL, to distinguish it from traditional database data definition languages, or DDLs. We use the OMG interface definition language (IDL) as the basis for ODL syntax. Release 2.0 adds another language, the object interchange format, or OIF, which can be used to exchange objects between databases, provide database documentation, or drive database test suites.

Object Query Language. We define a declarative (nonprocedural) language for querying and updating database objects. This object query language, or OQL, is described in Chapter 4. We have used the relational standard SQL as the basis for OQL, where possible, though OQL supports more powerful capabilities.

C++ Language Binding. Chapter 5 presents the standard binding of ODBMSs to C++; it explains how to write portable C++ code that manipulates persistent objects. This is called the C++ OML, or object manipulation language. The C++ binding also includes a version of the ODL that uses C++ syntax, a mechanism to invoke OQL, and procedures for operations on data-bases and transactions.

Smalltalk Language Binding. Chapter 6 presents the standard binding of ODBMSs to Smalltalk; it defines the binding in terms of the mapping between ODL and Smalltalk, which is based on the OMG Smalltalk binding for IDL. The Smalltalk binding also includes a mechanism to invoke OQL, and procedures for operations on databases and transactions.

Java Language Binding. Chapter 7 defines the binding between the ODMG Object Model (ODL and OML) and the Java programming language as defined by Version 1.1 of the Java™ Language Specification. The Java language binding also includes a mechanism to invoke OQL, and procedures for operations on databases and transactions.

It is possible to read and write the same database from C++, Smalltalk, and Java, as long as the programmer stays within the common subset of supported data types. More chapters may be added at a future date for other language bindings. Note that unlike SQL in relational systems, ODBMS data manipulation languages are tailored to specific application programming languages, in order to provide a single, integrated environment for programming and data manipulation. We don't believe exclusively in a universal DML syntax. We go further than relational systems, as we support a unified object model for sharing data across programming languages, as well as a common query language.

1.2.2 Additional Components

In addition to the object database standards, ODMG has produced some ancillary results aimed at forwarding the ODBMS industry. These are included as appendices:

OMG Object Model Profile. Appendix A describes the differences between our object model and the OMG object model, so that Chapter 2 can stand alone. As just mentioned, we have defined the components in an ODBMS profile for OMG's model. This appendix delineates these components.

OMG ORB Binding. Appendix B describes how ODBMS objects could participate as OMG objects, through an adaptor to an object request broker (ORB) that routes object invocations through object identifiers provided by an ODBMS. We also outline how ODBMSs can make use of the OMG ORB.

1.2.3 ODBMS Architecture Perspective

A better understanding of the architecture of an ODBMS will help put the components we have discussed into perspective. Figure 1-2 illustrates the use of the typical ODBMS product that we are trying to standardize.
The programmer writes declarations for the application schema (both data and interfaces) plus a source program for the application implementation. The source program is written in a programming language (PL) such as C++, using a class library that provides full database OML including transactions and object query. The schema declarations may be written in an extension of the programming language syntax, labeled PL ODL in the figure, or in a programming language-independent ODL. The latter could be used as a higher-level design language, or to allow schema definition independent of programming language.

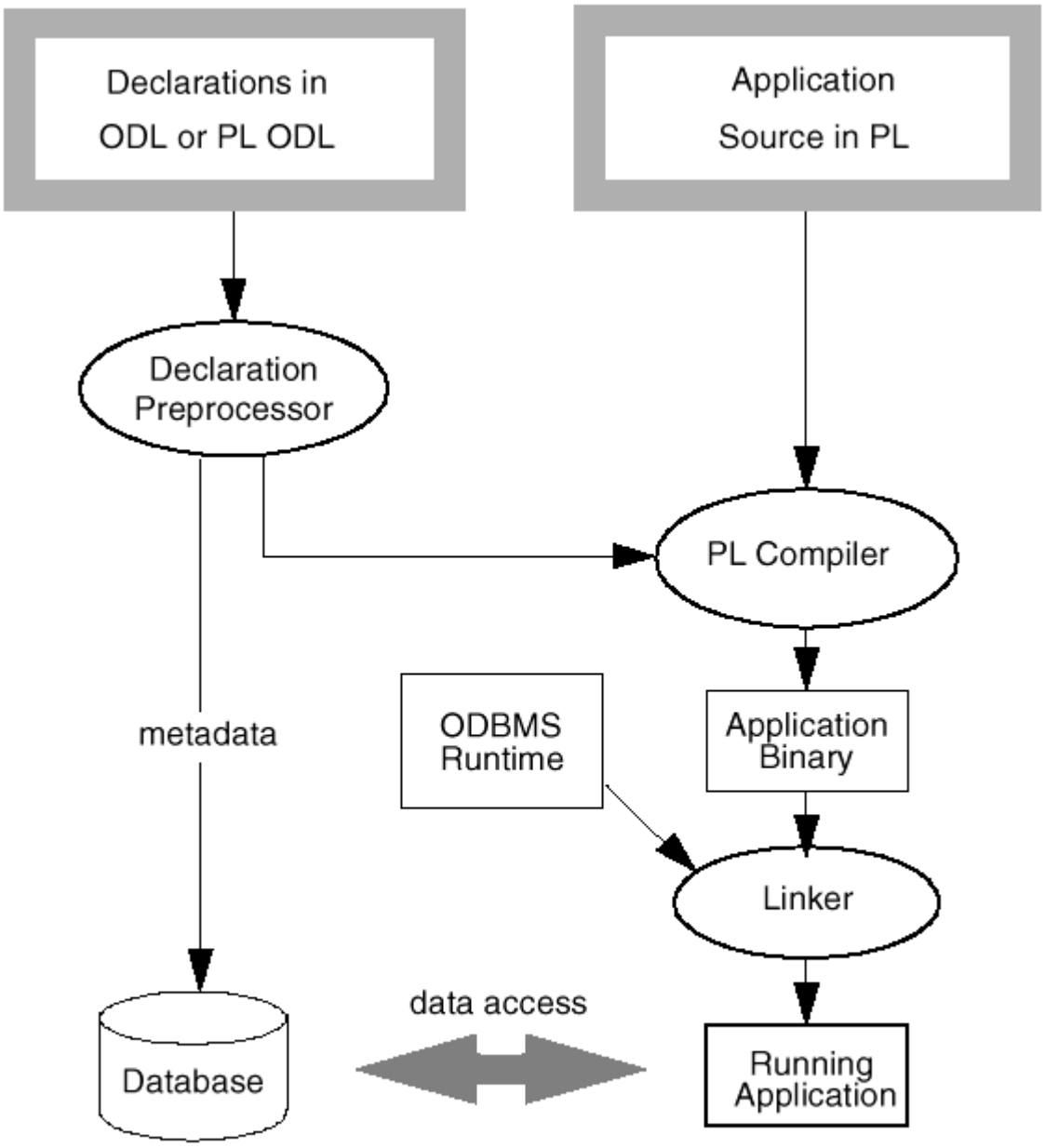


Figure 1-2. Using an ODBMS

The declarations and source program are then compiled and linked with the ODBMS to produce the running application. The application accesses a new or existing database, whose types must conform to the declarations. Databases may be shared with other applications on a network; the ODBMS provides a shared service for transaction and lock management, allowing data to be cached in the application.

1.3 Status

This document describes Release 2.0 of the ODMG standard. The ODMG voting member companies and many of the reviewer member companies are committed to support this standard in their products by the end of 1998.

1.3.1 Participants

As of March 1997, the participants in the ODMG are

- Rick Cattell (ODMG chair, Java workgroup chair, Release 1.0 editor),

- JavaSoft
- Jeff Eastman (ODMG vice-chair, object model workgroup chair,
- Smalltalk editor), Windward Solutions
- Douglas Barry (ODMG executive director, Release 1.1, 1.2 & 2.0 editor),
- Barry & Associates
- Mark Berler (object model and object specification languages editor),
- American Management Systems
- David Jordan (C++ editor), Lucent Technologies
- Francois Bancilhon, Sophie Gamerman (OQL editor), voting member,
- O 2 Technology
- Dirk Bartels (C++ workgroup chair), Olaf Schadow, voting member,
- POET Software
- William Kelly, voting member, UniSQL
- Paul Pazandak, voting member, IBEX
- Ken Sinclair, voting member, Object Design
- Adam Springer (Smalltalk workgroup chair), voting member,
- GemStone Systems
- Henry Strickland (Java editor), Craig Russell, voting member,
- Versant Object Technology
- Drew Wade (OQL workgroup chair), Jacob Butcher, Dann Treachler, voting member, Objectivity
- Michael Card, reviewer member, Lockheed Martin
- Edouard Duvillier, reviewer member, MATISSE Software
- Jean-Claude Franchitti, reviewer member, VMARK Software
- Marc Gille, reviewer member, MICRAM Object Technology
- William Herndon, reviewer member, MITRE
- Mamdouh Ibrahim, reviewer member, Electronic Data Systems
- Richard Jensen, reviewer member, Persistence Software
- Yutaka Kimura, reviewer member, NEC
- Shaun Marsh, reviewer member, Fujitsu Open Systems Solutions
- Richard Patterson, reviewer member, Microsoft
- Shirley Schneider, reviewer member, ONTOS
- Jamie Shiers, reviewer member, CERN
- John Shiner, reviewer member, Andersen Consulting
- Jacob Stein, reviewer member, Sybase
- Fernando Velez, reviewer member, Unidata
- Satoshi Wakayama, reviewer member, Hitachi

It is to the personal credit of all participants that the ODMG standard has been produced and revised expeditiously. All of the contributors put substantial time and personal investment into the meetings and this document. They showed remarkable dedication to our goals; no one attempted to twist the process to his or her company's advantage. The reviewers were also very helpful, always ready to contribute.

In addition to the regular ODMG participants above, we received valuable feedback from others in academia and industry. We would like to thank our academic reviewers, in particular, Eliot Moss for his contribution to the object model chapter. We would also like to thank Joshua Duhl for his exhaustive review of Release 1.2 as part of the process to create certification test suites.

1.3.2 History

Some of the history and methodology of ODMG may be helpful in understanding our work and the philosophy behind it. We learned a lot about how to make quick progress in standards in a new industry while avoiding “design by committee.”

ODMG was conceived at the invitation of Rick Cattell in the summer of 1991, in an impromptu breakfast with ODBMS vendors frustrated at the lack of progress toward ODBMS standards. Our first meeting was at SunSoft in the fall of 1991.

The group adopted rules that have been instrumental to our quick progress. We wanted to remain small and focused in our work, yet be open to all parties who are interested in our work. The structure evolved over time. Presently, we have established work-groups, one for each chapter of the specification. Each workgroup is intended to remain small, allowing for good discussion. The specifications adopted in each work-group, however, must go before the ODMG Board for final approval. The Board usually holds open meetings for representatives from all members to attend and comment on our work.

The people who come to our meetings from our member companies are called Technical Representatives. They are required to have a technical background in our industry. We also have established rules requiring the same Technical Representatives come repeatedly to our meetings to maintain continuity of our work. Technical Representatives from voting member companies often contribute 25 percent of their time to the ODMG. Some of the Technical Representatives from our reviewer members contribute an equal amount of time while others are more in the range of ten percent of their time.

Voting membership is open to organizations that have developed and commercially sell a currently shipping ODBMS as defined on page 3. Reviewer members are individuals or organizations having a direct and material interest in the work of the ODMG. Certification members, our newest group of members, are individuals or organizations having a direct and material interest in the certification work of the ODMG.

1.3.3 Accomplishments

Since the publication of Release 1.0, a number of activities have occurred.

1. Incorporation of the ODMG and the establishment of an office.
2. Affiliation with the Object Management Group (OMG), OMG adoption (February 1994) of a Persistence Service endorsing ODMG-93 as a standard interface for storing persistent state, and OMG adoption (May 1995) of a Query Service endorsing the ODMG OQL for querying OMG objects.
3. Establishment of liaisons with ANSI X3H2 (SQL), X3J16 (C++), and X3J20 (Smalltalk), and ongoing work between ODMG and X3H2 for converging OQL and SQL3.
4. Addition of reviewer membership to allow the user community to participate more fully in the efforts of the ODMG.
5. Addition of certification membership to allow the user community to participate in the development of our certification test suites.
6. Publication of articles written by ODMG participants that explain the goals of the ODMG and how they will affect the industry.
7. Collection of feedback on Release 1.0, 1.1 and 1.2, of which much was used in this release.

1.3.4 Next Steps

We now plan to proceed with several actions in parallel to keep things moving quickly.

1. Distribute Release 2.0 through this book.
2. Complete implementation of the specifications in our respective products.
3. Collect feedback and corrections for the next release of our standards specification.
4. Continue to maintain and develop our work.
5. Continue to submit our work to OMG or ANSI groups, as appropriate.

1.3.5 Suggestion and Proposal Process

If you have suggestions for improvements in future versions of our document, we welcome your input. We recommend that change proposals be submitted as follows:

1. State the essence of your proposal.
2. Outline the motivation and any pros/cons for the change.
3. State exactly what edits should be made to the text, referring to page number, section number, and paragraph.
4. Send your proposal to proposal@odmg.org.

1.3.6 Contact Information

For more information about the ODMG and the latest status of its work, send electronic mail to info@odmg.org. You will receive an automated response. If you have questions on ODMG 2.0, send them to question@odmg.org. If you have additional questions, or if you want membership information for the ODMG, please contact ODMG's executive director, Douglas Barry, at dbarry@odmg.org, or contact:

Object Database Management Group
14041 Burnhaven Drive, Suite 105
Burnsville, MN 55337 USA
voice: +1-612-953-7250
fax: +1-612-397-7146
email: info@odmg.org
web: www.odmg.org

1.3.7 Related Standards

There are references in this book to ANSI X3 documents, including SQL specifications (X3H2), Object Information Management (X3H7), the X3/SPARC/DBSSG OODB Task Group Report (contact fong@ecs.ncsl.nist.gov), and the C++ standard (X3J16). ANSI documents can be obtained from:

X3 Secretariat, CBEMA
1250 Eye Street, NW, Suite 200
Washington, DC 20005-3922 USA

There are also references to Object Management Group (OMG) specifications, from the Object Request Broker (ORB) Task Force (also called CORBA), the Object Model Task Force (OMTF), and the Object Services Task Force (OSTF). OMG can be contacted at:

Object Management Group
Framingham Corporate Center
492 Old Connecticut Path
Framingham, MA 01701 USA
voice: +1-508-820-4300
fax: +1-508-820-4303
email: omg@omg.org
web: www.omg.org

Chapter 2

Object Model

2.1 Introduction

This chapter defines the Object Model supported by ODMG-compliant object database management systems. The Object Model is important because it specifies the kinds of semantics that can be defined explicitly to an ODBMS. Among other things, the semantics of the Object Model determine the characteristics of objects, how objects can be related to each other, and how objects can be named and identified.

Chapter 3 defines programming-language independent Object Specification Languages. One such specification language, Object Definition Language (ODL), is used to specify application object models and is presented for all of the constructs explained in this chapter for the Object Model. It is also used in this chapter to define the operations on the various objects of the Object Model. Chapters 5, 6 and 7, respectively, define the C++, Smalltalk and Java programming language bindings for ODL and for manipulating objects. Programming languages have some inherent semantic differences; these are reflected in the ODL bindings. Thus some of the constructs that appear here as part of the Object Model may be modified slightly by the binding to a particular programming language. Modifications are explained in chapters 5, 6, and 7.

The Object Model specifies the constructs that are supported by an ODBMS:

- The basic modeling primitives are the *object* and the *literal*. Each object has a unique identifier. A literal has no identifier.
- Objects and literals can be categorized by their *types*. All elements of a given type have a common range of states (i.e., the same set of properties) and common behavior (i.e., the same set of defined operations). An object is sometimes referred to as an *instance* of its type.
- The state of an object is defined by the values it carries for a set of *properties*.These properties can be *attributes* of the object itself or *relationships* between the object and one or more other objects. Typically the values of an object's properties can change over time.
- The behavior of an object is defined by the set of *operations* that can be executed on or by the object. Operations may have a list of input and output parameters, each with a specified type. Each operation may also return a typed result.
- A *database* stores objects, enabling them to be shared by multiple users and applications. A database is based on a *schema* that is defined in ODL and contains instances of the types defined by its schema.

The ODMG Object Model specifies what is meant by objects, literals, types, operations, properties, attributes, relationships, and so forth. An application developer uses the constructs of the ODMG Object Model to construct the object model for the application. The application's object model specifies particular types, such as Document, Author, Publisher, and Chapter, and the operations and properties of each of these types. The application's object model is the database's (logical) schema.

Analogous to the ODMG Object Model for object databases is the relational model for relational databases, as embodied in SQL. The relational model is the fundamental definition of a relational database management system's functionality. The ODMG Object Model is the fundamental definition of an ODBMS's functionality. It includes significantly richer semantics than does the relational model, by declaring relation-ships and operations explicitly.

Chapter 3

Object Specification Languages

3.1 Introduction

This chapter defines the specification languages used to represent ODMG-compliant object database management systems. These programming-language independent specification languages are used to define the schema, operations and state of an object database. The primary objective of these languages is to facilitate the portability of databases across ODMG compliant implementations. These languages also provide a step toward the interoperability of ODBMSs from multiple vendors.

Two specification languages are discussed in this chapter: Object Definition Language (ODL) and Object Interchange Format (OIF).

3.2 Object Definition Language

The Object Definition Language is a specification language used to define the specifications of object types that conform to the ODMG Object Model. ODL is used to support the portability of database schemas across conforming ODBMSs.

Several principles have guided the development of the ODL, including:

- ODL should support all semantic constructs of the ODMG Object Model.
- ODL should not be a full programming language, but rather a definition language for object specifications.
- ODL should be programming-language independent.
- ODL should be compatible with the OMG's Interface Definition Language (IDL).
- ODL should be extensible, not only for future functionality, but also for physical optimizations.
- ODL should be practical, providing value to application developers, while being supportable by the ODBMS vendors within a relatively short time frame after publication of the specification.

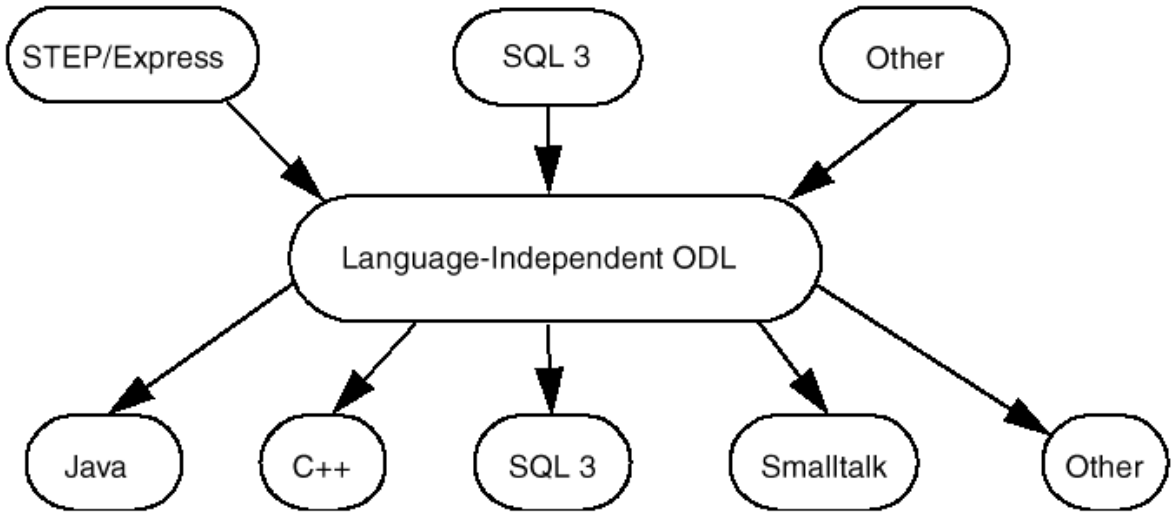
ODL is not intended to be a full programming language. It is a definition language for object specifications. Database management systems (DBMSs) have traditionally provided facilities that support data definition (using a Data Definition Language — DDL) and data manipulation (using a Data Manipulation Language — DML). The DDL allows users to define their data types and interfaces. DML allows programs to create, delete, read, change, etc., instances of those data types. The ODL described in this chapter is a DDL for object types. It defines the characteristics of types, including their properties and operations. ODL defines only the signatures of operations and does not address definition of the methods that implement those operations. The ODMG standard does not provide an OML specification. Chapters 5, 6 and 7 define standard APIs to bind conformant ODBMSs to C++, Smalltalk and Java respectively.

ODL is intended to define object types that can be implemented in a variety of programming languages. Therefore, ODL is not tied to the syntax of a particular programming language. Users can use ODL to define schema semantics in a programming language independent way. A schema specified in ODL can be supported by any ODMG-compliant ODBMS and by mixed-language implementations. This portability is necessary for an application to be able to run with minimal modification on a variety of ODBMSs. Some applications may in fact need simultaneous support from multiple ODBMSs. Others may need to access objects created and stored using different programming languages. ODL provides a degree of insulation for applications against the variations in both programming languages and underlying ODBMS products.

The C++, Smalltalk and Java ODL bindings are designed to fit smoothly into the declarative syntax of their host programming language. Due to the differences inherent in the object models native to these programming languages, it is not always possible to achieve consistent semantics across the programming-language specific versions of ODL. Our goal has been to minimize these inconsistencies, and we have noted, in Chapters 5, 6 and 7, the restrictions applicable to each particular language binding.

The syntax of ODL extends IDL — the Interface Definition Language developed by the OMG as part of the Common Object Request Broker Architecture (CORBA). IDL was itself influenced by C++, giving ODL a C++ flavor. Appendix B, “ODBMS in the OMG ORB Environment,” describes the relationship between ODL and IDL. ODL
adds to IDL the constructs required to specify the complete semantics of the ODMG Object Model.

ODL also provides a context for integrating schemas from multiple sources and applications. These source schemas may have been defined with any number of object models and data definition languages; ODL is a sort of lingua franca for integration. For example, various standards organizations like STEP/PDES (EXPRESS), ANSI X3H2 (SQL), ANSI X3H7 (Object Information Management), CFI (CAD Framework Initiative), and others have developed a variety of object models and, in some cases, data definition languages. Any of these models can be translated to an ODL specification (Figure 3-1). This common basis then allows the various models to be integrated with common semantics. An ODL specification can be realized concretely in an object programming language like C++, Smalltalk or Java.



Chapter 4

Object Query Language

4.1 Introduction

In this chapter, we describe an object query language named OQL, which supports the ODMG data model. It is complete and simple. It deals with complex objects without privileging the set construct and the select-from-where clause.

We first describe the design principles of the language in Section 4.2, then we introduce in the next sections the main features of OQL. We explain the input and result of a query in Section 4.3. Section 4.4 deals with object identity. Section 4.5 presents the path expressions. In Section 4.7, we show how OQL can invoke operations and Section 4.8 describes how polymorphism is managed by OQL. Section 4.9 concludes this part of the presentation of the main concepts by exemplifying the property of operators composition.

Finally, a formal and complete definition of the language is given in Section 4.10. For each feature of the language, we give the syntax, its semantics, and an example. Alternate syntax for some features are described in Section 4.11, which completes OQL in order to accept any syntactical form of SQL. The chapter ends with the formal syntax, which is given in Section 4.12.

4.2 Principles

Our design is based on the following principles and assumptions:

- OQL relies on the ODMG object model.
- OQL is very close to SQL 92. Extensions concern object-oriented notions, like complex objects, object identity, path expressions, polymorphism, operation invocation, late binding.
- OQL provides high-level primitives to deal with sets of objects but is not restricted to this collection construct. It also provides primitives to deal with structures, lists, arrays, and treats such constructs with the same efficiency.
- OQL is a functional language where operators can freely be composed, as long as the operands respect the type system. This is a consequence of the fact that the result of any query has a type which belongs to the ODMG type model, and thus can be queried again .
- OQL is not computationally complete. It is a simple-to-use query language which provides easy access to an ODBMS.
- Based on the same type system, OQL can be invoked from within programming languages for which an ODMG binding is defined. Conversely, OQL can invoke operations programmed in these languages.
- OQL does not provide explicit update operators but rather invokes operations defined on objects for that purpose, and thus does not breach the semantics of an ODBMS which, by definition, is managed by the “methods” defined on the objects.
- OQL provides declarative access to objects. Thus OQL queries can be easily optimized by virtue of this declarative nature.
- The formal semantics of OQL can easily be defined.

Chapter 5

C++ Binding

5.1 Introduction

This chapter defines the C++ binding for ODL/OML.

ODL stands for Object Definition Language. It is the declarative portion of C++ ODL/OML. The C++ binding of ODL is expressed as a library which provides classes and functions to implement the concepts defined in the ODMG object model. OML stands for Object Manipulation Language. It is the language used for retrieving objects from the database and modifying them. The C++ OML syntax and semantics are those of standard C++ in the context of the standard class library.

ODL/OML specifies only the logical characteristics of objects and the operations used to manipulate them. It does not discuss the physical storage of objects. It does not address the clustering or memory management issues associated with the stored physical representation of objects or access structures like indices used to accelerate object retrieval. In an ideal world these would be transparent to the programmer. In the real world they are not. An additional set of constructs called *physical pragmas* is defined to give the programmer some direct control over these issues, or at least to enable a programmer to provide "hints" to the storage management subsystem provided as part of the ODBMS runtime. Physical pragmas exist within the ODL and OML. They are added to object type definitions specified in ODL, expressed as OML operations, or shown as optional arguments to operations defined within OML. Because these pragmas are not in any sense a stand-alone language, but rather a set of constructs added to ODL/OML to address implementation issues, they are included within the relevant subsections of this chapter.

The chapter is organized as follows. Section 5.2 discusses the ODL. Section 5.3 discusses the OML. Section 5.4 discusses OQL — the distinguished subset of OML that supports associative retrieval. Associative retrieval is access based on the values of the properties of objects rather than on their IDs or names. Section 5.6 provides an example program.

5.1.1 Language Design Principles

The programming language–specific bindings for ODL/OML are based on one basic principle: The programmer feels that there is one language, not two separate languages with arbitrary boundaries between them. This principle has two corollaries that are evident in the design of the C++ binding defined in the body of this chapter:

1. There is a single unified type system across the programming language and the database; individual instances of these common types can be persistent or transient.
2. The programming language–specific binding for ODL/OML respects the syntax and semantics of the base programming language into which it is being inserted.

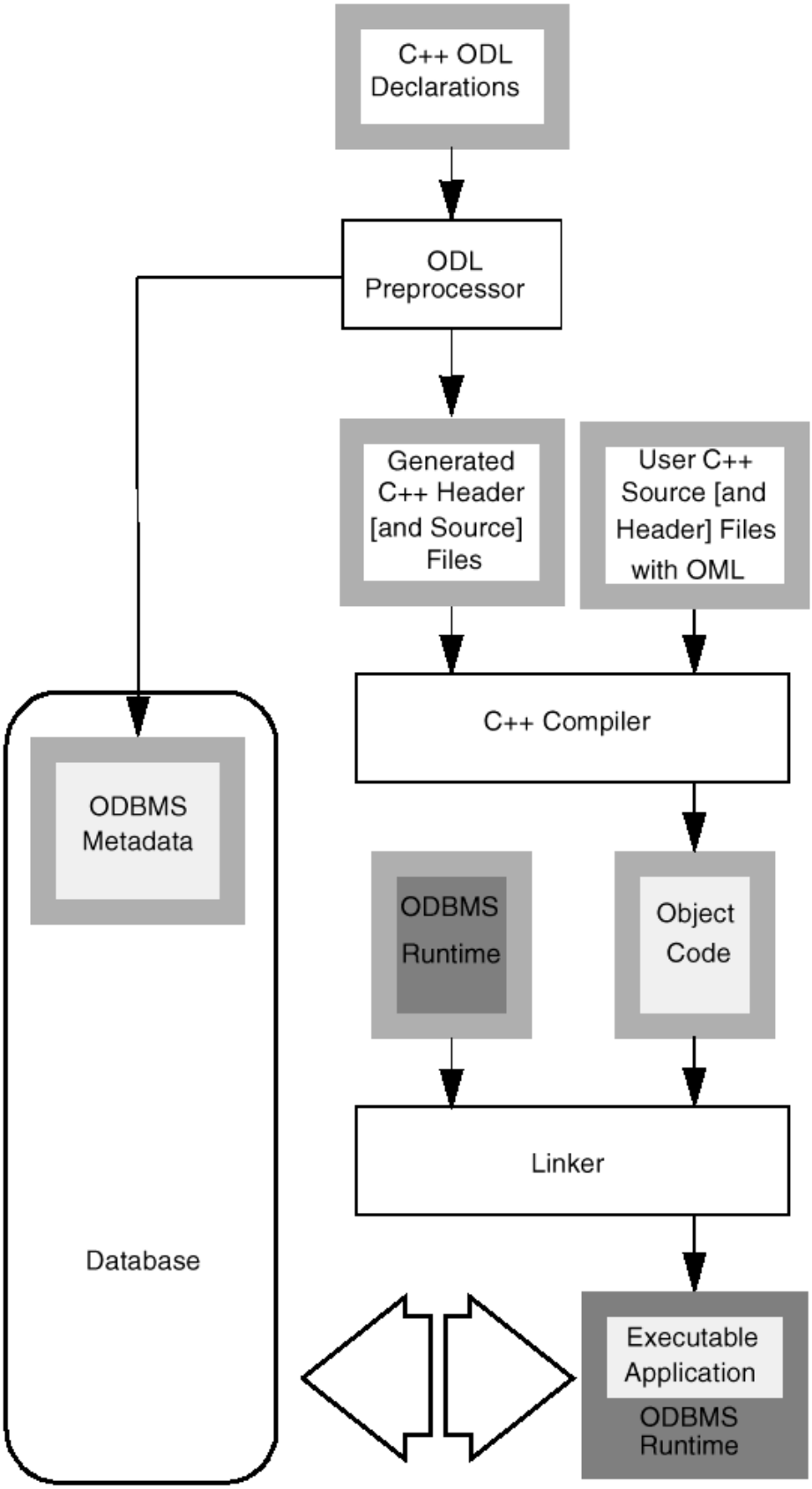


Figure 5-1. Language Hierarchy

Chapter 6

Smalltalk Binding

6.1 Introduction

This chapter defines the Smalltalk binding for the ODMG Object Model, ODL and OQL. While no Smalltalk language standard exists at this time, ODMG member organizations participate in the X3J20 ANSI Smalltalk standards committee. We expect that as standards are agreed upon by that committee and commercial implementations become available that the ODMG Smalltalk binding will evolve to accommodate them. In the interests of consistency and until an official Smalltalk standard exists, we will map many ODL concepts to class descriptions as specified by Smalltalk80.

6.1.1 Language Design Principles

The ODMG Smalltalk binding is based upon two principles: it should bind to Smalltalk in a natural way which is consistent with the principles of the language, and it should support language interoperability consistent with ODL specification and semantics. We believe that organizations who specify their objects in ODL will insist that the Smalltalk binding honor those specifications. These principles have several implications that are evident in the design of the binding described in the body of this chapter.

1. There is a unified type system which is shared by Smalltalk and the ODBMS. This type system is ODL as mapped into Smalltalk by the Smalltalk binding.
2. The binding respects the Smalltalk syntax, meaning the Smalltalk language will not have to be modified to accommodate this binding. ODL concepts will be represented using normal Smalltalk coding conventions.
3. The binding respects the fact that Smalltalk is dynamically typed. Arbitrary Smalltalk objects may be stored persistently, including ODL-specified objects which will obey the ODL typing semantics.
4. The binding respects the dynamic memory management semantics of Smalltalk. Objects will become persistent when they are referenced by other persistent objects in the database and will be removed when they are no longer reachable in this manner.

6.6 Future Directions

Many people believe that keys and extents are an essential ingredient of database query processing. Implicit extents and keys would be preferable to explicit mechanisms involving named Collections, yet there are challenging engineering issues which must be faced to rationalize these capabilities with the notions of transitive persistence and dynamic storage management herein presented.

A uniform set of Database administration operations would facilitate application portability and allow system administration tools to be constructed which could work uniformly across multiple vendors' database products. This binding has only touched upon the need for interface regeneration mechanisms. Such mechanisms would allow programmers with existing applications utilizing language-specific and even database-specific ODL mechanisms to produce the interface definitions which would insulate them from the differences between these mechanisms.

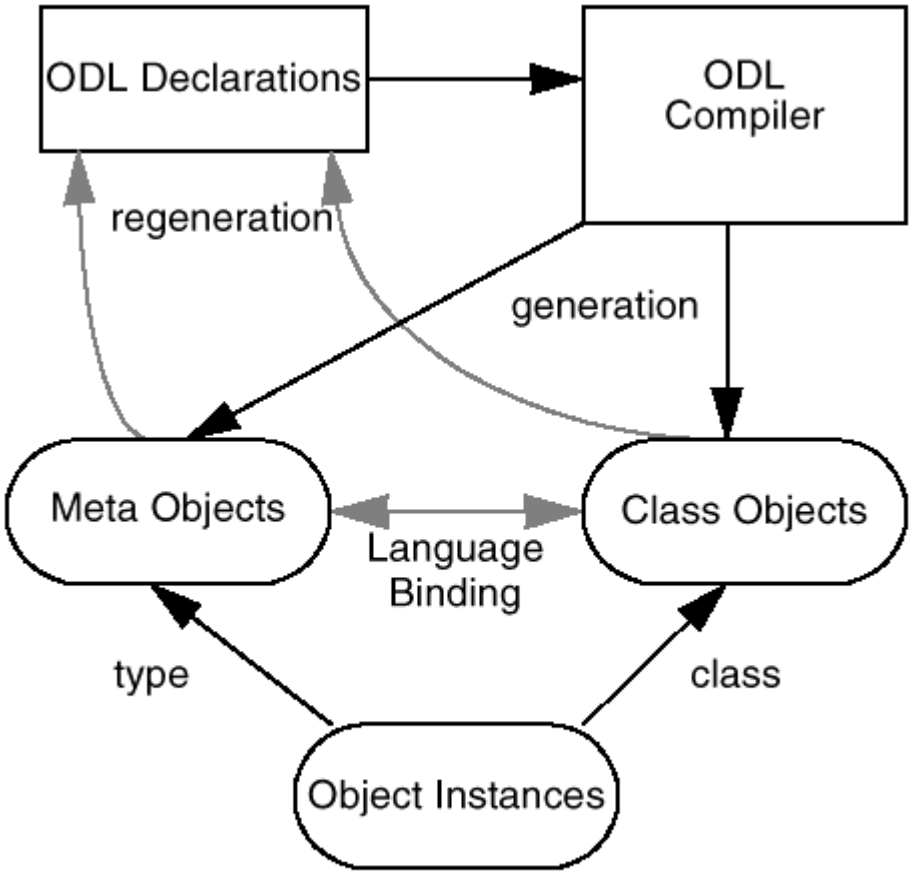


Figure 6-1. Smalltalk Language Binding

Chapter 7

Java Binding

7.1 Introduction

This chapter defines the binding between the ODMG Object Model (ODL and OML) and the Java programming language as defined by Version 1.1 of the Java™ Language Specification.

7.1.1 Language Design Principles

The ODMG Java binding is based on one fundamental principle: the programmer should perceive the binding as a single language for expressing both database and programming operations, not two separate languages with arbitrary boundaries between them. This principle has several corollaries evident throughout the definition of the Java binding in the body of this chapter:

- There is a single unified type system shared by the Java language and the object database; individual instances of these common types can be persistent or transient.
- The binding respects the Java language syntax, meaning that the Java language will not have to be modified to accommodate this binding.
- The binding respects the automatic storage management semantics of Java. Objects will become persistent when they are referenced by other persistent objects in the database and will be removed when they are no longer reach-able in this manner.

Note that the Java binding provides *persistence by reachability,* like the ODMG Smalltalk binding (this has also been called *transitive persistence*). On database commit, all objects reachable from database root objects are stored in the database.

7.1.2 Language Binding

The Java binding provides two ways to declare persistence-capable Java classes:

- Existing Java classes can be made persistence-capable.
- Java class declarations (as well as a database schema) may automatically be generated by a preprocessor for ODMG ODL.

One possible ODMG implementation that supports these capabilities would be a *post-processor* that takes as input the Java .class file (bytecodes) produced by the Java compiler and produces new modified bytecodes that support persistence. Another implementation would be a *preprocessor* that modifies Java source before it goes to the Java compiler. Another implementation would be a modified Java interpreter.

We want a binding that allows all of these possible implementations. Because Java does not have all hooks we might desire, and the Java binding must use standard Java syntax, it is necessary to distinguish special classes understood by the database system. These classes are called *persistence-capable classes.* They can have both persistent and transient instances. Only instances of these classes can be made persistent. The current version of the standard does not define how a Java class becomes a persistence-capable class.

Appendix A

Comparison with OMG Object Model

A.1 Introduction

This appendix compares the ODMG Object Model outlined in Chapter 2 of this specification with the OMG Object Model as outlined in Chapter 4 of the *OMG Architecture Guide*.

The bottom line is that the ODMG Object Model (ODMG/OM) is a superset of the OMG Object Model (OMG/OM). The subsections of this appendix discuss the purpose of the two models and how the ODMG/OM fits into the component/profile structure defined by the OMG/OM, and review the capability between the two models in the major areas defined by the OMG/ OM: types, instances, objects, and operations.

A.2 Purpose

The OMG/OM states that its primary objective is to support application portability. Three levels of portability are called out: (1) design portability, (2) source code portability, and (3) object code portability. The OMG/OM focused on design portability. The ODMG/OM goes a step further — to source code portability. The OMG/OM distinguishes two other dimensions of portability: portability across technology domains (e.g., a common object model across GUI, PL, and DBMS domains), and portability across products from different vendors within a technology domain (e.g., across ODBMS products from different vendors). The ODMG/OM focuses on portability within the technology domain of object database management systems. The ODMG standards suite is designed to allow application builders to write to a single ODBMS application programming interface (API), in the assurance that this API will be sup-ported by a wide range of ODBMS vendors. The ODMG/OM defines the semantics of the object types that make up this API. Subsequent chapters within the ODMG standard define the syntactic forms through which this model is bound to specific programming languages.

To offer real portability, a standard has to support a level of DBMS functionality rich enough to meet the needs of the applications expected to use the standard. It cannot define such a low-level API that real applications need to use functionality supplied only by vendor-specific extensions to the API. The low-level, least-common-denominator approach taken in the standards for relational data management has meant that real applications need to use functionality supplied only by vendor-specific extensions to the API. Several studies in the late 1980s that analyzed large bodies of applications written against the relational API (SQL) showed that 30–40% of the RDBMS calls in the application are actually “standard SQL”; the other 60–70% use vendor-specific ex-tensions. The result is that the relational standard does not in practice deliver the source-code-level application portability that it promised. The ODMG APIs have been designed to provide a much higher level of functionality and therefore a much higher degree of application portability.

A.3 Components and Profiles

The OMG Object Model is broken into a set of *components*, with a distinguished “Core Component” that defines objects and operations. The theory espoused by the OMG is that each “technology domain” (GUI, ODBMS, etc.) will assemble a set of these components into a *profile*. Figure A-1 illustrates this. Two profiles are shown — the Object Request Broker (ORB) profile and the Object DBMS (ODBMS) profile.

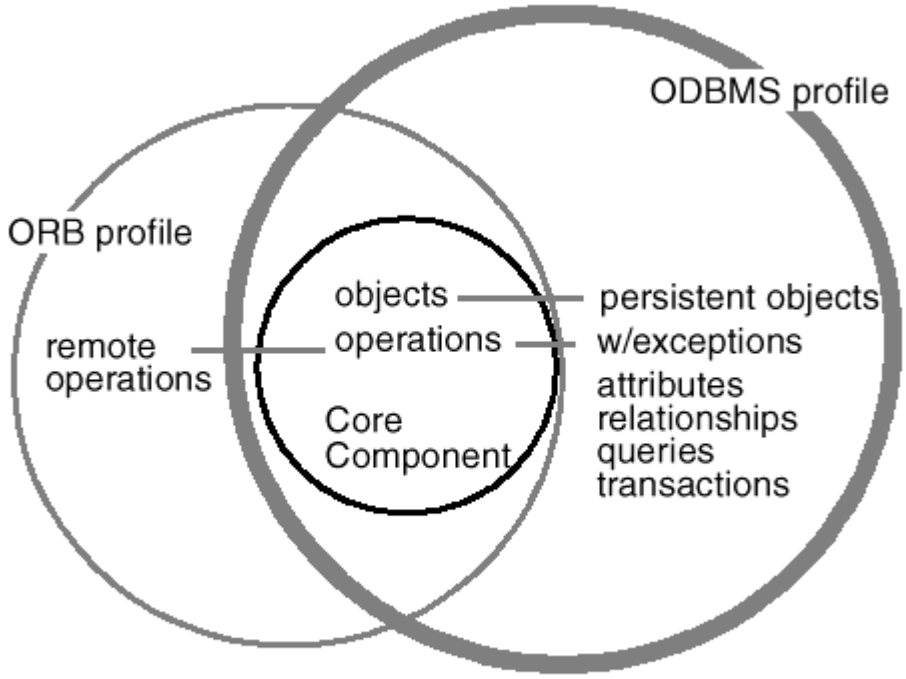


Figure A-1

The ORB profile includes the Core Component plus support for remote operations.
The ODBMS profile includes the Core Component plus support for

- persistent objects
- properties (attributes and relationships)
- queries
- transactions

It also strengthens the core component definition of operations by including exception returns.

To date, the only OMG/OM component that has been defined is the Core Component. The additional functionality included in the ORB profile has not been formally specified as a set of named components. Nor are there OMG component definitions for the functionality expected to be added by the ODBMS profile. One of the reasons for making the comparison between the OMG/OM (i.e., the Core Component) is that the members of ODMG expect to submit definitions of each of the items in the bulleted list above as candidate *components*, and the sum of them as a candidate *profile* for object database management systems. Since the submitting companies collectively represent 80+% of the commercially available ODBMS products on the market, we assume that adoption of an ODBMS profile along the lines of that outlined in Chapter 2 will move through the OMG process relatively quickly.

The OMG/OM is outlined below, with indications how the ODMG/OM agrees.

Types, Instances, Interfaces, and Implementations:

- Objects are instances of types.
- A type defines the behavior and state of its instances.
- Behavior is specified as a set of operations.
- An object can be an immediate instance of only one type.
- The type of an object is determined statically at the time the object is created; objects do not dynamically acquire and lose types.
- Types are organized into a subtype/supertype graph.

- A type may have multiple supertypes.
- Supertypes are explicitly specified; subtype/supertype relationships between types are not deduced from signature compatibility of the types.

Operations:

- Operations have signatures that specify the operation name, arguments, and return values.
- Operations are defined on a single type — the type of their distinguished first argument — rather than on two types.
- Operations may take either literals or objects as their arguments. Semantics of argument passing is pass by reference.
- Operations are invoked.
- Operations may have side effects.
- Operations are implemented by methods in the implementation portion of the type definition.

The OMG/OM does not currently define exception returns on operations; it says that it expects there to be an exception-handling component defined outside of the core model. The ODMG/OM does define exception returns to operations.

A.4 Type Hierarchy

The fact that the ODMG/OM is a superset of the OMG/OM can also be seen by looking at the built-in type hierarchy defined by the two models. Figure A-2 shows the ODMG/OM type hierarchy. The types whose names are shown in italics are those which are also defined in the OMG/OM. As in Chapter 2, indenting is used to show subtype/supertype relationships, e.g., the type Collection is a subtype of the type Object- type.

- *Literal_type*
    - Atomic_literal
    - Collection_literal
    - *Structured_literal*
- Object_type
    - Atomic_object
    - Collection

Figure A-2

The ODMG/OM is a richer model than the OMG/OM — particularly in its support for properties and in its more detailed development of a subtype hierarchy below the types Object and Literal. The only differences between the two models in the areas common to them are two type names. The type that is called Literal in the ODMG/OM is called Non-Object in the OMG/OM. Although the OMG/OM does not formally introduce a supertype of the types Object and Non-Object, in the body of the document it refers to instances of these two types as the set of all "denotable values" or "Dvals" in the model. In the ODMG/OM a common supertype for Object and Literal is defined. The in-stances of type Object are mutable; they are therefore given OIDs in the ODMG/OM; although the value of the object may change, its OID is invariant. The OID can therefore be used to denote the object. Literals, by contrast, are immutable. Since the in-stances of a literal type are distinguished from one another by their value, this value can be used directly to denote the instance. There is no need to ascribe separate OIDs to literals.

In summary, the ODMG/OM is a clean superset of the OMG/OM.

A.5 The ORB profile

A second question could be raised. One product category has already been approved by the OMG — the ORB. To what extent are the non-core components implicit in that product consistent or inconsistent with their counterpart non-core components in the ODMG/OM? There is some divergence in literals, inheritance semantics, and operations the latter because the ORB restricts in two key ways the semantics already defined in the OMG core object model: object identity and the semantics of arguments passed to operations. Those battles, however, are not ours. They are

between the OMG ORB task force and the OMG Object Model task force. The requirement placed on a prospective ODBMS task force is simply that the set of components included in the ODBMS profile include the Core Component objects and operations. This appendix addresses that question.

A.6 Other Standards Groups

There are several standards organizations in the process of defining object models.

1. Application-specific standards groups that have defined an object model as a basis for their work in defining schemas of common types in their application domain, e.g.,

- CFI (electrical CAD)
- PDES/STEP (mechanical CAD)
- ODA (computer-aided publishing)
- PCTE (CASE)
- OSI/NMF (telephony)
- ANSI X3H6 (CASE)
- ANSI X3H4 (IRDS reference model)

2. Formal standards bodies working on generic object models, e.g.,

- ISO ODP
- ANSI X3H7 (Object Information Systems)
- ANSI X3T5.4 (managed objects)
- ANSI X3T3

It is our current working assumption that the OMG-promulgated interface definitions for ORB and ODBMS will have sufficiently broad support across software vendors and hardware manufacturers that interface definitions put in the public domain through the OMG and supported by commercial vendors will develop the kind of de facto market share that has historically been an important prerequisite to adoption by ANSI and ISO. Should that prove not to be the case, the ODMG will make direct proposals to ANSI and ISO once the member companies of ODMG and their customers have developed a base of experience with the proposed API through use of commercial ODBMS products that support this API.

Appendix B

ODBMS in the OMG ORB Environment

B.1 Introduction

The existing documents of OMG do not yet address the issue of how an ODBMS fits into the OMG environment and, in particular, how it communicates with and cooperates with the ORB. This fundamental architectural issue is critical to the success of users of the OMG environment who also need ODBMSs.

This document is a position statement of the ODMG defining the desired architecture. It explicitly does not discuss the architecture of the internals of an ODBMS implementation but rather leaves that to the implementor of the ODBMS. Instead, it discusses how the ODBMS fits architecturally into the larger OMG environment.

The issues for a successful fit are the following:

- performance — e.g., direct object access
- distribution and heterogeneity — as managed by ODBMS for fine-grained objects
- ODBMS as Object Manager — responsible for multiple objects
- common repository — ability of ORB to use ODBMS as repository
- ODBMS as a user of the ORB — ability of ODBMS to use the services provided by the ORB (including other ODBMSs)

The architecture must support ODBMS implementations and client interfaces to achieve these.

B.2 Roles for the ORB and ODBMS

The ORB and the ODBMS are different. The ODBMS's role in the OMG environment is to support definition, creation, and manipulation with the services of persistence, transactions, recovery, and concurrent sharing for application objects varying from the smallest units (e.g., words in a word processor, cells or formula terms in a spreadsheet) to the largest (e.g., documents, systems). Many applications desire these services to include, within a single vendor product, transparent distribution in a heterogeneous mixture of platforms and other services such as versioning and security.

Note that we define ODBMS according to the services it provides, not according to any particular implementation of those services. Radically different implementations are possible, including not only traditional ODBMS approaches, but also file-based approaches, each offering different levels of services and trade-offs.

The ORB provides a larger-scale set of services across heterogeneous vendors and products; e.g., it allows clients to use multiple ODBMSs. The service it provides is behavior invocation, or method dispatch. In contrast, the ODBMS provides a single-vendor capability and only a specific set of services rather than arbitrary ones; however, those services include more detailed capabilities of high-performance, fine-grained persistence that are used directly within applications to support millions of primitive objects. The ORB, when it needs persistence services, could choose to implement them via use of an ODBMS. The ODBMS services may be invoked via the ORB.

B.3 Issues

Here we describe some of the key issues that this architecture must address. Since the ODBMS supports millions of fine-grained objects used directly by the applications, it must provide high-performance access to those objects. The pertinent characteristic differentiating large- and small-grain objects is access time. If an application is accessing only one or two objects (e.g., open a spreadsheet document), there is little concern for the time to communicate across networks through the ORB. However, if the application is accessing thousands or millions of objects (e.g., formulae and variables in cells in the spreadsheet), system overhead becomes a significant factor as perceived by the user.

In many cases this means access time that is comparable to native in-memory object usage. To provide this, the ODBMS must be able to move objects as necessary in the distributed environment and cache them locally in the address space of the application, if desired, and in efficient format.

Since the ODBMS objects are those used primarily within the application, it is desirable to support an interface that is natural and direct to the user.

Examples of applications and object granularities for which ODBMS services must be available and efficient include spreadsheets; word processors; documents of these; primitive elements within these such as cells, formulae, variables, words, phrases, and formatting specifications; network managers with objects representing machines, users, and sessions; resource allocation schemes; CAD and CAM with objects such as circuits and gates and pins, routing traces, form features, bezier curves, finite element mesh nodes, edges and faces, tool paths, simulation, and analysis support; financial portfolio analysis; and so on. There may be millions of such objects, in complex inter-connected networks of relationships.

The interfaces to those objects must be defined in such a way as to allow ORB access when appropriate (e.g., for cross-database-vendor object relationships) or direct use of the ODBMS (e.g., for objects with no need to publish themselves for public use through the ORB). This should be done with a single interface to allow transparency to the client and to allow the client to choose to vary functionality as desired.

The ODBMS acts as manager of many objects, so the architecture and interfaces must allow such assignment of responsibilities. The ODBMS can provide distribution of objects among multiple and potentially heterogeneous platforms, so the architecture and implementation must allow this functionality to be relegated to the ODBMS.

The ORB and other OMG components (service providers, library facilities, service users, etc.) may need the services of persistence, or management of objects that exist beyond process lifetimes, for various kinds of objects, including type-defining objects and instances of these. It is desirable, architecturally, to consolidate common services in a common shared component. The architecture must allow use of an ODBMS for this purpose in order to take advantage both of the capabilities it provides and integration with other OMG components using the same services.

As mentioned above, different ODBMS implementation approaches must be supported. The architecture and the OMG interfaces must provide a single interface (or set of interfaces) that allows use of a wide variety of such implementations. A single inter-face allows users to choose which implementation to use and when. This should cover not only full ODBMS implementations but other approaches with partial functionality, such as file management approaches.

In addition to direct use of an ODBMS through an interface such as that defined in the ODMG-93 specification, an ODBMS could be decomposed in order to implement a number of semi-independent *services*, such as persistence, transactions, relationships, and versions. The OMG Object Services Task Force is defining such services. This is an area for future work by the object database vendors. In addition to the ORB and users of the ORB accessing ODBMSs, it is also the case that an ODBMS may be a client of the ORB. The ODBMS may want to use the ORB services such as location and naming (for distributed name services) or may use the ORB in order to access other ODBMSs, thus allowing heterogeneous ODBMS access. Current ODBMSs provide object identifiers that work only within one vendor's products, sometimes only within one database. The ORB object references could serve as a common denominator that allows selected object references and invocations in an ODBMS to span database boundaries (via encapsulating ODBMS object identifiers within ORB object references).

B.4 ODBMS as an Object Manager

The ORB acts as a communication mechanism to provide distributed, transparent dispatching of requests to objects among applications and service providers. The ODBMS acts as manager of a collection of objects, most of which are not directly registered as objects to the ORB, some of which can be very small (fine-grained) application objects, and for which high-speed transparent distributed access must be supported.

If every ODBMS object that an application wanted to reference were individually registered with the ORB or if every request to those objects in the ODBMS went through the ORB Basic Object Adaptor, the overhead would be unacceptable. This is equivalent to saying that every test of a bit of data or change of an integer must invoke the overhead of an RPC mechanism. Instead, the application should have the flexibility to choose which objects and which granularities are in fact known to the ORB, when re-quests to those objects go through the ORB, and be able to change this choice from time to time.

To achieve this maximum flexibility, we specify that the ODBMS has the capability to manage objects unknown or known to the ORB, to register subspaces of object identifiers with the ORB (to allow the ORB to handle requests to all of the objects in an ODBMS without the registration of each individual object), and to provide direct ac-cess to the objects it manages. For the objects unknown to the ORB, this direct access is provided via an ORB request to a containing object (e.g., a database), which then makes those objects directly available to the application. This provides consistency with and participation in the ORB environment and still provides the ODBMS with the ability to move objects around the distributed environment, cache them as appropriate, and provide efficient access.

For objects that the ODBMS has registered with the ORB, it may choose either to let requests to them execute the normal ORB mechanism or request from the ORB that any requests to those objects be passed to the ODBMS, perhaps for some period of time. Requests to such objects, whether through the ORB or directly to the ODBMS, must produce the same effect and be compatible with other users employing both mechanisms. In this way the ODBMS can provide consistency with the ORB and still coordinate with direct object requests.

The currently adopted OMG CORBA document provides for normal object access via the Basic Object Adaptor (BOA). For complete generality, flexibility, and interoperability, it executes via an interprocess mechanism (RPC for short) for every dispatch of every method. An extension is the Library Object Adaptor (LOA) that allows direct, considerably faster access to objects. After the first invocation (via the usual ORB mechanism), or through a compile-time optimization, a direct link is established to the object. Later access by the client to the object is then direct until the client notifies the ORB that it has released the object. We offer a new type of Object Adaptor, the Object Database Adaptor (ODA), to provide the ability to register a subspace of object identifiers and to allow access (including direct access) to the objects as if they had been individually registered.

The ODA provides a mechanism to register a subspace of object identifiers with the ORB rather than having to register all objects in the ODBMS. From the client’s point of view, the objects in the registered subspace appear just as any other ORB-accessible objects, with the same interface. The ODA should allow for the use of direct access (as in the LOA) to improve the performance of ORB/ODBMS applications.

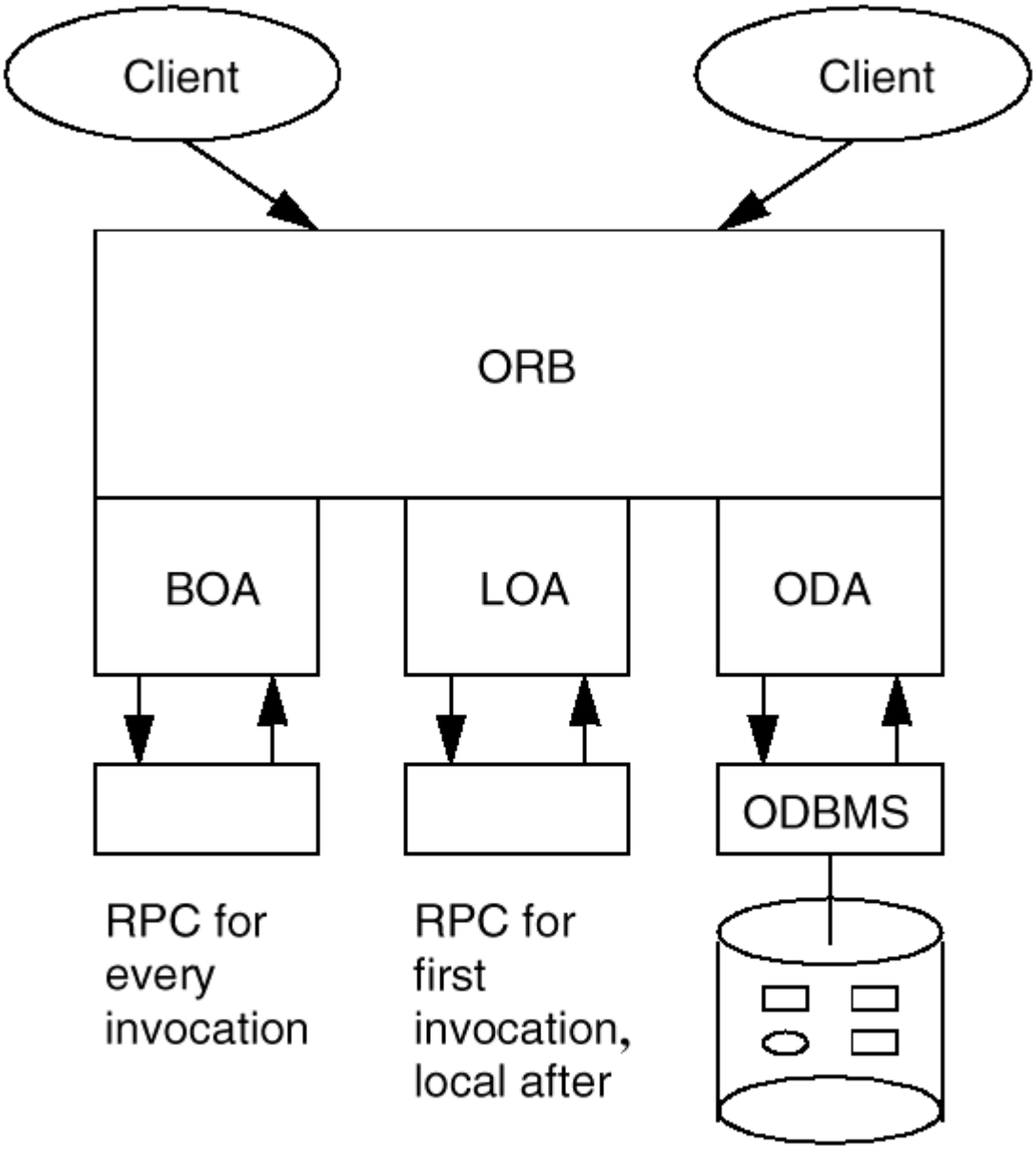


Figure B-1. ODBMS as Object Manager in OMG ORB Architecture

The Object Data Management Group (ODMG) was conceived in the summer of 1991 at a breakfast with object database vendors that was organized by Rick Cattell of Sun Microsystems. In 1998, the ODMG changed its name from the Object Database Management Group to reflect the expansion of its efforts to include specifications for both object database and object-relational mapping products. The Object Model â€¢ About ODMG Object Model â€¢ Database Types ï‚§ Specifications & Implementations â€¢ Object type definition consists of two components, they are Interface & one or more Implementations. U se r Interface Us er visible to the users of the type not visible to the users of the type Implementation Class Us Us er Type Specification er Objects â€¢ â€¢ â€¢. Object Identifiers Object Lifetime Collections Collection Objects Set Objects Bag Objects List Objects Array Objects Dictionary Objects Objects â€¢ Literal Values. ï‚§ ODMG model supports several literal types: â€¢ Atomic â€¢ Collection an Object Manager 253 Index 256 Morgan Kaufmann Object Database Standard ODMG 2.0 (Book Extract) Page 2. 3 Preface This book defines the ODMG standard for object database management systems. It should be useful to engineers, managers, and students interested in object database systems. Although product documentation from ODMG member companies covers similar information, this book represents the definitive reference for writing code that will work with multiple products. The book has also proven useful in courses on object databases, and as an overview of object database programming. Release 2.0