YU-INFO-2000

Advances in Infrastructure for E-Business on the Internet

Kopaonik, March 27 – 31, 2000

Internet and Object-oriented Design Methods

Tutorial

Andrea Domenici

SSSUP S. Anna, Pisa

andrea@sssup.it

http://www.ing.unipi.it/~d8651

Contents

1 Introduction

This talk introduces the basic concepts of the *Common Object Request Broker Architecture*. These concepts are tools that designers can use to meet the requirements of large scale distributed applications, such as those that are made possible by the existence of a world-scale Internet.

Slide 1

Introduction

This talk introduces the basic concepts of the *Common Object Request Broker Architecture*.

- Computation model.
- Architecture.
- Interface Description Language.
- Dynamic interfaces and implementations.
- Interoperability

The structure of a *very* simple application is described through an example taken from the book *MICO is CORBA*, by A. Puder and K. Römer, (dpunkt.verlag, Heidelberg, 1999).

2 Internet and distributed systems

Like Proteus in the ancient myth, the Internet can show innumerable shapes and faces to its users. Designers must cope with ever evolving needs and demands, and deliver open-ended, adjustable and extensible systems.

Several services or activities are supported by clusters of computers connected by local networks. Each service or activity needs to interact with the others. Each service or activity may belong to a different organization. New suppliers and consumers of services show up on the Net in time.

3 A large scale object-oriented approach

In this section we sketch the evolution of the design/programming approach to distributed systems in three major phases.

Marshaling and *demarshaling* refer to the encoding and decoding, respectively, of application data into and from a format suitable to transmission over a communication channel, as specified by a transport protocol (e.g., TCP/IP).

Slide 2

A Large Scale OO Approach

The Common Object Request Broker Architecture is an OO approach for large scale distributed systems, where emphasis is on *application-centered* object interaction. The CORBA architecture provides:

- An object-oriented computational model (*Object model*).
- A common interface language (*OMG IDL*).
- A networking and distribution infrastructure (*Object Request Broker*).
- Standard component interface libraries (*CORBA Services and Facilities*).

The CORBA specifications are produced and maintained by the *Object Management Group* (OMG) (`www.omg.org`).

4 The Common Object Request Broker Architecture

4.1 Computation model

Slide 3

The CORBA Model

- Application developers design *application* (or *business*) *objects*.
- Objects have a *server role*, as they offer services, defined by object *interfaces*. Objects often have a *client role* too, as they use other objects, but usually the term "object" refers to the server role.
- A client application accesses an object through an *object reference*. The operations defined by the object interface are called through the reference, and they are independent of the object's physical location and the communication infrastructure.

Slide 4

Execution Semantics and Synchronization

- At-most-once: 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: a best-effort operation is a request-only operation (i.e., it cannot return any results and the requester never synchronizes with the completion, if any, of the request).
- At-most-once operations are *synchronous* by default, but may also be invoked in *deferred-synchronous mode*, where the caller continues execution while the target object is processing of the request, and can later poll the target for the results.
- The deferred-synchronous mode is possible for requests issued through the *Dynamic Invocation Interface*.

At-most-once operations are synchronous by default, i.e., the caller is blocked until the call returns.

Best-effort operations *may* be (and in most implementations are) executed asynchronously, i.e., the caller continues execution after sending a request. Best-effort operations can be used when non-reliable communication is appropriate, but implementations are free to deliver best-effort operation requests on a reliable protocol.

Best-effort semantics is specified by the `oneway` keyword in IDL interfaces (see Slide 13).

In *deferred-synchronous mode* the caller continues execution while the target object is processing of the request, and can later poll the target for the results. This mode of execution is possible for requests issued through the *Dynamic Invocation Interface* (see Slide 25).

4.2 Core architecture

Slide 5

The CORBA Architecture (1)

To support the computation model, on the client side the CORBA architecture must:

- provide clients with (server) object references;
- intercept *requests* (operation invocations) on object references;
- translate them into network messages and send them to the "real" (usually remote) objects.
- receive results from objects, re-format them and deliver them to the client application.

Slide 6

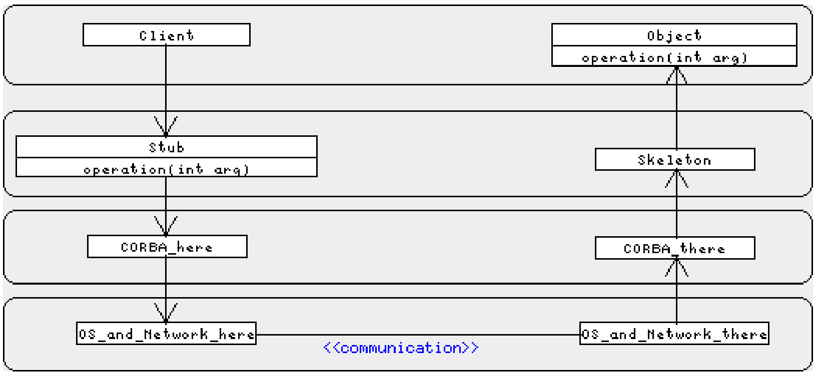
The CORBA Architecture (2)

On the server side:

- locate target objects (or create, or activate them if necessary);
- translate network messages back to operation invocations on objects;
- obtain results from objects;
- translate them into network messages and send them to the client application.

Slide 7

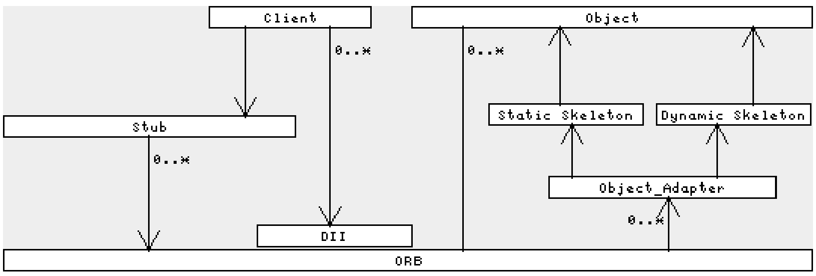
CORBA Architecture (3)



Classes `Client` and `Object` are designed by application developers (not necessarily by the same developers). Classes `Stub` and `Skeleton` are application-specific classes, generated by CORBA utilities, that connect the application code with the CORBA run time support, represented by `CORBA_here` and `CORBA_there`. The CORBA run time support uses the host operating system and networking facilities.

Slide 8

CORBA Architecture (4)



In this slide the OS and network layer is omitted, and the structure of the CORBA architecture is more detailed.

- The *Object Request Broker* (ORB) is the “software bus” upon which the CORBA architecture rests. The ORB provides the basic representation of objects and communication of requests.
- Clients and objects may access the ORB directly, by calling operations defined in the ORB interface (e.g., a client may call the ORB to obtain object references). All ORB’s have the same interface.

- The *Dynamic Invocation Interface* (DII) is the part of the ORB that allows clients to issue operation requests on object whose type is unknown at compile time (see Slide 25).
- The *Object Adapter* connects object skeletons to the ORB (see Slide 10).
- The *Static Skeleton* is the same as the *Skeleton* in Slide 7.
- The *Dynamic Skeleton* is a skeleton that the ORB may call even if its object's interface is unknown at compile time (see Slide 26.

Slide 9

Objects, Servants, and Servers

- An object is really an abstract entity: a set of services described by an interface.
- An object is implemented by a *servant*, a piece of code whose execution provides the object's services. Often the term 'object' is used when 'servant' is meant.
- A *server* is a computer program that executes one or more servants. We think of objects as "residing" in servers.

Slide 10

Object Adapter

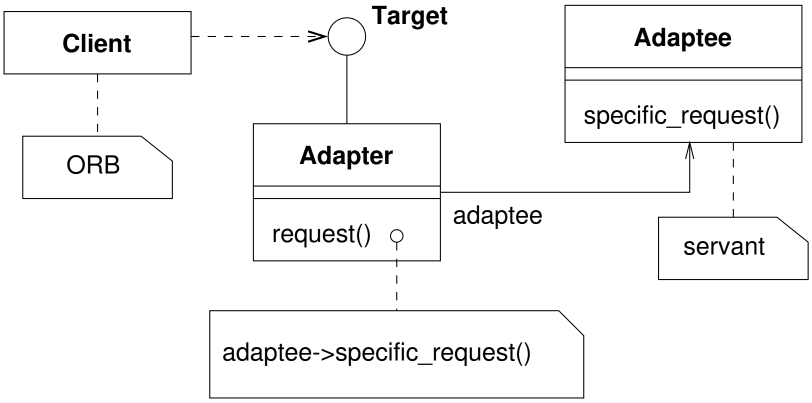
Object adapters are responsible for the following functions (from CORBA V2.2 Spec.):

- Generation and interpretation of object references.
- Method invocation.
- Security of interactions.
- Object and implementation activation and deactivation.
- Mapping object references to the corresponding object implementations.
- Registration of implementations.

The purpose of the Object Adapter in the architecture is to provide for maximum flexibility in the design and management of object implementation, while keeping the ORB interface small and its implementation efficient. The needs of different implementations can be accommodated for by different adapters.

Slide 11

The Adapter pattern



Slide 12

The POA

Early CORBA specifications (before CORBA 2.2) defined a *Basic Object Adapter*. Current specifications define a more flexible and powerful adapter, the *Portable Object Adapter*:

- Enables portable (across different ORB's) object implementations.
- Supports either persistent or transient objects.
- Enables multiple object identities per servant.
- Enables multiple adapters per server.

4.3 The Interface Description Language

Interface Description Language (1)

Slide 13

Object interfaces are specified in the *Interface Description Language* (IDL). This language allows application components to communicate with each other and with the CORBA environment.

- IDL has no procedural statements (total implementation hiding!).
- IDL is (obviously) an OO language, with multiple inheritance.
- IDL is statically typed, but has a universal type `any`.
- Specification of exceptions (system- and user-defined) is supported.
- Language mappings provide a safe dynamic downcasting operation.

Interface Description Language (2)

Slide 14

- All CORBA components (ORB, CORBAservices, etc.) appear as objects defined by IDL interfaces. These objects are available as library modules.
- Application designers define objects in IDL.
- An *IDL compiler* generates, for each IDL interface, stub and skeleton source code, to be compiled and linked with the client and, respectively, the server code.
- It generates also a translation of the interface in the required programming language.

Slide 15

Example: banking

```
// account.idl

interface Account {
    void deposit( in unsigned long amount );
    void withdraw( in unsigned long amount );
    long balance();
};

interface Bank {
    Account create();
};
```

Slide 16

Banking: C++ mapping

```
/*  MICO --- a free CORBA implementation
 *  Copyright (C) 1997-98 Kay Roemer & Arno Puder */

// account.h

class Account : virtual public CORBA::Object {
public:
    virtual void deposit( CORBA::ULong amount ) = 0;
    virtual void withdraw( CORBA::ULong amount ) = 0;
    virtual CORBA::Long balance() = 0;
    // ...
};
```

Slide 17

Banking: client-side mapping

```
// Stub for interface Account
class Account_stub : virtual public Account {
public:
    void deposit( CORBA::ULong amount );
    void withdraw( CORBA::ULong amount );
    CORBA::Long balance();
    // ...
};
```

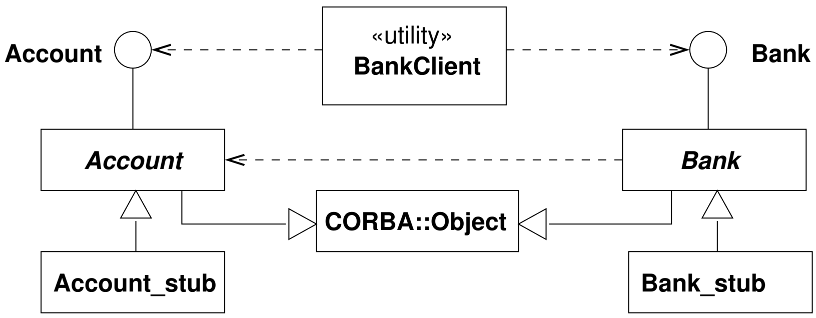
Slide 18

Banking client (C++)

```
main (int argc, char *argv[])
{ CORBA::ORB_var orb =
       CORBA::ORB_init(argc, argv, "mico-local-orb");
  // Connect to the Bank
  CORBA::Object_var obj = orb->bind("IDL:Bank:1.0");
  Bank_var bank = Bank::_narrow(obj);
  // Open an account
  Account_var account = bank->create();
  // Deposit and withdraw some money
  account->deposit(700);
  account->withdraw(450);
  printf("Balance is %ld.\n", account->balance ());
}
```

Slide 19

Banking client (UML)



Slide 20

Banking: server-side mapping (1)

```
class Account_impl : virtual public POA_Account {
public:
    void deposit (CORBA::ULong);
    void withdraw (CORBA::ULong);
    CORBA::Long balance ();
    // ...
};
```

Slide 21

Banking: server-side mapping (2)

```
class Bank_impl : virtual public POA_Bank {
    PortableServer::POA_var mypoa;
public:
    Bank_impl(PortableServer::POA_ptr);
    Account_ptr create();
};
Account_ptr Bank_impl::create()
{
    CORBA::Object_var obj =
         mypoa->create_reference("IDL:Account:1.0");
    Account_ptr aref = Account::_narrow(obj);
    return aref;
}
```

Slide 22

Banking server (C++) (1)

```
main (int argc, char *argv[])
{
  CORBA::ORB_var orb =
       CORBA::ORB_init(argc, argv, "mico-local-orb");
  // Obtain a reference to the RootPOA and its Manager
  CORBA::Object_var poaobj =
       orb->resolve_initial_references("RootPOA");
  PortableServer::POA_var poa =
       PortableServer::POA::_narrow(poaobj);
  PortableServer::POAManager_var mgr =
       poa->the_POAManager();
  // more POA setup...
```

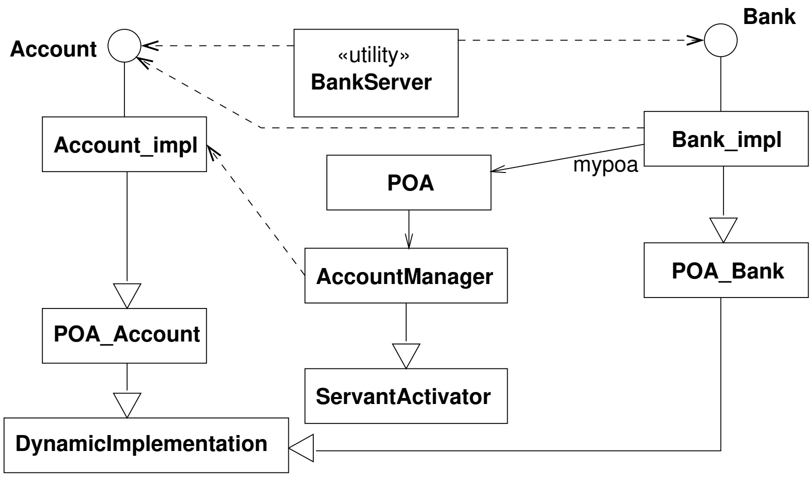
Slide 23

Banking server (C++) (2)

```
  // more POA setup...
  // Create a Bank
  Bank_impl* micocash = new Bank_impl(mypoa);
  // Activate the Bank
  PortableServer::ObjectId_var oid =
      poa->activate_object(micocash);
  // Activate both POAs and start serving requests
  mgr->activate();
  orb->run();
}
```

Slide 24

Banking server (UML)



4.4 Dynamic interfaces

Slide 25

Dynamic Invocation Interface

Static invocation (by way of stubs and skeletons) suits most applications, but some, such as browsers, design tools, and gateways, may need a more flexible mechanism.

- The interface of an object may not be known at compile time.
- A client may find out the details of an interface at run-time.
- The client may then use the *Dynamic Invocation Interface* to assemble a request "on the fly" and send it to the object.
- Dynamic invocations may be made in *deferred synchronous mode*: the caller executes in parallel with the target's processing of the request. The caller can later poll for the results.

Slide 26

Dynamic Skeleton Interface

- The *Dynamic Skeleton Interface* allows object requests to be delivered to objects whose interface is unknown to the (sever side) ORB at compile time.
- The DSI is particularly useful for gateways towards non-CORBA environments.
- The DSI is called by the Object Adapter.

Slide 27

Interface Repository

- The Interface Repository is a database containing IDL interface definitions.
- The IR may be dynamically accessed by ORB components, by clients, by object implementations, and other components, such as the IDL compiler.

4.5 The Object Management Architecture

The *Object Management Architecture* builds upon the core architecture (the ORB and its mechanisms and interfaces) to provide a comprehensive framework of higher-level services.

Slide 28

Interoperability and Protocols

- The *General Inter-ORB Protocol* specifies the message formats and data representation used by communicating ORB's.
- GIOP can be implemented on top of various reliable connection-oriented transport protocols, such as *Systems Network Architecture* (SNA), *Xerox Network Systems Internet Protocol* (XNS/ITP), and *Asynchronous Transfer Mode* (ATM).
- The *Internet Inter-ORB Protocol* (IIOP) is the TCP/IP-based implementation of GIOP.
- *Environment-Specific Inter-ORB Protocols* (ESIOP) are built on top of existing environments, such as OSF DCE, and are optimized towards them.