Zeitschrift: Comtec : Informations- und Telekommunikationstechnologie = information and telecommunication technology

Herausgeber: Swisscom

Band: 75 (1997)

Heft: 7

Artikel: A framework for transparent communication

Autor: Zweiacker, Marc

DOI: https://doi.org/10.5169/seals-876951

A BRIEF SURVEY OF CORBA

A FRAMEWORK FOR TRANSPARENT COMMUNICATION

In 1989 the most powerful standardization body for distributed computing has been founded, the Object Management Group (OMG). More than 700 companies worldwide are a member of the OMG, and this figure is still rising continuously. 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 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), which specifies a framework for transparent communication between application objects.

In 1989 the Object Management Group (OMG) was founded by eight companies as a nonprofit organization with the aim '. . . to adopt interface

MARC ZWEIACKER, BERN

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' [4]. Today, more than 700 companies are a member of the OMG, with almost every significant computer vendor represented.

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 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), which specifies a framework for transparent communication between application objects. CORBA is the first specification adopted by the OMG.

Figure 1 is a widely used representation of the OMA, comprising normal application objects, object services, common facilities, and the ORB which enables object interaction. The OMG Object Services Specifications define a set of objects which perform fundamental functions such as naming, life cycle services, and transactions. Common facilities have an application focus and are mostly used by distributed application developers. Essentially, services and facilities are components which ease the creation of distributed applications; however, they are not conceptually different from ordinary, normal application objects. They work according to the same architecture and use the same communication mechanisms.

Interfaces and language mapping

This section introduces object interfaces and how they relate to services. The decoupling of a service specification from the actual implementation is realized through the language mappings. Language mappings are rules that transform the abstract service description into the inherent constructs of a programming language.

Interfaces and IDL

According to the object model, objects communicate through interfaces with one another. The interface is a conceptual element of the architecture that an object must support in order to

- enable communication with other objects
- be compliant with the OMA

The OMA promotes the client-server paradigm in that objects take different roles during a communication session. The functionality that an object offers to its clients (the service) is expressed using the OMG's Interface Definition Language (IDL), a metalanguage specifying data structures and operations that clients can invoke on an interface. Figure 2 depicts the client-server relation and shows a sample interface description. The server is a simple banking service that offers clients the possibility to create and delete accounts and to deposit or withdraw money from these accounts. The account interface defines the balance attribute, which is used to specify the amount of money to deposit or withdraw. makeDeposit and makeWithdrawal are operation names, the void keyword specifying that no return value is associated with these operations. The bank interface lets users create new accounts through the newAccount operation. This operation returns an account interface on which financial transactions can be invoked. It takes a string as the only parameter to specify the name of the account holder. Finally, deleteAccount destroys the account interface passed as the input parameter.

We are not going into the details of IDL in this article; the interested reader is directed to [3] for more information. However, there is an important thing to note: An IDL specification does not reveal the implementation details of a service, i.e., the banking service could be realized by competing banking institutes, each of which has its own distinct computer center running different hardware and software. As long as the IDL specification for the banking service is respected, clients can use either server. The implementation is hidden behind the interface. In all standardization efforts, the OMG aims at specifying interfaces, but does not dictate the implementation. This is one of the major success factors for CORBA: It allows service specification without exposing the implementation, thus creating a world of agreed services that leaves enough space for competition on the market (through the better quality of a service, for instance).

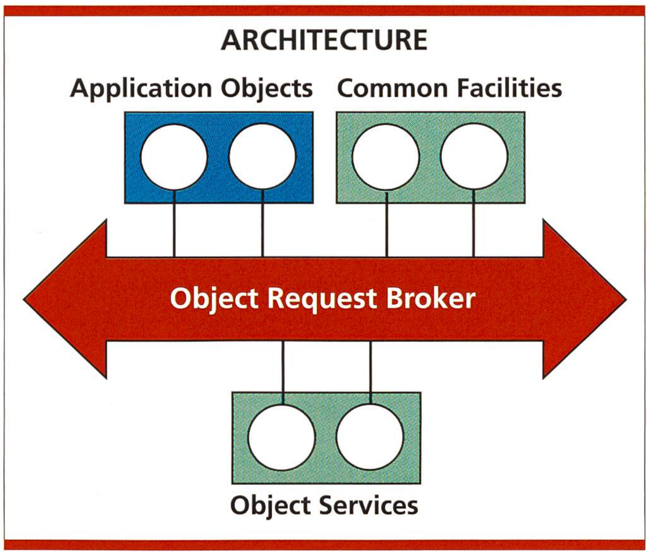


Fig. 1. Object Management Architecture.

Language mappings

Using IDL, the service is described independently of its implementation. To make such independence possible, IDL specifications are cross-compiled into the programming language being used, transferring the intended semantics of the IDL specifications into the language domain. For this particular reason, a number of language mappings (semantic correspondences between IDL and the target programming language) have been adopted by the OMG for widely used languages, including C, C++, SmallTalk, COBOL, and Ada. With these mappings programmers can use a mixture of languages to implement application objects without limiting interoperability, as the mappings ensure semantically consistent information crossing the interfaces between objects. It is therefore not uncommon, that an object written in SmallTalk communicates with another object written in C++. The differences in the implementation strategy and language is totally transparent to the objects, i.e., the SmallTalk object would in general not be able to tell that it communicates with a C++ object. Such implementation transparency is achieved through the ORB, which is in charge of the communication between the objects. Simply stated, the ORB can be regarded as a communication relay between objects. It silently adapts to the operating system environment and programming language of an object without exposing the details of this adaptation to the outside world (Fig. 3). This is another major strength of CORBA: It allows interoperability of objects across heterogeneous operating systems and programming languages. Quickly, these are the important points for your CORBA comprehension:

- CORBA enables transparent interaction for distributed objects across heterogeneous operating systems.
- Objects can be written in any language, provided that there is a mapping adopted for it by the OMG.
- Objects communicate with one another through the ORB, no matter what language has been used to implement them.

The interested reader is encouraged to look at [5], which is a more detailed, excellent introduction to OMA and CORBA.

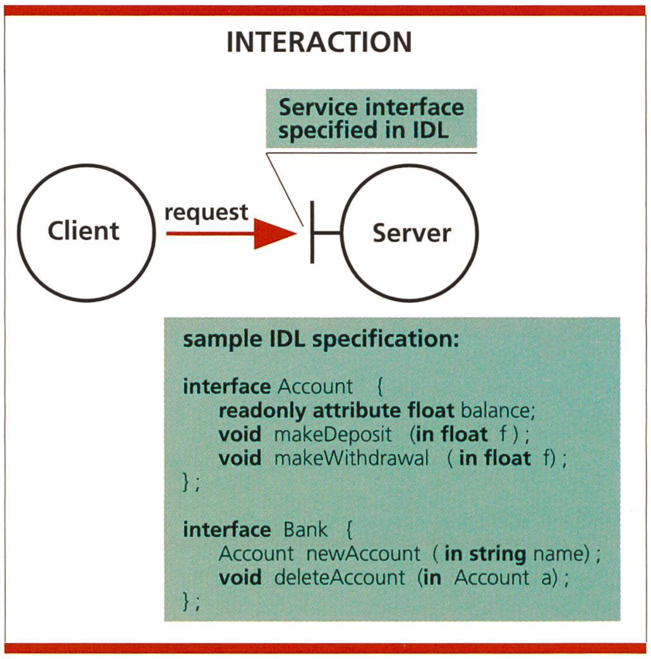
Services

This section is devoted to the way services are defined in CORBA. The Common Object Services Specifications are given special attention.

Service classification

The OMA promotes the client-server paradigm, meaning that the interaction between any two objects is of an asymmetric kind as far as communication behaviour is concerned:

- A server is an application object that waits for incoming connection requests from other objects. After a connection has been established, the server object would perform some action requested by the calling object.
- A client is an object that calls a server.



```
interface Account  {
      readonly attribute float balance;
      void  makeDeposit   (in float  f ) ;
      void  makeWithdrawal   ( in float  f) ;
} ;

interface Bank  {
      Account  newAccount  ( in string name) ;
      void  deleteAccount  (in  Account  a) ;
} ;
```

Fig. 2. Client-server interaction.

Note that in general server and client are not labels that you attach to an object. They are rather roles that specify the behaviour of the object for a particular communication session. An object can take both roles: It acts as a server when it waits for incoming requests, and it acts as a client when it initiates a communication session. Server and client objects are not distinguishable by the Core Object Model; they both obey the architectural rules. CORBA differentiates between the following classes of services:

- *Object Services,* a collection of services that ease the task of programming distributed applications. For example, the Life Cycle Service defines conventions for creating, deleting, copying, and moving objects; however, it does not dictate how the objects are implemented in the application. The Object Services are defined in the CORBAservices suite of specifications.
- *Common Facilities,* a set of lower level object services that many applications may share but which are not as fundamental as Object Services. For instance, the Information Management Common Facilities comprise specifications and further references for Information Modelling, Storage and Retrieval, Interchange, and Data Encoding & Representation. Information about Common Facilities is contained in the CORBAfacilities documentation.
- *Application Services* are the services built for a particular purpose, i.e., they are the services that end users actually want. They need to be engineered and programmed, possibly relying – to a certain extent – on Object Services or Common Facilities or both.

CORBAservices

The most advanced set of adopted service standards is the Common Object Service Specification (COSS), commonly known as CORBAservices. The following is an overview of adopted CORBAservices:

- The *Naming Service* provides the ability to bind a name to an object relative to a naming context. The naming context contains a set of name bindings in which each name is unique. The advantage of having a Naming Service lies in easier addressing of objects.
- The *Event Service* provides basic capabilities to support asynchronous events, notification, and – through appropriate event channel implementations – reliable event delivery.
- The *Life Cycle Service* defines conventions for creating, deleting, moving, and copying objects. Because CORBA-based environments support distributed objects, clients can perform life cycle operations on objects in different locations.
- The *Persistent Object Service* provides the capabilities to retain and manage the persistent state of objects.
- The *Transaction Service* supports multiple transaction models, including the flat (mandatory in the specification) and nested (optional) models.

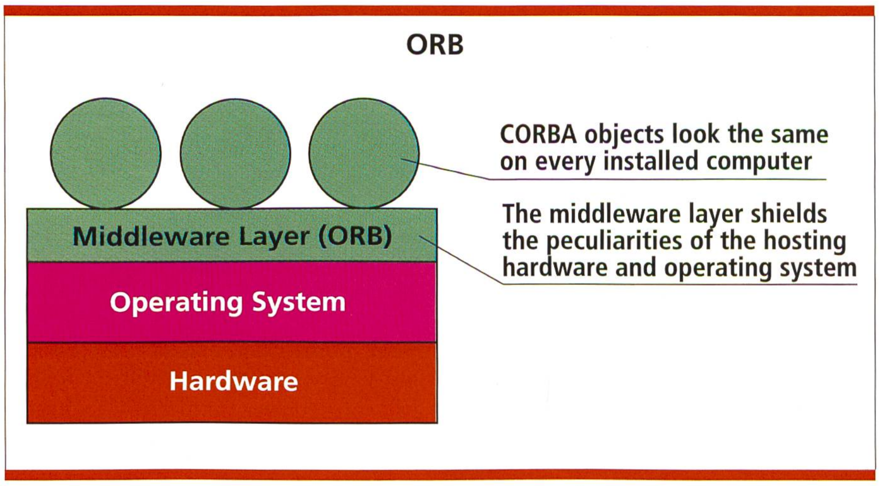

ORB
CORBA objects look the same on every installed computer
The middleware layer shields the peculiarities of the hosting hardware and operating system
Middleware Layer (ORB)
Operating System
Hardware

Fig. 3. The ORB hides heterogeneity of both hardware and operating systems.

- 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.
- The *Relationship Service* allows entities and relationships to be explicitly represented. The service defines two new kinds of objects: relationships and roles. Using these objects, type and cardinality constraints can be expressed and checked: Exceptions are raised, when the constraints are violated.
- The *Externalization Service* defines protocols and conventions for externalizing and internalizing objects. Externalizing means to record the object state in a stream of data (in memory, on a disk file, across the network, etc.) and then to internalized it into a new object in the same or a different process.
- The *Query Service* allows users and objects to invoke queries on a collection of other objects. The queries contain declarative statements with predicates and include the ability to specify values of attributes to invoke arbitrary operations and other Object Services.
- The *Licensing Service* provides mechanisms for producers to control the use of their intellectual property.
- The *Property Service* provides the ability to dynamically associate named values with objects outside the static IDL-type system.
- The *Time Service* enables users to obtain current time together with an error estimate associated with it. It can be used further to ascertain the order in which 'events' occurred, to generate time-based events, and to compute the interval between two events.
- The *Security Service* comprises specifications for *identification and authentication* of principals (human users and objects) to verify they are who they claim to be, for *authorization and access* control to decide whether a principal can access an object, for *auditing* to make users accountable for their security-related actions, for *secure communication* between objects (which is often over insecure lower layer communications), and for *administration* of security information.

Conclusions

The CORBA standards provide the flexibility to construct open distributed applications in a heterogeneous environment, comprising different hardware and software, possibly originating from different vendors. Services are specified in IDL to make them independent of the implementation. The language mapping maintains semantic consistency between these interface specifications and the programming language used. Besides defining an architectural model, the standards define a set of commonly used services, CORBAfacilities and CORBAservices, that ease the creation of distributed applications. 2

References

1 ISO/IEC Draft International Standard 10746-1 / ITU-T Recommendation X.901, Reference Model of Open Distributed Processing, Part 1: Overview, 1995.
2 ISO/IEC International Standard 10746-3 / ITU-T Recommendation X.903, Reference Model of Open Distributed Processing, Part 3: Architecture, 1995.
3 The Common Object Request Broker: Architecture and Specification, The Object Management Group, Revision 2.0, July 1995.
4 Object Management Architecture Guide, The Object Management Group, Revision 2.0, edited by R. M. Soley, OMG document number 92-11-01, 1992.
5 Yang Z., Duddy K., CORBA: A Platform for Distributed Object Computing. ACM Operating Systems Review, Vol. 30, No. 2, pp 4–31, April 1996.

ZUSAMMENFASSUNG

CORBA

Im Jahre 1989 wurde die wohl mächtigste Standardisierungsorganisation mit dem Namen Object Management Group (OMG) ins Leben gerufen. Weltweit gehören seitdem über 700 Unternehmen der OMG an, und die Anzahl der Mitglieder ist weiterhin zunehmend. Die OMG hat ein konzeptuelles Modell, Core Object Model, sowie eine dazugehörige Architektur, Object Management Architecture (OMA), entwickelt, womit verteilte Applikationen gebaut werden können. Das Kernstück der Architektur bildet der Object Request Broker (ORB), ein universeller Kommunikationsbus für Applikationsobjekte. Die Standards, die für den ORB definiert worden sind, bezeichnet man als Common Object Request Broker Architecture oder CORBA.

Marc Zweiacker ist nach Abschluss der Studienzeit an der HTL Burgdorf (Elektrotechnik) und der ETH Zürich (Informatik) im Frühjahr 1991 der Gruppe «Netzwerke und Kommunikation» der Direktion Forschung und Entwicklung beigetreten. Sein Spezialgebiet sind die verteilten Systeme in der Normierung wie auch in der praktischen Handhabung sowie dem Management. 1993 und 1994 hat er aktiv an der Definition des ITU-T Standards X.900 «Reference Model for Open Distributed Processing» mitgewirkt. Seit 1996 befasst sich Marc Zweiacker vornehmlich mit Fragen der Fehlertoleranz von verteilten Systemen und speziell von auf CORBA basierenden Applikationen.