In *ACM Operating Systems Review*, **Vol.30, No. 2. Pages 4-31. April 1996.**

CORBA: A Platform for Distributed Object Computing

(A State-of-the-Art Report on OMG/CORBA)

Zhonghua Yang and **Keith Duddy**

CRC for Distributed Systems Technology (DSTC)
Level 7 Gehrmann Laboratories
University of Queensland
Australia 4072

email: {**yang, dud**}**@dstc.edu.au**

Abstract

Distributed computing has made significant advances in the past few years as witnessed by the release of OSF's Distributed Computing Environment (DCE) and the adoption of a set of specifications for distributed object computing by the Object Management Group (OMG). In this report, we discuss OMG's work in distributed computing based on object technology. We attempt to provide a comprehensive coverage of OMG's technology, including an overview of Object Management Architecture, the rationale behind OMG's work, a technical tour of CORBA, and an overview of Object Services and Common Facilities. A technical comparison between DCE and CORBA is provided. User perspective and the future potential of OMG's technology are discussed in the conclusion.

1 Introduction

The Object Management Group (OMG) is an international industry consortium that promotes the theory and practice of object oriented (OO) software development. Their goal is to provide a common architectural framework, across heterogeneous hardware platforms and operating systems, for inter-communication of application objects.

The adoption of an object-oriented approach is motivated by the desire for software development with reusable components that interact with one another through well defined interfaces. Industry trends indicate that object technology is gaining wide acceptance, with most vendors now providing tools and environments to support OO software development. Just as structured modular programming became standard software engineering practice in the 1980s it is anticipated that this will give way to a dominance of the object-oriented paradigm within a few years.

The OMG was founded in May 1989, by eight companies: 3Com Corporation, American Airlines, Canon Inc., Data General, Hewlett-Packard, Philips Telecommunications N.V., Sun Microsystems and Unisys Corporation. The OMG began independent operations as a non-profit corporation in October 1989 [8]. It now boasts over 500 member companies internationally, with almost every significant computer vendor represented.

The organisation performs no development work, relying rather on existing technologies offered by member companies. Its approach is to issue Requests for Proposals (RFPs) on all aspects of object technology, which solicit specifications of components to fit into a broad Object Management Architecture (OMA). Members may then propose a specification that is accompanied by an implementation proving the concepts detailed therein. A review and voting process is conducted and once a specification is accepted, any vendor that conforms to it may offer alternative implementations to market.

2 Overview of OMG OMA

From the outset, the OMG realized that one of the main problems facing distributed computing is software component integration, and there is no single, commercially available, widely recognised and standardised approach and framework for achieving this integration. In its book *Object Management Architecture Guide* [8, page 30], the OMG offered its vision and approach to distributed computing:

> *To adopt interface and protocol specifications that define an object management architecture supporting interoperable applications based on distributed interoperable objects.*
>
> *The specifications are to be based on existing technology that can be demonstrated to satisfy OMG's Technical Objectives.*

The OMG has developed a conceptual model, known as the core object model, and a reference architecture, called the Object Management Architecture (OMA) upon which applications can be constructed. The OMG OMA attempts to define, at a high level of abstraction, the various facilities necessary for distributed object-oriented computing. The OMG OMA partitions the OMG problem space into practical, high-level architectural components that can be addressed by technology proposers. It consists of four components: *Object Request Broker (ORB), Object Services (OS), Common facilities (CF)*, and *Application Objects (AO)* (Figure 1). These components define the composition of objects and their interfaces. Objects are categorised into Object Services, Common Facilities, and Application Objects to establish the standardisation strategy for the OMG.

The core of the OMA is the Object Request Broker (ORB) which is a common communication bus for objects. The technology adopted for ORBs is known as the Common Object Request Broker Architecture (CORBA) [29, 27, 43], which specifies a framework for transparent communication between application objects. CORBA is the first specification adopted by the OMG. The latest version is CORBA2 [27], adopted at the end of 1994.

Object Services is comprised of a set interfaces to lower level important infrastructure objects. The OMG Object Services specifications define sets of objects which perform fundamental functions such as naming, life cycle services, and transactions [24]. So far, the OMG has adopted two volumes of Object Services specifications, the Common Object Services Specification 1 and 2 (COSS 1 [15, 20]; COSS 2 [16, 20]).

Common Facilities (CF) [13] are the most recent area of efforts by the OMG. OMA defines Common Facilities as having an application focus and mostly used by developers working in a distributed environment. Common Facilities provide standardised interfaces to common application interfaces.

The Application Objects component reflects the need for independently developed application interfaces. While this is one of the components of the OMA, the OMG will probably never develop standards for it.

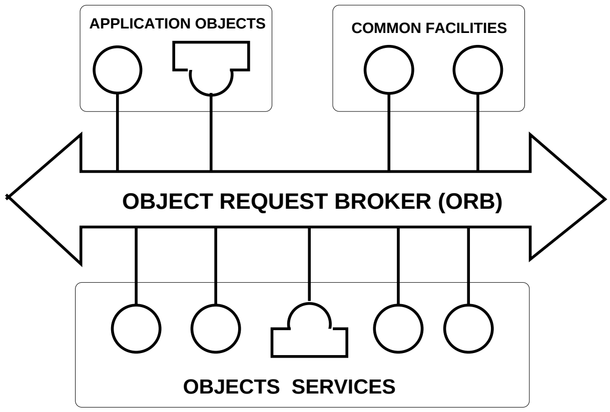

Figure 1: OMG's Object Management Architecture.

The OMA architecture is shown in Figure 1.

Underlying the OMA, and OMG's work in general, is OMG's object model [23] which is not shown in Figure 1. It is described briefly in the next section.

3 Rationale of OMG Technology

As we have mentioned, OMG offers a single view of distributed computing and develops the reference architecture (OMA) based on existing object technology. Underlying the OMA is a classical object model. In this section, we explore OMG technology with reference to a number of issues:

1). object model
2). object reference
3). object granularity
4). inheritance
5). multiple interface objects
6). static and dynamic invocation interfaces
7). interaction models
8). interoperability
9). legacy systems

3.1 Object models

The diversity of application domains means that there is no single common object model that can satisfy the requirements of all applications. The OMG has taken the approach of defining a core object model as a base for further extension for different domains. OMG's object model [23] (the core object model, extensions, and profiles) defines a common object semantics for specifying the characteristics of objects in a standard and implementation independent way. The object model defines a set of requirements that must be supported by all OMG-compliant systems.

The *core object model* is the highest level of object model. It defines a set of concepts commonly needed to define other object models. The OMG core object model was first designed in 1992, and called Core 92 model. An *extension*, as the name implies, is an extension to the core object model and introduces a further set of concepts. A *profile* is some combination of the core model

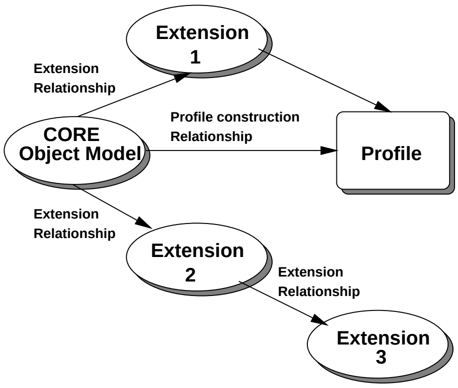

Figure 2: OMG's Modelling concepts.

with one or more extensions, and provides a useful set of concepts for specifying applications. For example, ORB profile (CORBA), ODBMS profile, Object Analysis and Design Model [23]. This can be summarised as follows (ref. Figure 2.)

$$\textit{Core + Extension(s) = Profile.}$$

The Core 92 model defines the concepts within its scope at three different levels [23]:

- Precisely defined concepts:

```
Object                      Operation
Signature                   Operation Name
Parameter                   Parameter Type
Result                      Non-Object
Object interface            Type (Partial)
Interface inheritance       Request
```

- Imprecise definitions:

```
Object identity             Exception
Behaviour                   Values
Denotable Values            Type (the Full definition)
Supertyping                 Subtyping
Refinement                  Collective behaviour
Response information        Substitution
```

- Dictionary definitions

  Those concepts that can merely be defined in a dictionary and the model does not give its own specific definitions.

Core 95

Core 92 model has an extension called Core 95 which identifies the concepts required to improve the portability and interoperability of systems by describing the behaviour of objects. Most of the components of the extension are still under discussion. The concepts expected to be part of a complete Core 95 are [23]:

- Object identity.

  In Core 92, there is no notion of unique object identity, an object reference is used to locate an object instead.

- Multiple interfaces per object and type conformance.

- Improvements in the definition of interfaces and their dynamic behaviour.

- Stream interfaces.

- Binding.

- Services.

These enhancements are inspired by the Open Distributed Processing (ODP) work [33].

This *Core + Extension(s)* approach is advantageous because:

- it avoids putting everything in the Core.

- it leads to definitions of compatible object model extensions.

- it is natural to make extensions to meet new requirements.

3.2 Object Reference and Interoperable Object Reference (IOR)

Whereas Microsoft's COM (*Component Object Model*) [7] uses globally unique identifiers (128-bit integer), objects in OMG model are identified by *object references* – an implementation defined type guaranteed to identify the same object each time the reference is used in a request. While CORBA does not specify how references are to be implemented, it explicitly states that references are not guaranteed to be unique.

That the OMG chose not to define a Universal Unique IDentifier (UUID) is not without reason: it has some concerns about implementation efficiency, management and interaction with legacy applications that have different approaches to object IDs. An object UUID which is guaranteed unique in a distributed environment may be larger for some distributed objects than the actual representation of the state of the object.

CORBA provides a very abstract view of objects. In CORBA, the *object* exists only as an abstraction. It is visible through operations defined by an interface and invoked using an *object reference.* The object reference itself is an abstraction, and is available to clients only as an opaque value which can be passed as a parameter in a operation, or stringified to give an external representation that may be stored and subsequently turned back into an object reference.

The interface determines the operations that a client may perform using the object reference [46]. The only way a client can observe or affect the object is through operations defined for that interface. The rest of the details of the object are *hidden.* Thus, the object presented to the client via its interface is an *abstract* object. (Figure 3).

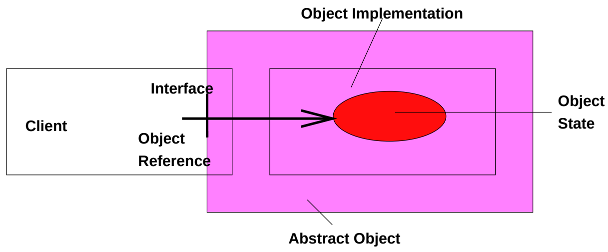

Figure 3: A CORBA object.

CORBA does not define an object reference comparison function, since it requires access to reference data on the client side. To handle comparison, explicit operations ("equal", "lessThan") must be defined on interfaces.

CORBA does not specify how an object reference is to be implemented by ORBs. Every ORB may implement it in whatever way is convenient. This gives rise to an interoperability problem. For an object to make a request to an object within another ORB domain, ORBs must understand the object reference passed over from the foreign ORB. This interoperability problem is solved in CORBA2 [27] where a data structure, *Interoperable Object Reference (IOR)*, has been specified. An interoperable object reference is a sequence of object-specific protocol profiles, plus a type ID. It is only used when crossing object reference domain (ORB) boundaries. IORs need not be used *internally* by any given ORB, and are not intended to be *visible* to application level ORB programmers.

3.3 Fine-grained vs. coarse-grained objects

There are no obvious architectural arguments about CORBA only supporting fine- or coarse-grained objects. In this respect, the CORBA specification does state explicitly that it supports *a wide range of granularities* [27, page 2-9]. From a programmer's viewpoint, objects in CORBA tend to be much larger than they are at programming language level. In an extreme case, a whole application could be a CORBA object. This flexibility is necessary for CORBA to support non-object-oriented applications. But for new distributed object applications, objects would normally be more finely grained.

3.4 Interface inheritance

Among the three key ingredients of the object-oriented paradigm (*encapsulation, inheritance*, and *polymorphism*) [46], CORBA does not support polymorphism, and encapsulation and inheritance are supported in a restricted way. Inheritance (including multiple inheritance) is restricted to interface inheritance only and provides no overriding or specialisation of operations. Inheritance of object interfaces is specified using OMG Interface Definition Language (IDL) [27]. Since the OMG's goal is to establish specifications for distributed object computing based on existing technology, it has no concern about how they are implemented, *implementation inheritance* is therefore not considered. The ORB has no knowledge that a set of services accessed through an interface hierarchy is can also be related by implementation inheritance. On the other hand, nothing in CORBA prevents the developer of a set of service objects from using implementation inheritance. In fact, some vendors' implementations (for example, IBM's DSOM) support implementation inheritance [32]. In addition, function (operation) overloading is unsupported in the current CORBA.

3.5 Single vs. multiple interface(s) objects

In the current OMG Object Model and CORBA, one interface only is defined for each object, and thus terms *object* and *interface* are synonyms. While in many applications it is desirable to aggregate several interfaces into an entity, there is no significant difference between single interface and multiple interface objects from the viewpoint of an application developer. If a multiple interface object is needed, interface inheritance can be used to provide the *composition mechanism* for permitting an object to support multiple interface.

3.6 Static and dynamic invocation interfaces

Operation invocations can be through either *static* or *dynamic* interfaces. The static invocation interfaces are determined at compile time, and are presented to client codes using *stubs*, hence also called *stub-style* invocation. On the other hand, the dynamic invocation interface (DII) allows clients to construct and issue a request whose signature is possibly not known until run time. That is, the request is fully constructed at run time using information from the interface repository. Dynamic invocation interfaces are more flexible, and is best used when a client application wants to be able to discover new object types at run-time. Dynamic invocation uses the interface repository to validate and possibly retrieve the signature of the operation on which a request is made. CORBA supports both static and dynamic interfaces. The client side and server side dynamic invocation interfaces are necessary mechanisms for CORBA interoperability.

3.7 CORBA interaction models

CORBA is an architecture for distributed objects. At the lower communication level, the object interaction takes the form of Remote Procedure Calls (RPCs), which are synchronous invocations. In addition, CORBA allows asynchronous interaction by specifying a *oneway* operation semantics in which the client continues its computation without waiting for a result from the server after issuing the request. The *oneway* operation is a request-only operation with *best-effort* semantics, i.e. it does not return any result and the requester never synchronises with the completion (if any) of the request.

By using the Dynamic Invocation Interface (DII), CORBA supports *deferred synchronous interactions* in which the client is allowed to get a response from the server some time after issuing the request (Figure 4).

A client is able to invoke an *at-most-once* operation in a synchronous or deferred-synchronous manner.

Table 1 shows the relationship between invocation types and communication styles.

3.8 ORB interoperability

As early as the release of CORBA 1.0 in 1991, the OMG was criticised for failure to provide *out-of-the-box* multi-vendor interoperability between ORB implementations. In the CORBA 1.x specification, there are a number of areas which are deliberately vague or non-specific for the

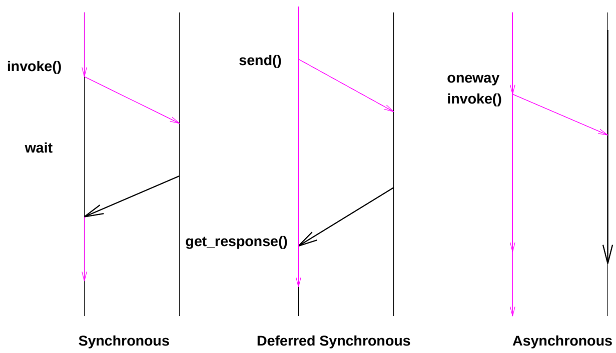

Figure 4: CORBA Interaction Models.

Table 1: Communication styles supported by invocation types

Communication Styles	Static Invocation	Dynamic Invocation
Synchronous	Yes	Yes
Deferred Synchronous	No	Yes
Oneway (Asynchronous)	Yes	Yes

purpose of allowing a wide range of implementations. However, this vagueness and lack of specificity leads to a problem if one is trying to build a system of interoperable ORBs where the ORBs have different implementations by different vendors. This is the *interoperability problem* in CORBA 1.x that took the OMG a few years to solve.

After extensive debate, this problem was finally resolved by CORBA2. In order to guarantee that two separately developed CORBA implementations will work together, a single *wire-format* must be defined. A wire-format specifies how detailed information representing a CORBA request is laid out on a network transport service. The hot debate centred around which wire-format was to be adopted. There were two camps: one insisted that the wire format should be based on OSF DCE RPC (characterised as *tried-and-true*), while the other maintained that there is a need for *simple and object-oriented format*, called UNO GIOP (*Universal Networked Objects, General Inter-ORB Protocol*).

The ultimate outcome of this debate is not purely based on technical reasoning. Rather, a compromise was reached [28]:

- CORBA2 implementations will provide guaranteed out-of-the-box interoperability by means of UNO GIOP hosted on the TCP/IP networking transport service, called IIOP (Internet Inter-ORB Protocol).
- CORBA2 implementations will have the option of supporting an additional protocol, called the DCE CIOP (DCE Common Inter-ORB Protocol) based on OSF's DCE protocol.

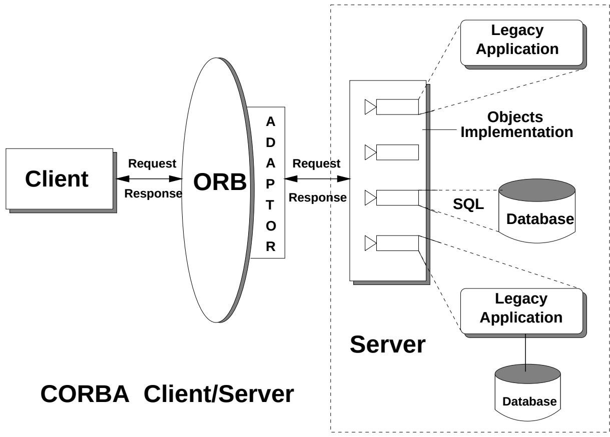

Figure 5: CORBA Client/Server.

3.9 Legacy systems

Many large institutions are held back by existing information systems developed some years ago. These systems are mission-critical and must remain operational at all times; they are typically large and written in a legacy language, and use legacy database management systems. These systems pose a serious problem for large organisations [4, 3]. A fundamental requirement for new platforms is to support legacy system migration such that the migrated target system remains fully operational in new environments.

CORBA provides a distributed object-oriented approach to integrating legacy applications and enables the developers to write an *object wrapper* to encapsulate the legacy system. With CORBA, the developer writes a description of the service provided by the legacy system using IDL, then writes code which invokes the appropriate set of actions in the legacy application when one of the object wrapper's methods is invoked (ref. Figure 5).

This approach has been proved very efficient and effective [4, 48, 40].

4 A Tour of the CORBA Specification

As mentioned above, an ORB is a communication mechanism between interacting objects (ref. Figure 5). It includes all functions required to support distributed computing, such as location of objects, marshalling of request parameters and results, and object referencing.

In this section, we provide a tour to the major components of the CORBA specification [27, 43]. The issues covered include:

1). Interface Definition Language (IDL)

2). CORBA Interface Architecture

3). ORB Core

4). CORBA Repositories

5). ORB Interoperability

6). What is CORBA2?

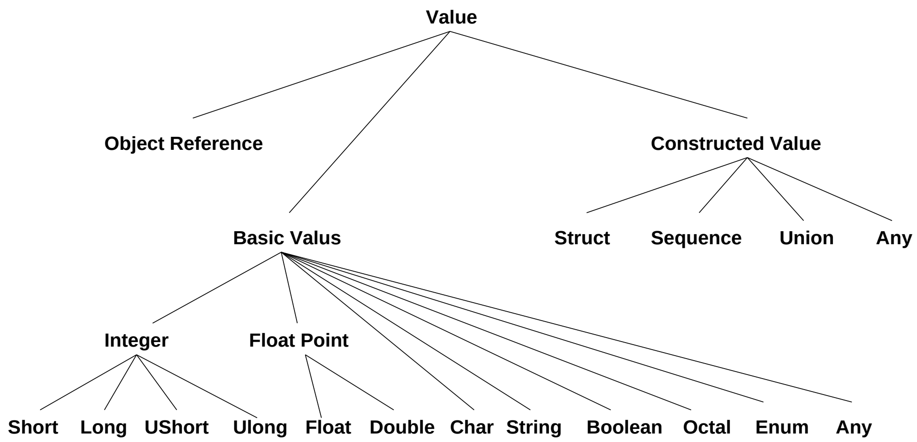

Figure 6: OMG's IDL type hierarchy

4.1 Interface Definition Language (IDL)

The Interface Definition Language (IDL) defines the types of objects by specifying their interfaces. It is required that all OMG services be specified using a declarative language emphasising the separation of interface and implementation. It is programming language neutral and network neutral, and thus it is used as a means of describing *data types*. It provides a two-tier encapsulation system: data-types (basic and user-defined), and objects, which allows for sophisticated modelling of a distributed domain. The syntax of OMG IDL is derived from C++, removing the constructs of an implementation language and adding a number of new keywords required to specify distributed systems. OMG IDL's type hierarchy is shown in Figure 6.

OMG IDL is used to *statically* define the interfaces to objects, to allow invocation of operations on objects with differing underlying implementations. From the IDL definitions, it is possible to map CORBA objects into particular programming languages or object systems.

Listing 1 is an example IDL definition that shows what an OMG IDL specification looks like. As can be seen, an interface consists of a set of named operations and the parameters to these operations.

4.2 CORBA interface architecture

The CORBA specification defines an architecture of interfaces consisting of three specific components: *client-side interface, object implementation side interfaces*, and *ORB Core* (Figure 7).

4.2.1 Client-side interface

The client-side architecture provides clients with the following interfaces to the ORB and to server objects:

a) IDL stubs.

The IDL stub presents interfaces comprised of functions generated from IDL interface definitions and linked into the client program. This is the static invocation interface, representing a language mapping between the client language and the ORB implementation. Currently adopted language mappings are for C, C++, and Smalltalk. The IDL stub interface brings the ORB right into the application programmer's domain: the client interacts with (remote) server objects by invoking their operations just as it would invoke operations of local objects.

```
// OMG IDL specification example: myBank.idl
module BANK {

   interface BankAccount {
      // types
      enum account_kind {checking, saving};

      // exceptions
      exception account_not_available {string reason;};
      exception incorrect_pin {};

      // attributes
      readonly attribute float balance;
      attribute account_kind what_kind_of_account;

      // operations
      void access (in string account, in string pin)
         raises (account_not_available, incorrect_pin);
      void deposit (in float f, out float new_balance)
         raises (account_not_available);
      void withdraw (in float f, out float new_balance)
         raises (account_not_available);
   };  // end of interface  BankAccount
};  // end of module BANK
```

Listing 1. An example of OMG IDL definition

b) Dynamic Invocation Interface (DII).

While most clients in the CORBA world probably use stubs to invoke object services, the ORB provides a dynamic invocation interface mechanism for specifying and building a request at run time, rather than calling linked-in stubs. The dynamic interface is necessary when the object interface cannot be known at compile time. Using the DII mechanism, an object is accessed by a call to the ORB or by a series of calls to the ORB in which the object, method, and parameters are specified. It is the client's responsibility to specify the types of the parameters and expected results.

The operations that support the DII include: *create_request, invoke, send, get_response.* Requests made by static and dynamic invocation are indistinguishable by the server object.

Using the DII, the interaction between client and server object can be RPC-like synchronous as well as *Deferred Synchronous.* In the latter case, the caller may choose whether or not to wait for the response. This allows asynchronous interaction between caller and callee.

c) ORB interface.

The ORB interface allows functions of the ORB to be accessed directly by the client code. As it is currently specified, this interface provides only a few operations, such as stringifying an object reference. The ORB interface is the one component that is shared by both the client-side and the implementation-side architecture.

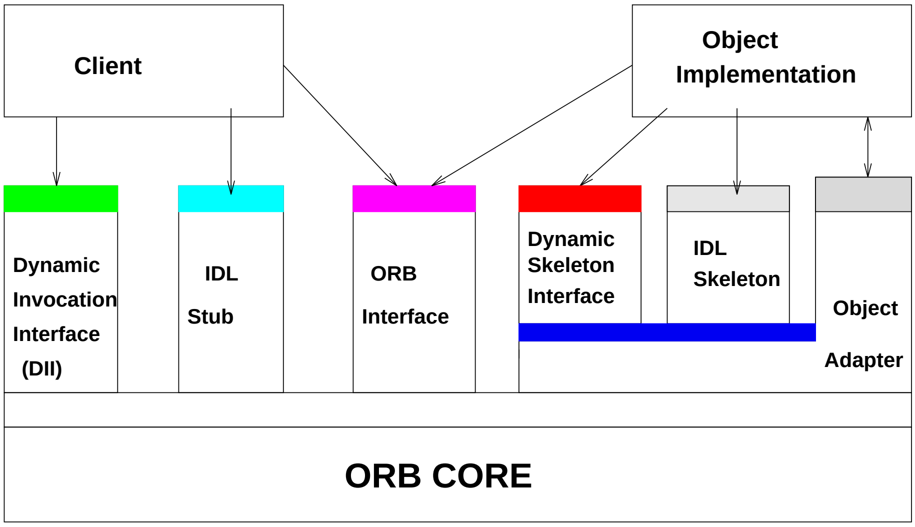

Figure 7: The structure of ORBA interfaces.

4.2.2 Implementation-side interface

The implementation-side interfaces consist of the following *up-call* interfaces, allowing calls from the ORB up to object implementations:

a) IDL skeleton.

This is the server-side counterpart of the IDL stub interface.

b) Dynamic skeleton Interface (DSI).

The DII as specified in CORBA 1.2 is a client-side mechanism but there is no corresponding server-side dynamic invocation mechanism. In CORBA2, as required by ORB-interoperability, the *Dynamic Skeleton Interface* (DSI) is introduced [28]. The DSI is the server side's analogue to the client side's DII, and provides a run time binding mechanism for servers to deliver request from an ORB to an object implementation that does not have compile-time knowledge of the type of the object. This contrast with type-specific, IDL-based skeleton, but serves the same architectural role. The dynamic skeleton inspects the parameters of an incoming request to determine a target object and method. This mechanism is very handy for building bridges between ORBs. When a client on one ORB calls a server on another ORB, the DSI transmits the request to the target ORB, and then the bridge uses the dynamic invocation interface to invoke the target object on that ORB. The DSI can receive either dynamic or static invocation from clients.

c) Object Adapter.

How does a CORBA server deal with millions of small objects? Who manages the object references? How are these references stored persistently? The answer to all these questions is the Object Adapter.

The Object Adapter sits on top of the ORB Core communication services, accepting requests for service on behalf of the server's objects. The object implementation accesses most

ORB services and the ORB Core through the Object Adapter. It provides the run-time environment for instantiating server objects, passing requests to them, and assigning object references to them. The Object Adapter is the means by which object implementations access most ORB services, including generation and interpretation of object references, method invocation, activation of an object, mapping references to implementations, etc.

The object adapter has three different interfaces:

- a private interface to the IDL skeleton.
- a private interface to the ORB Core.
- a public interface for use by the object implementations.

From these three kinds of interfaces it can be seen that the adapter is intended to isolate object implementations from the ORB Core as much as possible.

The OMG prefers not to see a proliferation of object adapters, so CORBA specifies a standard adapter called the Basic Object Adapter (BOA). The BOA is supplied by every ORB and can be used for most CORBA objects. The BOA includes interfaces for generating object references, registering implementations, activating implementations, and some security-related request (e.g. authentication).

d) ORB interface.

This interface is shared by the object implementations and the object clients (see **Client-side interface** above).

4.3 ORB Core

The ORB Core provides the basic representation of objects and communication requests. It moves a request from a client to an appropriate adapter for the target object. Since CORBA is intended to support a wide range of object granularity, object mechanisms, lifetime, policies, and implementation style, it would be difficult for the ORB Core to provide a single interface convenient and efficient for all objects. Rather, it structures the ORB with components above the ORB Core, which provide interfaces that can mask the difference between ORB Cores. For example, an *Object Adapter* targets a particular domain of object implementations that have similar requirements with interfaces tailored to them, and thus allowing the ORB Core to remain simple.

4.4 Repositories

The *interface repository* [22, 27] provides yet another way to specify the interfaces to objects. Interfaces can be added to an Interface Repository service which defines operations for run time retrieval of information from the repository. In any ORB implementation, the two approaches to defining interfaces (using IDL and using information in the interface repository), have equivalent expressive power.

Using an interface repository, a client should be able to locate an object unknown at compile time, enquire about its interface, and then build a request to be forwarded through the ORB.

The CORBA specification also mentions the existence of an *implementation repository*, but gives no specification for it. Ideally, the implementation repository contains information which allows the ORB to locate and activate object implementations to fulfill dynamic requests.

4.5 ORB interoperability

ORB-interoperability specifies a comprehensive, flexible approach to supporting networks of objects that are distributed across and managed by multiple, heterogeneous CORBA-compliant ORBs [27]. This approach is *universal* in that its elements can be combined in a variety of ways to satisfy a very broad range of ORB product and customer needs, hence called *Universal Networked Objects*, or *UNO* [28]. While enabling inter-ORB interoperability, the CORBA2 specification preserves the interface between clients and ORBs, and delegates responsibility for bridging between two object systems primarily to ORBs [28].

The elements of interoperability as specified in CORBA2 include:

- ORB interoperability architecture.
- Inter-ORB bridge support.
- General and Internet Inter-ORB Protocols (GIOPs and IIOPs).

In addition, the architecture accommodates *Environment Specific Inter-ORB Protocols (ESIOPs)* that are optimised for particular environments such as DCE.

The ORB Interoperability Architecture provides a conceptual framework for defining the elements of interoperability and for identifying its compliant points. The architecture clearly identifies the roles of different *domains* of ORB-specific information. Domains are joined by bridges, which map concepts in one domain to the equivalent in another. Full interoperability requires that all concepts used in one domain be translatable into concepts in the other. Obviously, the greater the commonality of the domains, the greater the degree of interoperability achieved. The OMG OMA and its common object model with interfaces defined in IDL ensures a high degree of commonality among independently developed ORBs.

The ORB Interoperability Architecture introduces the concepts of *immediate* and *mediated bridging* between ORB domains. A bridge that provides one-to-one protocol translation is called a *full bridge* which performs *immediate bridging*. This is a simple and effective solution as long as the number of protocols remains small.

In order to avoid an explosion in the number of bridges caused by increasing numbers of protocols needing to be translated, a *half bridge* and *mediated bridging* can be used. The inter-domain traffic may then cross two (half) bridges with a backbone network running a common protocol between them.

CORBA2 has designed a very basic inter-ORB protocol, called *General Inter-ORB Protocol (GIOP)*, which serves as a *common backbone protocol* so that the number of different combinations of “half bridges” needed between IOP domains is minimised (one per ORB domain).

The GIOP specifications are designed to be easily implementable on most transport systems. Internet IOP (Inter-ORB Protocol) is one of variants of the GIOP. All variants of the GIOP, such as the Internet IOP (IIOP), share common specifications for the following:

a) Common Data Representation (CDR) including marshalling conventions.

b) Interoperable Object Reference (IOR).

c) Interoperable TypeCodes.

d) IOP message content, format and semantics (independent of the method of message conveyance).

CORBA2 specifies IIOP as a mandated protocol for inter-bridge communication, and specifies an optional DCE RPC based ESIOP (Environment Specific Inter-ORB Protocol).

4.6 What is CORBA2?

CORBA2 is used to denote the family of component specifications that together represent the next major revision of CORBA 1.x. CORBA2 was adopted by OMG at the end of 1994. Additional specifications will be added to this family in the future, for example, the IDL/Ada95 mapping [21].

The CORBA2 family of *component* specifications consists of[1]:

- CORBA2/Core Common Object Request Environment.
- CORBA2/Interoperable.
- CORBA2 IDL language mappings.

The **CORBA2/Core** Common Object Request Environment consists of:

- CORBA 1.2 minus IDL C mapping. IDL C mapping is moved to CORBA IDL language mappings (see below).
- Extensions required to support RFP2 Object Services.
- Interface Repository extensions resulting from the *ORB 2.0 Interface Repository RFP*.
- ORB Initialisation resulting from the *ORB 2.0 Interface Repository RFP*.
- Inter-ORB Bridge Support.

The **CORBA/Interoperable** consists of:

- CORBA2/Internet IOP (Inter-ORB Protocol).
- Optional Environment Specific Inter-ORB Protocols (ESIOPS).

CORBA IDL language mapping is a stand-alone specification: no vendor is obliged to support any given language mapping, but if they do then they must comply with the OMG specifications for:

- CORBA2/C++
- CORBA2/SmallTalk
- CORBA2/C

[1] The CORBA2 specification components are also OMG-compliant components.

5 An Overview of OMA Object Services

Objects in OMG's OMA are divided into Object Services, Common Facilities, and Application Objects. The only purpose of this categorisation is to reflect the standardisation strategy for OMG.

Object Services are defined by OMG as those *"interfaces and sequencing semantics that are widely available and are mostly commonly used to support building well-formed applications in a distributed object environment built on a CORBA-compliant ORB."* In contrast to the Common Facilities, which will be discussed in the next section, Object Services provide fundamental (infrastructure-level) object interfaces necessary for building object-oriented distributed applications [24].

OMG Object Services have the following features:

- Operations provided by Object Services are expected to serve as building blocks for OMG Common Facilities or Application Objects.
- The OMG Object Services API is modular, and objects may use few or many Object Services.
- The Operations provided by Object Services are specified in IDL.

The OMG Object Services Task Force (OSTF) has so far issued five RFPs (Request For Proposal). RFP1 [10] and RFP2 [11] led to the adoption by the OMG of a set of specifications known as *Common Object Services Specification, Volumes 1 and 2 (COSS 1 and COSS 2).* COSS 1 and COSS 2 include services for *Object Naming, Object Events, Object Lifecycle, Persistent Object, Object Relationships, Object Externalisation, Object Transactions and Object Concurrency Control* [20]. Submissions to RFP3 [12] and RFP4 [17] which cover services for *Object Licensing, Object Properties, Object Query, Object Time, and OMA Security*, are currently being reviewed by the OSTF. The adopted specification for these services will become COSS 3 and COSS 4. RFP5 [25] covers the *Object Change Management Service, Object Collection Service, Object Trader Service, and Object Startup Service*, and was issued in June, 1995. RFP6 [26] is the latest RFP which covers the Object Change Management Service, and was issued October 1995. These services covered by RFP5 and RFP6 are expected to be adopted by OMG within 1996.

5.1 Adopted Object Services

The first two volumes of Object Service Specifications (COSS 1 and COSS 2) are now published as a single volume called *CORBAservices*.

5.1.1 COSS 1

The services described in this section are part of COSS 1 [15, 20].

(1) Object Naming Service.

This service supports a name-to-object association called a *name binding*. A name binding is always defined relative to a *naming context*. Different names can be bound to an object in the same or different contexts at the same time. There is no requirement that all objects must be named. A naming context is an object that contains a set of name bindings in which each name is unique.

This service addresses many design points identified for a name service:

- ◇ *Naming standards.*
  It allows encapsulation of existing naming standards using a naming context.
- ◇ *Federation of namespaces.*
  It supports distributed federation of name spaces.
- ◇ *Scope of names.*
  The name context defines the name scope.
- ◇ *Operations.*
  The service supports *bind, unbind, lookup,* and sequence operations on a name context. Rename operation is not supported.

(2) Object Event Notification Service.

This service supports notification of events to interested objects. It decouples the communication between objects. The event service defines two roles for objects: the *supplier* role (which produces event data) and the *consumer* role (which processes event data). Event data are communicated between suppliers and consumers by issuing standard CORBA requests.

The service defines two approaches to initiating event communication: the *push model* and the *pull model.* The push model allows a supplier of events to initiate the transfer of the event data to consumers. The pull model allows a consumer of events to request the event data from a supplier.

An *event channel* is an intervening object that allows multiple suppliers to communicate with multiple consumers asynchronously. Different event channel implementations will provide push or pull models (or a combination of these) and offer different qualities of service.

(3) Object Lifecycle Service.

This service represents a framework for creating, deleting, copying and moving objects based on location.

Any piece of code that initiates a lifecycle operation is a *client.* The client's model of creation is defined in terms of *factory* objects. A factory is an object that creates another object. However, there is no *standard* interface for a factory, only a *generic factory* interface. This service also defines *factory finder* objects which support one operation, `find_factories`, for a sequence of factories. Clients pass factory finders to the `move` and `copy` operations, which typically invoke this operation to find a factory to interact with. The new copy or the *migrated* object will then be within the scope of the factory returned from the factory finder.

(4) Persistent Object Service.

This service (POS) provides common interfaces to the mechanisms used for retaining and managing the persistent state of objects in a data store independent manner.

The object ultimately has the responsibility of managing its state, but can use or delegate to this service for the actual work. There is no requirement that any object use any particular persistence mechanism. However, the Persistent Object Service provides capabilities that should be useful to a wide variety of objects.

5.1.2 COSS 2

The services in this section are part of COSS 2 [16, 20].

1. Concurrency Control Service.

   This service defines how an object mediates simultaneous access by one or more clients such that objects it accesses remain consistent and coherent. It ensures that *transactional* and *non-transactional* clients are serialised with respect to one another.

   The Object Concurrency Control Service is designed as a service that can be used with the Object Transaction Service to coordinate the activities of concurrent transactions.

2. Externalisation Service.

   This service defines protocols and conventions for object externalisation and internalisation. Externalizing an object means to record the object state in a stream of data (in memory, on a disk file, across the network, etc.) which can subsequently be internalized into a new object in the same or a different process.

   Externalizing and internalising an object is similar to copying the object – the copy operation creates a new object which is initialised from the state of an existing object.

3. Object Relationship Service.

   This service provides for creating, deleting, navigating, and managing relationships between objects. This service defines two new kinds of objects: *relationship* and *roles*. A role represents a CORBA object in a relationship. A relationship is created by passing a set of roles to a relationship factory. This service allows relationships of arbitrary degree to be defined.

   The Object Relationship Service defines three levels of service:

   (a) The basic level defines relationships and roles.

   (b) When objects are related, they form a *graph of related objects*. The graph level extends the basic level service with *nodes* and *traversal objects*.

   (c) Specific relationships are defined by the third level. The Relationship Service defines two binary relationships at this level: *containment* and *reference* [16, 20].

4. Object Transaction Service.

   This service (OTS) provides support for ensuring that a computation consisting of one or more operations on one or more objects provides properties of *atomicity, consistency, isolation,* and *durability*. These properties are known as *ACID* properties.

   The service provides a transactional infrastructure and facilities to develop transactional classes of objects. It has two types of transactions:

   ◇ *flat* transactions.

     A flat transaction groups all operations within its scope into a single transactional entity.

   ◇ *nested* transactions (optional).

     A nested transaction is a transaction embedded within another transactions.

   The OTS depends on *Object Concurrency Control Service*, and *Persistent Object Service*.

5.2 Other Object Services

The following object services are part of the ongoing OMG standardisation of object services. The services listed below are in the process of being standardised; Requests for Proposal (RFP) have been issued, and in some cases submissions have been presented and mergers are in progress. The content of new object services yet to have RFPs issued is constantly being debated.

The following Object Services are the subject of the OS RFP 3 issued on September 26, 1994 [12].

(1) Object Security Services.

Security is concerned with:

- **Confidentiality:** Information is disclosed only to users authorised to access it.
- **Integrity:** Information is modified only by users who have the right to do so, and only in authorised ways.
- **Accountability:** Users are accountable for their security related actions.
- **Availability:** Authorised users cannot be denied access by the system.

The OMG security architecture is designed to support a variety of security policies to meet different needs.

(2) Object Time Service.

This service provides a mechanism for synchronising clocks in distributed systems. It has two requirements:

◇ It supports a synchronised notion of time that is compatible with *Coordinated Universal Time* (UTC).

◇ It must be a *Trusted Time Service*, that is, it supports the policies and operations of secure OMA compliant systems as described in the Object Security Services.

The following Object Services are the subject of the OS RFP 4 issued on September 26, 1994 [17].

(1) Object Licensing Service.

This service provides a framework for the specification and management of licensing policies and usage metering. It also provides support for usage charging.

(2) Object Properties Service.

Object properties are *typed, named values.* They are dynamically associated with any object. By using the interfaces defined by the Property Service, useful information can be associated with an object state.

The property service interfaces allow you to define properties by name, enumerate them, give them value, retrieve the value by name and delete them.

This service is now adopted.

(3) Object Query Service.

This service defines interfaces for operations on collections of objects that have a predicate-based, declarative specification and may result in collections of objects. It allows specification of queries in object derivatives of SQL, or in direct manipulation query languages.

The Object Query Service stresses *fine-grained* object access.

This service is now adopted.

The following Object Services are the subject of the OS RFP 5 issued in June, 1995 [25].

(1) Collection Object Service.

This service provides a uniform way for creation and manipulation of groups of objects such as sets, queues, stacks, lists, etc.

(2) Trading Service.

This supports matching of offered services to the needs of a client object.

(3) Startup Service.

This service is used to fire off *requests* when the ORB is started so that Object Services, Common Facilities and other ORB-based applications can do initial housekeeping.

The following Object Services are the subject of the OS RFP 6 issued in October, 1995 [26].

(1) Object Change Management Service.

Object Change Management Service provide for object interface, implementation and instance versioning and configuration management. They are basic object services that can be used to support the evolution and composition of complex system.

All Object Services are accessed through standard OMG IDL interfaces.

6 An Overview of OMA Common Facilities

Common Facilities is the third and final area of the OMA to be defined. They provide standardised interfaces to common application services. So far, the OMG Common Facility Task Force (CFTF) has issued RFP 1 and RFP 2. No specification has been adopted by the OMG for the Common Facilities at this stage. In this section, we give a brief technical overview of those common facilities requested.

The OMG has defined a Common Facilities Architecture which identifies the high-level service interfaces. There are two categories of Common Facilities: horizontal and vertical facility sets [13].

6.1 Horizontal Common Facilities

The *horizontal* set of Common Facilities includes functions covering many or most systems, regardless of application content. The OMA has considered almost sixty (60) candidate areas for standardisation of horizontal Common Facilities and they are divided into four groups [13]:

(1) User Interface.

This group covers all aspects of the user interface which makes an information system accessible to its users and responsive to their needs.

(2) Information Management.

This is a comprehensive group covering the modelling, storage, retrieval and interchange of information.

(3) System Management.

The facilities in this group make computer systems accountable and manageable for their intended mission. This is achieved by providing a common set of operating system interfaces to systems management tools, applications and managed resources.

(4) Task Management.

This group provides an infrastructure for the automation of work, including automation of both user processes and system processes.

6.2 Vertical Common Facilities

The *vertical* set of Common Facilities represent technology which supports various vertical market segments, such as financial systems or CAD systems.

All Common Facilities are accessed through standard OMG IDL interfaces.

6.3 Common Facilities: work in progress

The Common Facilities Task Force has so far issued three Requests for Proposals (CF RFP1, CF RFP2, and CF RFP3) for technologies in the horizontal common facilities.

1. Technologies requested in RFP 1 (issued September, 1995) include [14]:

   ◇ *Compound Presentation Facility.*

   This facility provides a framework for the sharing and subdividing a display window into multiple *parts*. These parts may be peers of each other, or may in turn be embedded into other parts. This facility roughly maps to the *display* portion of a compound document architecture. The requested technology is expected to be a *Compound Document Architecture* that is as independent as possible of any window systems [14].

   ◇ *Compound Interchange Facility.*

   This facility provides a framework for the storage and interchange of data objects, specifically to support facilities such as the Compound Presentation facility. This facility roughly maps to the persistent storage subsystem of a compound document architecture [14].

2. Technologies requested in RFP 2 include [18]:

   ◇ *Internationalisation Facility.*

     This facility enables users to use an information system or an application in their own language [18].

   ◇ *Time Operation Facility.*

     This facility provides a standard way of representing a *Date/Time Group* (DTG) as an instance of a time stamp, or a duration of time between two time stamps [18].

3. Technologies requested in RFP 3 include [19]:

   ◇ Data Interchange Facility.

     The Data Interchange Facility provides a framework for exchanging digital data in a uniform way. The facility supports interchange of data in both internalized and externalized format [19].

   ◇ Mobile Agent Facility.

     The Mobile Agent Facility provides for a creation and management of agents and agencies. Agents are objects that contains knowledge for performing tasks. Agencies are the place in which agents perform their tasks. Thus, they provide a processing environment and related facilities for agents [19].

7 CORBA vs. DCE

Both OMG's CORBA and OSF's Distributed Computing Environment (DCE) [9] represent significant advancements in distributed computing. With them, distributed computing has stepped out of research labs and the academic field, and has become an industry endeavour.

CORBA and DCE both attempt to tackle the development and deployment of distributed applications, but from different directions and perspectives. It is interesting to see their different technological approaches to some issues in distributed computing. Since a thorough comparison deserves a separate article, in this section we only provide an overview on how these two important platforms have approached distributed computing differently.

(1) Implementation vs. specification.

OSF provides a *reference implementation* of DCE, and most vendors' products use this code. Thus, the interoperability between different vendors' products is straightforward. The portability of applications across different vendors' products is also relatively straightforward.

OMG produces *specifications only*, defined by consensus-oriented committees, and there is no reference implementation of CORBA. Interoperability used to be impossible until CORBA2 specifically addressed this problem. Now portability is possible, but not trivial.

(2) Infrastructure vs. Interface to an infrastructure.

DCE provides an *infrastructure* including a comprehensive set of services to support distributed applications.

CORBA defines *interfaces* to an infrastructure.

(3) Procedural Model vs. Object-Oriented Model.

DCE is based on the *Remote Procedure Call (RPC)* paradigm. While objects and related concepts exist, DCE is not an object-oriented middleware.

CORBA is *object-oriented* technology, and there is an underlying *object model* defined for it. In the CORBA world, an object accesses the service of another object by sending *requests* to methods on that object. At a lower communication level, this sending of messages to methods appears to be a procedure invocation.

(4) Static vs. dynamic invocation interfaces.

DCE only supports static interfaces.

CORBA supports a variety of invocation mechanisms:

(a) the static invocation interfaces (Stubs or skeleton).
(b) the Dynamic Invocation Interface (DII) at the client side.
(c) the Dynamic Skeleton Interface (DSI) at the server side.

(5) Blocking vs. non-blocking Operation semantics.

DCE provides a variety of RPC semantics (*at-most-once, idempotent, broadcast*, and *maybe*), and an application always sees a blocking RPC.

Two styles of execution semantics are defined by CORBA's object model:

- *at-most-once* (default)
  If an operation request returns successfully, it was performed exactly once; if it returns an exception indication, it was performed at-most-once.
- *best-effort (oneway).*
  This is a request-only operation, i.e. it does not return any result and the requester never synchronises with the completion (if any) of the request.

It supports more interaction models:

(a) with the static interface, only a blocking (synchronous) RPC is supported;
(b) with the DII, an application can realize a *deferred synchronous RPC.*
(c) by specifying *oneway*, asynchronous interactions between the caller and the callee can be achieved.

(6) Threading vs. concurrency.

Threading is an integral part of DCE, and is handy for application use.

In CORBA, at the CORBA object level, concurrency should be managed through the *Concurrency Object Service* which may not always be shipped with the ORB implementations by vendors. Some vendors provide other concurrency mechanisms at the implementation level.

(7) DCE IDL vs. CORBA IDL.

DCE IDL and CORBA IDL are very similar.

DCE IDL is derived from C and the syntax of the CORBA IDL is derived from C++. However, CORBA IDL has the following features that are not in DCE IDL:

- The *any* type, used to denote any valid CORBA IDL type.
- Exceptions.

  CORBA IDL has *syntax* provision for exception handling. There is no such provision in DCE IDL, although DCE allows you to handle errors using macros such as CATCHANY, CATCHALL, or using the exception attributes such as fault_status and comm_status as specified in an Attribute Configuration File (ACF).
- Inheritance.

  Since DCE is not meant to be an object-oriented platform, there is no syntax provision for interface inheritance in DCE IDL.

DCE IDL also has features not included in CORBA IDL:

- Pointers.
- Pipes.
- Context handles.

(8) Interface UUID vs. object reference.

Each interface defined in DCE IDL is assigned a Universal Unique IDentifier (UUID) and version number. The UUID is guaranteed unique across space and time. The interface subtyping relationship is solely defined via version number.

In CORBA, an object supporting an interface is assigned an object reference by ORB, then the interface is registered with the *interface repository*. It is assigned a *repository ID* which is not necessarily universally unique. CORBA 2, however, specifies a means of providing pseudo-unique tags to identify interfaces.

(9) Accessing basic services.

DCE does not define standard interfaces for accessing many basic services such as transaction services.

In the OMA, Object Services are fundamental components in its architecture. All object services adopted by the OMG have standard IDL interfaces for access to these services. In general, these services provide a higher-level of abstraction than DCE does and cover a broader technological area [24].

Since its release, DCE has gained an increasingly wide user base, whereas the user response to OMG CORBA is relatively slow. In the Closing Remarks at the end of this report, we point out some of the reasons for this situation. DCE and CORBA were developed not for competition, but to meet different requirements. However, as a technology, OMG CORBA, based on the object-oriented paradigm, seems more suited to application integration while DCE is likely to become more a part of operating systems infrastructure.

8 Closing Remarks

Since its founding in 1989, the industry consortium Object Management Group (OMG) has made great strides in defining a distributed object computing platform for application integration in

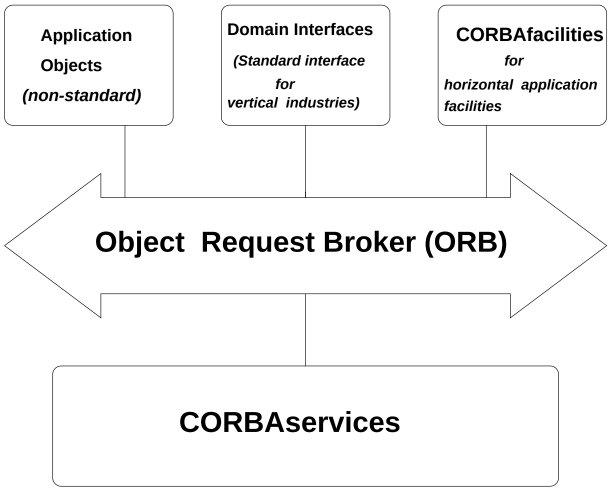

Figure 8: The revised Object Management Architecture.

widely distributed, heterogeneous environments. CORBA is maturing very rapidly, and the specifications have received broad industry support. CORBA-compliant implementations are available for nearly all major platforms. However, OMG technology has had only slow acceptance, not without reason. OMG CORBA technology is object-oriented technology for distributed computing, and many application developers and system integrators are not yet technically capable of applying and deploying it in real environments. Furthermore, OMG has resolved the interoperability issue only at the end of 1994 and vendors are only just beginning to release interoperable ORBs in early 1996. Another major concern is security. This has not been sufficiently addressed until recently. It can be expected that an Object Security Service Specification will be adopted by the OMG during 1996. System management of distributed objects has also received little attention. These three issues have been major obstacles to the acceptance of CORBA for serious and mission-critical applications development, particularly for large institutions which have taken a wait-and-see attitude.

OMG is now the largest computer industry consortium and has established a large set of API specifications to maintain, enhance and supplement. To manage this growth and to ensure consistency among this large set of specifications, in November 1995 the OMG's board of directors restructured the organisation resulting in the following changes:

- **A New Architecture Board** was established to ensure the coherence and consistency of the OMG's growing set of specifications and to support a future compliance test suite.
- **New structure with more user rights.** A new Domain Technology Committee (DTC) was established to allow for end-user organisations and independent software vendors (ISVs) to contribute to OMG specifications. The distinction between end-user and "corporate" members was abolished. The old Technical Committee (TC) works under the new name of *Platform Technology Committee (PTC).*
- **Revised OMA.** The revised OMA expands the old OMA, which now has the following components: *ORB, CORBAservices* (replacing the names Objects Services and COSS), *CORBAfacilities* (previously Common Facilities), and a new *Domain Interfaces* (taken from ap-

plication objects and vertical common facilities), and *Application Interfaces* which will never be standardised. The "Domain Interfaces" will be defined by DTC (Figure 8.)

Distributed object computing is a promising technology, and is gaining increasing mindshare. It could become the dominant computing paradigm of the near- to mid-term future. It will provide the infrastructure for an increasing number of future information systems and will provide improved mechanisms for accommodating legacy systems.

OMG currently represents the best framework for heterogeneous distributed object-oriented computing, and will evolve and stabilise. CORBA technology is quietly being integrated into other products, such as compound document builders, object-oriented databases and graphical user interface builders. ORB products are poised to infiltrate the market beyond the current "early-adopter" niche, and could gain wide acceptance once the ORBs and their development support environments reach the necessary maturity.

9 Acknowledgements

This work has been funded in part by the Cooperative Research Centres Program through the department of the Prime Minister and Cabinet of the Commonwealth Government of Australia.

We would like to thank our colleagues at the DSTC, in particular, Kerry Raymond and Melfyn Lloyd for their insight on the formation of this report, and David Jackson for his careful reading and many instructive comments.

10 References

In addition to the cited references, some suggested further reading materials are also included here.

[1] Robin Bloor. *The Object Management Guide.* Butler Bloor Ltd., Challenge House, Sherwood Drive, Bletchley, Milton Keynes, MK3 6DP United Kingdom, April 1992.

[2] Grady Booch. *Object-Oriented Design with Applications.* The Benjamin Cummings Publishing Company Inc., Redwood City, CA., 1991.

[3] Michael L. Brodie and Michael Stonebraker. DARWIN: On the Incremental Migration of Legacy Information Systems. Technical Report TR-0222-10-92-165, GTE Laboratories, Inc., Waltham, MA 02254, March 1993.

[4] Michael L. Brodie and Michael Stonebraker. *Migrating Legacy Systems: Gateways, Interfaces, and the Incremental Approach.* Morgan Kaufmann Publishers, 340 Pine, San Francisco, CA 94104, 1995. ISBN 1-55860-330-1.

[5] R. G. G. Cattell. *Object Data Management, Object-Oriented and Extended Relational Database Systems.* Addison-Wesley Publishing, 1991.

[6] Peter Coad and Edward Yourdan. *Object-Oriented Analysis.* Yourdan Press Computing Series. Prentice Hall Publishers, 1990.

[7] Microsoft Corporation. *The Component Object Model (Technical Overview).* Microsoft Corporation, One Microsoft Way, Redmond, WA.

[8] Richard M. Soley (ed.) and Christopher M. Stone. *The Object Management Architecture Guide.* John Wiley & Sons, Inc., third edition, June 13 1995.

[9] Open Software Foundation. *OSF DCE Application Development Guide.* Open Software Foundation, 11 Cambridge Center, Cambridge, MA., revision 1.0, update 1.0.2 edition, 1994.

[10] Object Management Group. *Object Services RFP1.* OMG TC Document 92-8-6, August 28 1992.

[11] Object Management Group. *Object Services RFP2.* OMG TC Document 93-6-1, June 1 1993.

[12] Object Management Group. *Object Services RFP3.* OMG TC Document 94-7-1, July 5 1993.

[13] Object Management Group. *Common Facilities Architecture.* Object Management Group (OMG), Framingham, MA., revision 2.0 edition, 26 September 1994. OMG TC Document 94-9-40.

[14] Object Management Group. *Common Facilities RFP1.* OMG TC Document 94-9-13, September 14 1994.

[15] Object Management Group. *Common Object Services Specifications*, volume 1. John Wiley & Sons, Inc, 1994.

[16] Object Management Group. *Common Object Services Specifications*, volume II. John Wiley & Sons, Inc, 1994.

[17] Object Management Group. *Object Services RFP4.* OMG TC Document 94-4-18, May 11 1994.

[18] Object Management Group. *Common Facilities RFP2.* OMG TC Document 95-1-31, February 2 1995.

[19] Object Management Group. *Common Facilities RFP3.* OMG TC Document 95-9-40, October 17 1995.

[20] Object Management Group. *CORBAservices: Common Object Service Specification.* Object Management Group (OMG), Framingham, MA, March 31 1995. (Revised Edition).

[21] Object Management Group. *IDL – Ada Language Mapping Specification (Version 1.0).* Object Management Group (OMG), Framingham, MA, 22 May 1995. OMG TC Document 95-5-16.

[22] Object Management Group. *Interface Repository.* Object Management Group (OMG), Framingham, MA, version 1.0.2 edition, 3 April 1995. OMG TC Document 94-11-7.

[23] Object Management Group. *Object Model.* Object Management Group (OMG), Framingham, MA., draft 0.3 edition, 12 January 1995. OMG TC Document 95-1-13.

[24] Object Management Group. *Object Services Architecture* . Object Management Group (OMG), Framingham, MA., 12 January 1995. OMG TC Document 95-1-47.

[25] Object Management Group. *Object Services RFP5.* OMG TC Document 95-6-18, June 5 1995.

[26] Object Management Group. *Object Services RFP6.* OMG TC Document 95-10-13, October 30 1995.

[27] Object Management Group. *The Common Object Request Broker: Architecture and Specification (Revision 2.0).* Object Management Group (OMG), Framingham, MA., July 1995.

[28] Object Management Group. *Universal Networked Objects.* Object Management Group (OMG), Framingham, MA, March 1995.

[29] Object Management Group. *The Common Object Request Broker: Architecture and Specification (CORBA)* . Object Management Group (OMG), Framingham, MA., Revision 1.2. Draft 29 December 1993.

[30] Paul Harmon and David A. Taylor. *Objects in Action: Commercial Applications of Object-Oriented Technologies.* Addison-Wesley Publishing, 1993.

[31] Andrew T. F. Hutt. *Object Analysis and Design: Comparison of Methods* . J Wiley, 1994.

[32] IBM. *SOMobjects Developer Toolkit: Users Guide.* International Business Machines Corporation, June 1993. (Second Edition).

[33] ITU/ISO. Basic Reference Model of Open Distributed Processing — Part 3: Prescriptive Model. International Standard 10746-3, ITU-T Recommendation X.903, 1995.

[34] Judith Jeffcoate, Keith Hales, and Valerie Downes. *Object-Oriented Systems: the Commercial Benefits.* OVUM ltd, RathboneStreet, London. Next publication March 1992 with OMG.

[35] Won Kim. *Introduction to Object-Oriented Databases.* Computer Series. MIT Press, Cambridge,MA., 1990.

[36] Won Kim and Frederick Lochovsky. *Object-Oriented Concepts, Databases and Applications.* ACM press Frontier Series. Addison-Wesley Publishing, 1989.

[37] Mary E. S. Loomis. *Object Databases: The Essentials.* Addison-Wesley Publishing, 1995.

[38] James Martin and James Odell. *Object-Oriented Analysis and Design.* Prentice-Hall, Englewood Cliffs, NJ USA., 1992.

[39] Bertrand Meyer. *Object-Oriented Software Construction.* International Series in Computer Science. Prentice Hall Publishing, 1988.

[40] Thomas J. Mowbray and Ron Zahavi. *The ESSENTIAL CORBA: System Integration Using Distributed Objects.* John Wiley & Sons and the Object Management Group, July 1995.

[41] Robert Orfali, Dan Harkey, and Jeri Edwards. *Essential Client/Server Survival Guide.* Van Nostrand Reinhold, 1994.

[42] Robert Orfali, Dan Harkey, and Jeri Edwards. *The Essential Distributed Objects Survival Guide.* John Wiley & Sons, Inc., 1996. ISBN 0-471-12993-3.

[43] Randy Otte, Paul Patrick, and Mark Roy. *Understanding CORBA.* Prentice-Hall, 1995. ISBN 0-13-459884-9.

[44] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen. *Object-Oriented Modeling and Design.* Prentice-Hall, Englewood Cliffs, N.J., 1991.

[45] Sally Shlaer. *Object-Oriented Systems Analysis: Modeling the World in Data.* Englewood Cliffs, NJ: Yourdon Press, 1988.

[46] Alan Snyder. The Essence of Objects: Concepts and Terms. *IEEE Software*, 10:31–42, January 1993.

[47] David Taylor. *Managers Guide to Object-Oriented Technology.* Addison-Wesley Publishing Company, Reading MA, USA., 1990.

[48] Gomer Thomas and Rob van der Linden. Remote Database Queries in Open Distributed Systems. Technical Report APM 1138, APM Ltd, Cambridge, U.K., June 1994.

[49] Ann Winblad, Samuel D. Edwards, and David R. King. *Object-Oriented Software.* Addison-Wesley Publishing Company, 1990.

[50] Rebecca Wirfs-Brock, Brian Wilkerson, and Lauren Wiener. *Designing Object-Oriented Software.* Prentice Hall, Englewood Cliffs NJ, USA., 1990.