Preface

This document is the revised third edition of the first public document delivered by the Object Management Group (OMG). It puts forth the goals and purpose of the OMG, the structure and procedures of its Technical Committee, an Object Model for design portability, and an outline of both object technology in general and a Reference Model for the distributed application support framework being built by the OMG. It is not intended to be regarded as a complete document: it is a work-in-progress that will change as the OMG develops interface standards and specifications.

This manual is the result of the effort of scores of talented individuals from the OMG member companies and the OMG staff itself. In particular, the following people, listed alphabetically, were instrumental in completing portions of this manual, including the original draft:

William Andreas (HyperDesk Inc.)  Gerhard Muller (Siemens Nixdorf GmbH)
Dr. Gerald Barber (Aldus, Inc.)    Pat O’Brien (Digital)
Ian Fuller (Hewlett-Packard)       Lee Scheffler (Constellation Software)
Andrew Hutt (ICL)                 John Schwartz (Mentor Graphics)
Bill Kent (Hewlett-Packard)        Doug Smith (Hewlett-Packard)
Michael Kilian (Digital)          Alan Snyder (SunSoft)
Matt Koehler (Sun Microsystems)   Dr. Richard Soley (OMG)
Jan te Kiefte (Philips Telecommunications)  Christopher Stone (OMG)
Mike Mathews (Hewlett-Packard)    Craig Thompson (Texas Instruments, Inc.)

This abstract, though complex, document would not have been possible without the splendid spirit of consensus among the OMG corporate members. We would like to thank all those member representatives actively involved in the OMG, and in particular all those who took the time to help with the design and production of this document.

Richard Mark Soley, Ph.D.
Christopher M. Stone
Framingham, Massachusetts USA
First Edition, September 1990
CHAPTER 5  Reference Model

5.1 Purpose of the Reference Model

This chapter describes the Reference Model for the Object Management Architecture (OMA). The Reference Model forms a conceptual road map for assembling technology that satisfies the OMG's Technical Objectives (see Chapter 3). It has three intended audiences:

• The OMG itself. The Reference Model provides a framework for guiding the process of soliciting and evaluating distributed object management technology.

• Potential technology providers. The Reference Model provides an architectural structure for positioning and presenting proposed technology in relation to others.
• Application developers and the software industry in general. The Reference Model articulates the OMG's vision for highly interoperable applications and services using object technology.

Though the Reference Model is intended to influence the high-level architectural and component designs of proposed approaches, it also accommodates a variety of different design solutions. The function of the Reference Model is more to map out areas to be addressed than to impose design constraints, except at the highest architectural level. The Reference Model identifies and characterizes the components, interfaces, and protocols that compose the OMG's Object Management Architecture, but does not itself define them in detail. It puts in place a structure that allows requirements to be defined and solutions to be proposed; it provides a framework that the OMG can populate with detailed interface and protocol specifications. (All OMG specifications are evaluated and approved by the membership of the OMG. The procedures that are used to evaluate and approve specifications are explained in Appendix B.)

The specifications that have been, to date, accepted by the OMG and its membership are described in “Associated Documents,” on page 13.

In addition to providing the framework for writing specifications, the Reference Model:
• Identifies the major separable components of the total Object Management Architecture. The components are: Object Request Broker, Object Services, Common Facilities, and Application Objects.

• Characterizes the functions provided by each component.

• Explains the relationships between the components and with the external operating environment.

• Identifies the protocols and interfaces for accessing the components.

Specifically, the Reference Model addresses:

• How objects make and receive requests and responses.

• The basic operations that must be provided for every object.

• Object interfaces that provide common facilities useful in many applications.

5.2 Object Management Architecture

An application that is "OMA-compliant" consists of a set of interworking classes and instances that interact via the ORB (as defined in the next section). Compliance therefore means conformance to the OMA and the protocol definitions and ORB enables objects to make and receive requests and responses.
• The Object Request Broker is the central component of the Object Management Architecture and is the key to interoperability. It allows invocation of operations on objects distributed around a network without regard to the networking software, operating system, or application that implements those objects. Specifications for an OMG-compliant ORB are contained in *CORBA: Common Object Request Broker Architecture and Specification*.

• Object Services is a collection of services with object interfaces that provide basic functions for realizing and maintaining objects. Specifications for Object Services are contained in *CORBA Services: Common Object Services Specification*.

• Common Facilities is a collection of classes and objects that provide general purpose capabilities useful in many applications. Specifications for Common Facilities are contained in *CORBA Facilities*, which will be published in 1995.

• Application Objects (AO) are specific to particular end user applications.

In general, the Application Objects and Common Facilities have an application orientation while the Object Request Broker and Object Services are concerned with the "system" or infrastructure aspects of distributed object management. Common Facilities may, however,
provide higher-level services, such as transactions and versioning, that use primitives provided by Object Services.

Of the four Reference Model components, three lend themselves to standardization: Object Services, Common Facilities, and Common Object Request Broker. Thus, they will be the focus of OMG standardization efforts. The fourth component, Application Objects, represents functions that are too specialized to standardize at this time.

In general, Object Services, Common Facilities, and Application Objects all communicate using the Object Request Broker. Objects may also use non-object interfaces to external services, but these are outside the scope of the OMA. Although not explicit in the Reference Model, objects may (or may not) communicate with the Object Services via object interfaces. For example, the addition of a new class may be cast as a request to an object that provides this service, but equivalently, it could be performed by editing a class definition script or a C++ include file.

The Application Objects and Common Facilities use and provide functions and services via object interfaces. Objects can issue and process requests. Thus, objects categorized as Application Objects can provide services for other applications or facilities. For example, an application specific service such as printer rendering could be cast as an application object invoked by a common facility such as a print
queue. Equally, objects categorized as Common Facilities may use services provided elsewhere.

Common Facilities exemplifies a key concept that the OMA promotes, class reusability.

It is important to note that applications need only provide or use OMA-compliant interfaces to participate in the Object Management Architecture. They need not themselves be constructed using the object-oriented paradigm. This also applies to the provision of Object Services. For example, existing relational or object-oriented database management systems could be used to provide some or all of the Object Services. Figure 5-2 on page 96 shows how existing applications, external tools, and system support software can be embedded as objects that participate in the Object Management Architecture, using class interface front-ends (otherwise called *adapters* or *wrappers*).

The Reference Model does not impose any restrictions on how applications and common facilities are structured and implemented. Objects of a given application class may deal with the presentation of information, interaction with the user, semantics, functionality, the persistent storage of data, or a combination of the above.
The Object Management Architecture (OMA) assumes that underlying services provided by a platform’s operating system and lower-level basic services, such as network computing facilities, are available and usable by OMA implementations. Specifically, the Object Management Architecture (OMA) does not address user interface support. The interfaces between applications and windowing systems or other display support are the subjects of standardization efforts outside the OMG. Eventually, however, Common Facilities may provide standard user interface classes. In addition, the Reference Model does not deal explicitly with the choice of possible binding mechanisms (e.g., compile time, load time, and runtime).

The following sections describe each part of the OMA in more detail. It should be noted that the partitioning of functionality, particularly between Object Services and Common Facilities, represents a "best guess" and can be expected to evolve as different design approaches are evaluated and the Reference Model is refined.

5.3 Object Request Broker (ORB)

The Object Request Broker (ORB) provides the mechanisms by which objects transparently make and receive requests and responses. In so doing, the ORB provides interoperability between applications
on different machines in heterogeneous, distributed environments and seamlessly connects multiple-object systems.

The specifications for an OMA-compliant ORB are contained in *CORBA: Common Object Request Broker Architecture and Specification*.

The Core Object Model (see Chapter 4) defines an object request and its associated result (response) as the fundamental interaction mechanism. A request names an operation and includes zero or more parameter values, any of which may be object names identifying specific objects. The ORB arranges for the request to be processed. This entails identifying and causing some method to be invoked that performs the operation using the parameters. After the operation terminates, the ORB conveys the results to the requester.

The ORB itself might not maintain all of the information needed to carry out its functions. In the process of conveying a request, the ORB may generate requests of its own to Object Services, or otherwise use them. For example, in order to find the specific method to be executed for a given request, the ORB might use a class dictionary service or might search runtime method libraries.

In order to satisfy the OMG Technical Objectives, the ORB is expected to address all of the following areas, at least to some degree.
• Name services. Object name mapping services map object names in the naming domain of the requester into equivalent names in the domain of the method to be executed, and vice versa. The OMG Object Model does not require object names to be unique or universal. Object location services use the object names in the request to locate the method to perform the requested operation. Object location services may involve simple attribute lookups on objects. In practice, different object systems or domains will have locally preferred object naming schemes.

• Request dispatch. This function determines which method to invoke. The OMG Object Model does not require a request to be delivered to any particular object. As far as the requester is concerned, it does not matter whether the request first goes to a method that then operates on the state variables of objects passed as parameters, or whether it goes to any particular object in the parameter list.

• Parameter encoding. These facilities convey the local representation of parameter values in the requester's environment to equivalent representations in the recipient's environment. To accomplish this, parameter encodings may employ standards or de facto standards (e.g., OSF/DCE, ONC/NFS/XDR, NCA/NCS/NDR, ASN.1).
• Delivery. Requests and results must be delivered to the proper location as characterized by a particular node, address, space, thread, entry point. These facilities may use standard transport protocols (e.g., TCP/UDP/IP, ISO/TPn).

• Synchronization. Synchronization primarily deals with handling the parallelism of the object’s making and processing a request and the rendezvousing of the requester with the response to the request. Possible synchronization models include: asynchronous (request with no response), synchronous (request; await reply), and deferred synchronous (proceed after sending request; claim reply later).

• Activation. Activation is the housekeeping processing necessary before a method can be invoked. Activation and deactivation ("passivation") of persistent objects is needed to obtain the object state for use when the object is accessed, and save the state when it no longer needs to be accessed. For objects that hold persistent information in non-object storage facilities (e.g., files and databases), explicit requests can be made to objects to activate and deactivate themselves.

• Exception handling. Various failures in the process of object location and attempted request delivery must be reported to requester and/or recipient in ways that distinguish them from other errors.
Object Request Broker (ORB)

Actions are needed to recover session resources and resynchronize requester and recipient. The ORB coordinates recovery housekeeping activities.

- Security mechanisms. The ORB provides security enforcement mechanisms that support higher-level security control and policies. These mechanisms ensure the secure conveyance of requests among objects. Authentication mechanisms ensure the identities of requesting and receiving objects, threads, address spaces, nodes, and communication routes. Protection mechanisms assure the integrity of data being conveyed, and assure that the data being communicated and the fact of communication are accessible only to authorized parties. Access enforcement mechanisms enforce access and licensing policies.

As an example of the function of the ORB, consider the request print layout_312 laser_plotter. This could be sent to the object layout_312 whose print method would then print it on laser_plotter. Or the request could be sent to laser_plotter whose print method would access layout_312. Or the request could be sent to a generalized print routine that would figure out a good way to arrange the printing, based on some attributes of these two objects. Or, instead of relying on a generalized print routine, the Name Service in the ORB could determine an appropriate method jointly owned by (the classes of) layout_312 and laser_plotter.
5.3.1 *Common Object Request Broker Architecture Specification*

In September 1991, the OMG selected a standard interface for the ORB component of the Object Management Architecture. This ORB standard, adopted from a joint proposal of Digital Equipment Corporation, Hewlett Packard Company, HyperDesk Corporation, NCR Corporation, Object Design Inc., and SunSoft Corporation, is called CORBA. It is outlined in detail in *CORBA: Common Object Request Broker Architecture and Specification*. In 1992, the CORBA specification was revised. In 1995, the CORBA specification was again updated to support the following items:

- Mappings to the C++ and Smalltalk programming languages
- Extended interface repositories
- Portable initialization procedures
- Full Object Request Broker interoperability, including transactioning and security

The most important feature of any CORBA specification is its Interface Definition Language (OMG IDL). The OMG IDL language is used by applications to specify the various interfaces they intend to offer to other applications via the ORB layer. Application may make use of this interface specification information to access local or
remote services in both a static fashion (high-performance, compile-time, optimized), or dynamically (with much greater flexibility).

5.4 Object Services

Object Services provides basic operations for the logical modeling and physical storage of objects.

Object Services defines a set of intrinsic or root operations that all classes should implement or inherit. Objects do not have to use the implementation of basic operations provided by Object Services, nor do objects have to provide all basic operations. For example, an object may provide its own data storage; an object that models a "process" may not provide transactions.

The operations provided by Object Services are made available through the ORB. Object Services may also be made available through other interfaces. For example, there may be additional interfaces that comply with non-OMG standards or that are optimized for higher performance. Stated differently, Object Services does not impose a single implementation or interface, rather, it defines at least one interface that can be used regardless of an object’s realization and regardless of other interfaces provided by the infrastructure.
The operations provided by Object Services can serve as the building blocks for extended or augmented functionality provided by Common Facilities. For example, Object Services can provide transaction management that spans objects, implementations, and machines. Certain aspects of a function can more easily be provided by software that is intrinsically concerned with controlling an object, while the more generalized, abstract, or multi-object implementation of a function can be better provided by Common Facilities.

The operations that Object Services can provide include:

- **Class management.** The ability to create, modify, delete, copy, distribute, describe, and control the definitions of classes, the interfaces to classes, and the relationships between class definitions.

- **Instance management.** The ability to create, modify, delete, copy, move, invoke, and control objects and the relationships between objects.

- **Storage.** The provision of permanent or transient storage for large and small objects, including their state and methods.

- **Integrity.** The ability to ensure the consistency and integrity of object state both within single objects (e.g., through locks) and among objects (e.g., through transactions).
Object Services

- Security. The ability to provide (define and enforce) access constraints at an appropriate level of granularity on objects and their components.
- Query. The ability to select objects or classes from implicitly or explicitly identified collections based on a specified predicate.
- Versions. The ability to store, correlate, and manage variants of objects.

The types of subcomponents that could be used to implement Object Services include object-oriented database management systems, transaction managers, query facilities, directory services and file services.

5.4.1 Object Services Specification

Specifications for Object Services typically consist of a set of OMG IDL interface definitions (syntax) and a description of operation behavior and request sequencing (semantics). Specifications for the following Object Services are contained in the CORBA services manual:

- Naming
- Event
- Life Cycle (including Compound Life Cycle)
5.4.2 Summary of Object Services

This section provides a brief description of each Object Service.

The Naming Service provides the ability to bind a name to an object relative to a naming context. A naming context is an object that contains a set of name bindings in which each name is unique. To resolve a name is to determine the object associated with the name in a given context. Through the use of a very general model and in dealing with names in their structural form, Naming Service implementations can be application specific or be based on a variety of naming systems currently available on system platforms.

Graphs of naming contexts can be supported in a distributed, federated fashion. The scalable design allows the distributed, heterogeneous implementation and administration of names and name contexts.
Because name component attribute values are not assigned or interpreted by the Naming Service, higher levels of software are not constrained in terms of policies about the use and management of attribute values.

The Event Service provides basic capabilities that can be configured together flexibly and powerfully. The service supports asynchronous events (decoupled event suppliers and consumers), event “fan-in,” notification “fan-out,”—and through appropriate event channel implementations—reliable event delivery.

The Event Service design is scalable and is suitable for distributed environments. There is no requirement for a centralized server or dependency on any global service. Both push and pull event delivery models are supported; that is, consumers can either request events or be notified of events.

Suppliers can generate events without knowing the identities of the consumers. Conversely, consumers can receive events without knowing the identities of the suppliers. There can be multiple consumers and multiple suppliers of events. Because event suppliers, consumers, and channels are objects, advantage can be taken of performance optimizations provided by ORB implementations for local and remote objects. No extension is required to CORBA.
The Life Cycle Service defines services and conventions for creating, deleting, copying, and moving objects. Because CORBA-based environments support distributed objects, life cycle services define services and conventions that allow clients to perform life cycle operations on objects in different locations. The client’s model of creation is defined in terms of factory objects. A factory is an object that creates another object. Factories are not special objects. As with any object, factories have well-defined OMG IDL interfaces and implementations in some programming language.

The Life Cycle Service also supports compound life cycle operations on groups of related objects.

The Persistent Object Service (POS) provides a set of common interfaces to the mechanisms used for retaining and managing the persistent state of objects. The object ultimately has the responsibility of managing its state, but can use or delegate to the Persistent Object Service for the actual work. A major feature of the Persistent Object Service (and the OMG architecture) is its openness. In this case, that means that there can be a variety of different clients and implementations of the Persistent Object Service, and they can work together. This is particularly important for storage, where mechanisms useful for documents may not be appropriate for employee databases, or the
mechanisms appropriate for mobile computers do not apply to mainframes.

The Transaction Service supports multiple transaction models, including the flat (mandatory in the specification) and nested (optional) models. The Transaction Service supports interoperability between different programming models. For instance, some users want to add object implementations to existing procedural applications and to augment object implementations with code that uses the procedural paradigm. To do so in a transaction environment requires the object and procedural code to share a single transaction. Network interoperability is also supported, since users need communication between different systems, including the ability to have one transaction service interoperate with a cooperating transaction service using different ORBs.

The Transaction Service supports both implicit (system-managed transaction) propagation and explicit (application-managed) propagation. With implicit propagation, transactional behavior is not specified in the operation’s signature. With explicit propagation, applications define their own mechanisms for sharing a common transaction.

The Transaction Service can be implemented in a TP monitor environment, so it supports the ability to execute multiple transactions.
concurrently, and to execute clients, servers, and transaction services in separate processes.

The Concurrency Control Service enables multiple clients to coordinate their access to shared resources. Coordinating access to a resource means that when multiple, concurrent clients access a single resource, any conflicting actions by the clients are reconciled so that the resource remains in a consistent state.

Concurrent use of a resource is regulated with locks. Each lock is associated with a single resource and a single client. Coordination is achieved by preventing multiple clients from simultaneously possessing locks for the same resource if the client’s activities might conflict. Hence, a client must obtain an appropriate lock before accessing a shared resource. The Concurrency Control Service defines several lock modes, which correspond to different categories of access. This variety of lock modes provides flexible conflict resolution. For example, providing different modes for reading and writing lets a resource support multiple concurrent clients on a read-only transaction. The Concurrency Control service also defines Intention Locks that support locking at multiple levels of granularity.

The Relationship Service allows entities and relationships to be explicitly represented. Entities are represented as CORBA objects. The service defines two new kinds of objects: relationships and roles.
A role represents a CORBA object in a relationship. The Relationship interface can be extended to add relationship-specific attributes and operations. In addition, relationships of arbitrary degree can be defined. Similarly, the Role interface can be extended to add role-specific attributes and operations. Type and cardinality constraints can be expressed and checked: exceptions are raised when the constraints are violated.

The Life Cycle Service defines operations to copy, move, and remove graphs of related objects, while the Relationship Service allows graphs of related objects to be traversed without activating the related objects. Distributed implementations of the Relationship Service can have navigation performance and availability similar to CORBA object references: role objects can be located with their objects and need not depend on a centralized repository of relationship information. As such, navigating a relationship can be a local operation.

The Externalization Service defines protocols and conventions for externalizing and internalizing objects. Externalizing an object is to record the object state in a stream of data (in memory, on a disk file, across the network, and so forth) and then be internalized into a new object in the same or a different process. The externalized object can exist for arbitrary amounts of time, be transported by means outside of the ORB, and be internalized in a different, disconnected ORB.
For portability, clients can request that externalized data be stored in a file whose format is defined with the Externalization Service Specification.

The Externalization Service is related to the Relationship Service and parallels the Life Cycle Service in defining externalization protocols for simple objects, for arbitrarily related objects, and for facilities, directory services, and file services.

5.4.3 Common Facilities (CF)

Common Facilities comprises facilities that are useful in many application domains and which are made available through OMA-compliant object interfaces. Unlike Object Services, which will be supported on all platforms, Common Facilities is optional. Not all standardized facilities will be available on all platforms, but, if available, they will provide the OMG approved semantics.

For application developers, Common Facilities reduces the effort needed to build OMA-compliant applications. For example, an OMA-compliant CAD schematic editor could use a common OMG Help facility to provide end-user help. In addition, application developers may define subclasses to enrich or customize the functionality of Common Facilities for specific applications.
For end users, Common Facilities provides uniform semantics that are shared across applications, making OMA-compliant applications easier to use.

A service becomes a Common Facility when it:

- Communicates using the ORB
- Implements a facility that OMG chooses to adopt
- Has an OMA-compliant object interface

The following list shows the Common Facilities that have been identified by OMG and its membership as candidates for the Common Facilities category:

User Interface Common Facilities

- Rendering management (such as printing and display)
- Compound presentation management (such as printing and display in compound documents)
- User support facilities (such as text checking)
- Desktop management (facilities for end user desktop)
- Scripting (interactive creation of automation scripts)

Information Management Common Facilities

- Information modeling
Object Services

- Information storage and retrieval
- Data interchange and compound data interchange
- Information exchange
- Data encoding and representation (data format encodings and translations)
- Time operations (manipulation of calendar and time data)

Systems Management Common Facilities
- Management tools (interoperability of such tools)
- Collection management (integration of collection management facilities and managed objects)
- Control facilities (control of system resources and managed objects)

Task Management Common Facilities
- Workflow (coordination of objects in a work process)
- Agent (supporting static and dynamic agents)
- Rule management (knowledge acquisition, maintenance, execution of rule-based objects)
- Automation (access to functionality of one object from another object)
Vertical Market Facilities, including facilities for imagery, information, CIM, distributed simulation, oil and gas industry exploitation and production, accounting, application development, and mapping.

5.5 Application Objects (AO)

The AO classification corresponds to the traditional notion of an application. Generally developed by VARs (Value-Added Resellers) and ISVs (Independent Software Vendors), AOs represent individual related sets of functionality (e.g., word processing and stock ticker display). The value of developing these applications inside OMA is the improved ability to wed that single-minded functionality with other classes. The ability to integrate application classes also extends to the integration of traditional, extant, non-object–oriented applications within the same framework. Thus an existing spreadsheet product could be fully integrated into an OMA-compliant system using embedding.

It is important to realize that classes that fall into the AO classification are at the same OMA semantic level as CF classes; the difference is that CF classes represent very common functionality (which is adopted by the OMG as a standard), while AO classes are more specialized, representing interfaces that are specific to an application.
application domain and not standardized by the OMG. However, specific applications may be structured along standards arising from other standards organizations such as ISO and ANSI.

An OMA-compliant application consists of a collection of interworking objects. Classes and objects may be used in multiple applications. For example, a given class could provide shared services, perhaps specific to a particular application domain. Existing applications and data elements (files, databases, etc.) may be embedded in classes and objects.

A long list of possible programs is expected to fall within the AO classification, including:

- Office applications: word processing, spreadsheets, electronic mail, etc.
- CAD applications: EDA, ECAD, MCAD, CAE, architectural CAD, etc.
- CASE tools: programming support, database design tools, etc.
- Network management applications.
- Information access and query applications. The entire range of query systems would fall into the AO classification, including general database query, geographic information systems, reservation transaction systems, etc.
Application Objects (AO)

- Knowledge-based systems.

It is important to note that individual AO-classified classes can "migrate" into the CF classification if a commonality of interface becomes apparent. In addition, in new programs one might expect traditional applications to extend across boundaries; for instance, the query portion of a PC database product might be found in the AO classification in a port to a more OMA-oriented style.

Examples of possible requests in the AO classification include:

- `lookup_spreadsheet_cell(spreadsheet_object,cell_location)` to read and return the value of a cell in a spreadsheet object.

- `edit_ascii_text(editor_object,text_string)` to run a text editor on some text to be edited, allowing user interaction with an editing tool represented by an editor object.

- `query_database(query_object,sql_string)` to do database lookup through the good graces of an SQL-compliant database query processor.
Figure 5-1 Reference Model: OMA Overview.

Figure 5-2 Wrapping Existing Applications