Distributed Systems
CORBA

Gianpaolo Cugola

Dipartimento di Elettronica e Informazione
Politecnico di Milano, Italy
cugola@elet.polimi.it
http://home.dei.polimi.it/cugola

Contents

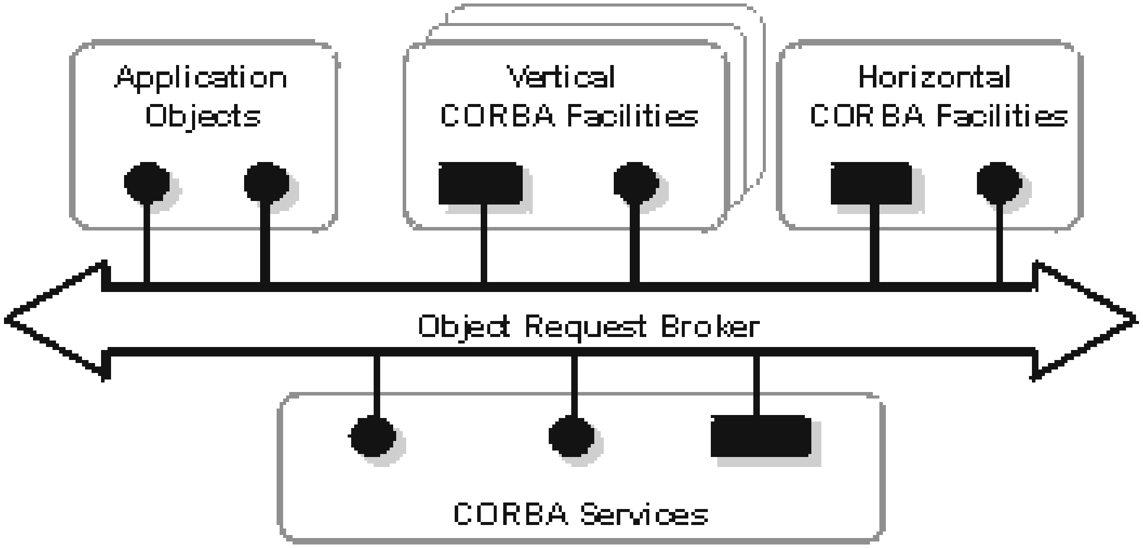
- **Introduction to CORBA**
- Developing CORBA based applications with J2SE 6
  - Writing the interface: The CORBA IDL and the mapping to Java
  - Implementing the servant and the client
  - Running the application
- More on object references and naming
  - Passing references as parameters and return values
  - Using stringified references
  - The Naming Service in details
    - Naming contexts
    - The corbaname url

Politecnico
di Milano

What is CORBA?

- CORBA = Common Object Request Broker Architecture
- It is the core of the *Object Management Architecture* (OMA)
- Developed by the *Object Management Group* (OMG), the OMA defines an *open framework* for OO distributed applications
  - Facilitates interoperability
  - Provides mappings to several (even non-OO) programming languages
- Allows developers to design a distributed application as a set of cooperating objects
- Distributed programs interact as they were on a single machine, regardless of
  - The programming language they are written in,
  - The hardware architecture they are running on
- The OMG is a standardization body. It provides specifications, not implementations. The latter are provided by *CORBA vendors*

The Object Management Architecture

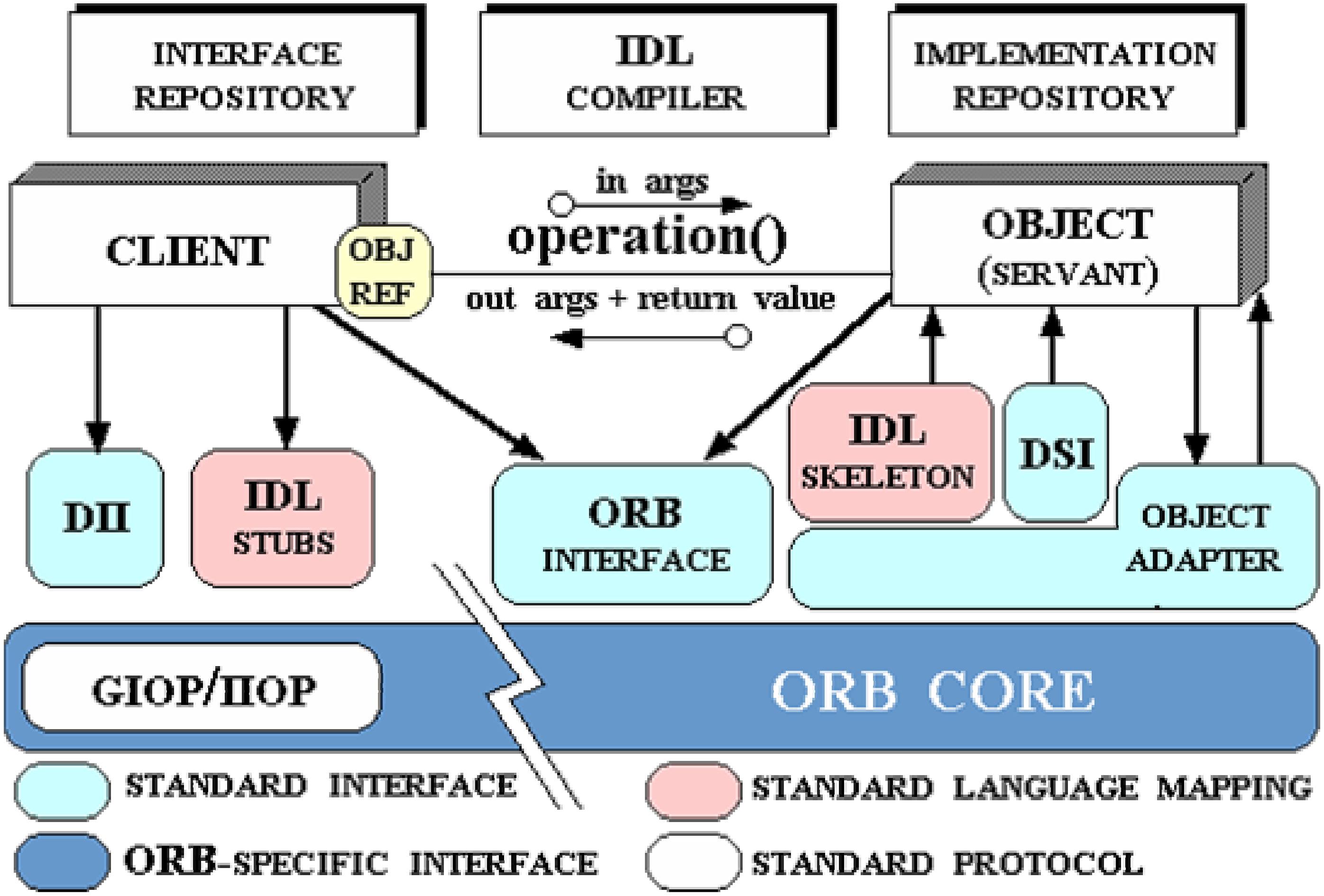


Interface Definition Language

- The OMA defines an *Interface Definition Language* (IDL) to specify the APIs of CORBA objects in terms of interfaces and operations
- The OMA includes the specification of how CORBA IDL has to be mapped to the main programming languages
  - C++
  - SmallTalk
  - Java
  - C
  - Cobol
  - ADA
  - Python
  - ...

Politecnico di Milano

Object Request Broker (1)



Object Request Broker (2)

- Client: The component that invokes a service
  - It has an reference to the remote object
- Servant: *Represents* a CORBA object, *it is not* a CORBA object
  - A CORBA object is just a concept. A servant, instead, is an object in the target programming language that is used to implement one or more CORBA objects
  - If the server process is restarted, a new servant will be created to represent the same CORBA object
- ORB Core: It is in charge of dispatching calls to remote objects, while hiding network communication from the programmer:
  - Locates the remote object on the network
  - Communicates the request to the object
  - Waits for the result
  - Sends the results back to the requester

Object Request Broker (3)

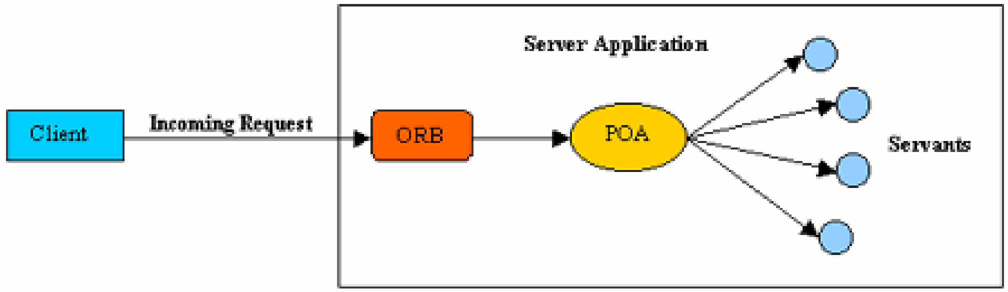
- ORB Interface: The standard interface to access core ORB services
  - To decouple client and server from the specific ORB implementation
- Stub e skeleton: Built from the remote object interface using the IDL compiler
  - Together with the ORB allows service requests issued by clients to be dispatched to the right remote object
- Dynamic Invokation Interface (DII): A standard interface used by clients to access the services provided by a remote object whose interface is not known at compile time
- Dynamic Skeleton Interface (DSI): A standard interface used to implement remote objects whose interface is not known at compile time

Object Request Broker (4)

- Object Adapter: The component that mediates the communication between the remote object and the ORB. It embeds the main mechanisms and policies to implement the following operations:
  - Registering, activating, and deactivating the servant
  - Creating and interpreting object references
  - Mediating service invocation
- GIOP e IIOP: Protocols used to connect client and server
  - Being standardized by the OMG they allow two or more ORBs developed by different vendors to interact

Politecnico
di Milano

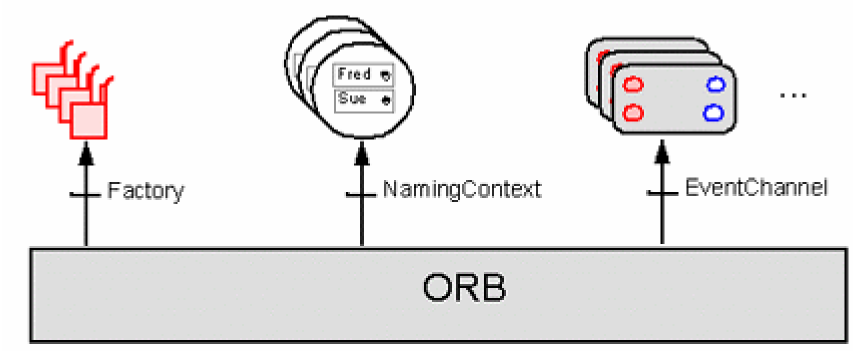
Portable Object Adapter (POA)



- The client invokes a request using a reference to the target object
- The request is then received by the ORB, which will dispatch the request to the POA that hosts the target object
- The POA will then dispatch the request to the servant
  - It performs the operation and sends the results back to the POA, to the ORB, and finally to the client
- Fulfill three requirements:
  - Create object references, which allow clients to address objects
  - Ensure each target object is incarnated by a servant
  - Takes requests dispatched by a client-side ORB and further directs them to servants incarnating each of the target objects

Politecnico di Milano

CORBA Services



- A set of standardized services, which provide basic functionality to support integration and interoperation of distributed objects
- Defined as standard CORBA objects with IDL interfaces
- Also known as "Object Services"

Main CORBA services

- The *Naming* and *Trading Object* services allow a server application to advertise its objects
  - Thereby making it easy for clients to find those objects
- The *Event* and *Notification* services support many-to-many, asynchronous communication
  - Used to implement the Publish-Subscribe interaction style
- The *Transaction* service provides transactional support
  - I.e., a set of (distributed) operation seen as atomic from the rest of the system
    - Either a transaction is committed, i.e., all the operation are successfully concluded
    - or a transaction is aborted, i.e. none of the operation are actually performed
- Much more...

Other elements of the OMA

- The *Horizontal CORBA Facilities* sit between the CORBA Services and Application Objects providing functionalities potentially useful across business domains
  - Currently there are only four Horizontal CORBA Facilities: The Printing Facility, the Secure Time Facility, the Internationalization Facility, and the Mobile Agent Facility
- The *Vertical CORBA Facilities* define standard interfaces for objects that every company in an industry wants to share
- Application Objects are the constituents of CORBA applications
  - Being typically customized for an individual application they are not standardized by the OMG

ORB implementations

- Different CORBA implementation exist, both commercial and free:
  - Orbacus (C++ / Java, free for academic use), http://www.orbacus.com
  - JacORB (Java, GPL), http://www.jacorb.org/
  - Orbix (C++, commercial), http://www.iona.com/
  - TAO (C++, open-source), http://www.cs.wustl.edu/~schmidt/TAO.html
  - OmniORB, MICO, ORBit, …

Essential Bibliography

- Ciaran McHale, “CORBA Explained Simply”, 2004
  - A very good introduction to CORBA concepts
- Specification written and maintained by the Object Management Group (OMG) http://www.omg.org

Contents

- Introduction to CORBA
- **Developing CORBA based applications with J2SE 6**
  - **Writing the interface: The CORBA IDL and the mapping to Java**
  - Implementing the servant and the client
  - Running the application
- More on object references and naming
  - Passing references as parameters and return values
  - Using stringified references
  - The Naming Service in details
    - Naming contexts
    - The corbaname url

CORBA and J2SE 6

- J2SE 6 (but also previous versions) include:
  - A basic (but fully functional) CORBA broker
  - An IDL to Java compiler
  - A Naming Service daemon
- No more services are provided
- A more complete open source product is JacORB. It includes several services:
  - Notification and Event Services
  - Transaction Service
  - Collection and Concurrency services
  - Trading Service
  - Data Distribution Service (DDS)
  - Object domain management service
- We will use J2SE but students are free to use JacORB in your projects

Interface Definition Language

- IDL's purpose is to allow object interfaces to be defined in a manner that is independent of any particular programming language or implementation
- To call a member function on a CORBA object, the client needs only the object's IDL
- Client need not know
  - The programming language used to implement the object's functionalities
  - The object's location
  - The operating system on which the server runs

More on IDL

- IDL supports *multiple inheritance* and *genericity*
- IDL operations support `in`, `out` and `inout` parameters
- Operations may throw *exceptions*, defined in IDL as well
- IDL has built-in types such as `string`, `boolean`, `int`, `long`, `float`, and `double`
- IDL allows to define complex types using `struct`, `sequence`, `array`, `typedef`, `enum`, and `union` constructs

IDL data types: Example

Enumerations

```
enum Genre {male, female};
typedef sequence<string> StringSeq;
struct AccountRecord{
        string name;
        Genre g;
        StringSeq address;
        long accountNumber;
        double balance;
        any data;
}
```

C/C++ like struct data type

A collection type, i.e., a dynamic array able to grow or shrink

Also known as “variant” type, can hold a value of any data type

IDL interfaces

- The `interface` is the main IDL type
- Defines the interface of a CORBA object as a set of methods
- Each method is defined as follow:

```
[oneway] <resType> <name>(par_1,..par_n)
  [raises(ex_1,...,ex_n)]
  [context(c_1,...,c_n)]
```

IDL: A complete example

File Account.idl

```
module Finance {
  struct AccountDetails {
    string     name;
    string     address;
    long       number;
    double     balance;
  };

  exception InsufficientFunds { };

  interface Account {
    void deposit(in double amount);
    void withdraw(in double amount) raises(InsufficientFunds);
    readonly attribute AccountDetails details;
  };
};
```

Compiling the IDL

- J2SE includes an IDL-to-Java compiler to generate stubs and skeletons
- To compile our example invoke it as follows:

```
idlj -fall Account.idl
```

- This produces a bunch of classes and interfaces, each on its own source file:
  - An `AccountDetails` class that implements the IDL struct
  - An `Account` interface
  - An `InsufficentFunds` exception
  - Helpers and Holders for the previous components
    - Each `XXXHelper` provides static methods to manipulate `XXX` instances. Most important is the `narrow` method for casting
    - Each `XXXHolder` wraps an `XXX` instance to pass it as an `out` or `inout` parameter
  - An `AccountOperations` interface containing the methods defined into the Account IDL type (the Account interface extends it)
  - An `AccountPOA` class providing basic CORBA functionality for the servant
  - An `_AccountStub` class which is there for backward compatibility (its place has been taken by the POA)

Object By Value

- A CORBA interface has operations but no state variables
- Conversely, a CORBA `struct` has state variables (fields) but no operations
- A `valuetype` has both operations and state variables
- When a `valuetype` is passed as a parameter, its state variables are transmitted
  - Operations invoked upon a `valuetype` are always invoked on the local copy
- However, there is no guarantee that the server-side implementation of the `valuetype` is semantically equivalent to the client-side implementation
  - Only state (not code), is transmitted across the network

Contents

- Introduction to CORBA
- Developing CORBA based applications with J2SE 6
  - Writing the interface: The CORBA IDL and the mapping to Java
  - **Implementing the servant and the client**
  - Running the application
- More on object references and naming
  - Passing references as parameters and return values
  - Using stringified references
  - The Naming Service in details
    - Naming contexts
    - The corbaname url

The servant

```
import Finance.*;

public class AccountImpl extends AccountPOA {
  private AccountDetails acc;

  public AccountImpl(String name, String address, int number) {
    acc = new AccountDetails(name, address, number, 0);
  }

  public void deposit(double amount) {
    acc.balance+=amount;
  }

  public void withdraw(double amount) throws InsufficientFunds {
    if(acc.balance<amount) throw new InsufficientFunds();
    acc.balance-=amount;
  }

  public AccountDetails details() {
    return acc;
  }
}
```

The server

```
import Finance.*;  import org.omg.CORBA.*;  import org.omg.PortableServer.*;
import org.omg.CosNaming.*;

public class AccountServer {
  public static void main(String args[]) {
    try{
      // create the servant
      AccountImpl accountImpl = new AccountImpl("Paolo Rossi", "Milano", 100);
      // create and initialize the ORB
      ORB orb = ORB.init(args, null);
      // get a reference to the rootpoa & activate its POAManager
      POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
      rootpoa.the_POAManager().activate();
      // activate the servant associating it to the poa and getting its object reference
      org.omg.CORBA.Object ref = rootpoa.servant_to_reference(accountImpl);
      // get the root naming context
      org.omg.CORBA.Object objRef =
          orb.resolve_initial_references("NameService");
      NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
      // bind the servant in naming
      NameComponent path[] = ncRef.to_name("RossiAccount");
      ncRef.rebind(path, ref);
      // wait for invocations from clients
      System.out.println("FinanceServer ready and waiting ...");
      orb.run();
    } catch(Exception e) { e.printStackTrace(); }
  }
}
```

The client

```
import Finance.*;  import org.omg.CORBA.*;  import org.omg.CosNaming.*;

public class AccountClient {
  public static void main(String args[]) {
    try{
      // create and initialize the ORB
      ORB orb = ORB.init(args, null);
      // get the root naming context
      org.omg.CORBA.Object objRef;
      objRef = orb.resolve_initial_references("NameService");
      NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
      // resolve the object Reference in naming
      String name = "RossiAccount";
      Account account = AccountHelper.narrow(ncRef.resolve_str(name));
      // invoke operations
      account.deposit(100);
      System.out.println("Current balance: "+account.details().balance);
      account.withdraw(20);
      System.out.println("Current balance: "+account.details().balance);
    } catch (Exception e) { e.printStackTrace(); }
  }
}
```

Contents

- Introduction to CORBA
- Developing CORBA based applications with J2SE 6
  - Writing the interface: The CORBA IDL and the mapping to Java
  - Implementing the servant and the client
  - **Running the application**
- More on object references and naming
  - Passing references as parameters and return values
  - Using stringified references
  - The Naming Service in details
    - Naming contexts
    - The corbaname url

Running the application

- Run the `orbd` daemon that implements the persistent name service provided with J2SE

```
orbd -ORBInitialPort 1050
```

- Run the server which instantiates and binds the servant

```
java AccountServer -ORBInitialPort 1050
  -ORBInitialHost localhost
```

- Run the client

```
java AccountClient -ORBInitialPort 1050
  -ORBInitialHost localhost
```

Contents

- Introduction to CORBA
- Developing CORBA based applications with J2SE 6
  - Writing the interface: The CORBA IDL and the mapping to Java
  - Implementing the servant and the client
  - Running the application
- **More on object references and naming**
  - **Passing references as parameters and return values**
  - Using stringified references
  - The Naming Service in details
    - Naming contexts
    - The corbaname url

Passing references around

```
interface Bank {
  Account createNewAccount(in string name,
                           in string address);
  Account getAccount(in long number);
};
```

Passing references around

```
import Finance.*;  import java.util.*;

public class BankImpl extends BankPOA {
  private List<Account> accounts;
  public BankImpl() { accounts = new ArrayList<Account>(); }
  public Account createNewAccount(String name, String address) {
    try {
      int number = accounts.size();
      AccountImpl accImpl = new AccountImpl(name, address, number);
      org.omg.CORBA.Object ref = _poa().servant_to_reference(accImpl);
      Account acc = AccountHelper.narrow(ref);
      accounts.add(acc);
      return acc;
    } catch(Exception e) { e.printStackTrace(); return null; }
  }
  public Account getAccount (int number) {
    return accounts.get(number);
  }
}
```

Passing references around

```
import Finance.*;  import org.omg.CORBA.*;  import org.omg.CosNaming.*;

public class BankClient {
  public static void main(String args[]) {
    try{
      // create and initialize the ORB
      ORB orb = ORB.init(args, null);
      // get the root naming context
      org.omg.CORBA.Object objRef;
      objRef = orb.resolve_initial_references("NameService");
      NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
      // resolve the object Reference in naming
      Bank bank = BankHelper.narrow(ncRef.resolve_str(“MyBank”));
      // invoke operations
      Account acc1 = bank.createNewAccount("Rossi", "Milano");
      Account acc2 = bank.createNewAccount("Verdi", "Roma");
      acc1.deposit(20);  acc2.deposit(100);
      System.out.println("acc1: "+acc1.details().name+" “+
            acc1.details().number+" "+acc1.details().balance);
      System.out.println("acc1: "+acc2.details().name+" "+
            acc2.details().number+" "+ acc2.details().balance);
    } catch (Exception e) { e.printStackTrace(); }
  }
}
```

Contents

- Introduction to CORBA
- Developing CORBA based applications with J2SE 6
  - Writing the interface: The CORBA IDL and the mapping to Java
  - Implementing the servant and the client
  - Running the application
- More on object references and naming
  - Passing references as parameters and return values
  - **Using stringified references**
  - The Naming Service in details
    - Naming contexts
    - The corbaname url

References as strings

- Each object defined in a CORBA environment has a 128-byte unique identifier called *Interoperable Object Reference* - IOR
  - It contains the *contact details* that a client application uses to communicate with a CORBA object
  - It is interoperable as it works across different implementations of CORBA
- Two methods allows object references to be transformed into strings and viceversa

```
String object_to_string(orm.omg.CORBA.Object)
org.omg.CORBA.Object string_to_object(String)
```

- Both are exported by the `org.omg.CORBA.ORB`

Politecnico di Milano

References as strings: Example

- Into the server:

```
PrintWriter out = new
  PrintWriter("RossiAccount.ref");
out.println(orb.object_to_string(ref));
out.close();
```

- Into the client:

```
BufferedReader in = new BufferedReader(
  new FileReader("RossiAccount.ref"));
String objRefAsString = in.readLine();
in.close();
objRef = orb.string_to_object(objRefAsString);
Account acc = AccountHelper.narrow(objRef);
```

Contents

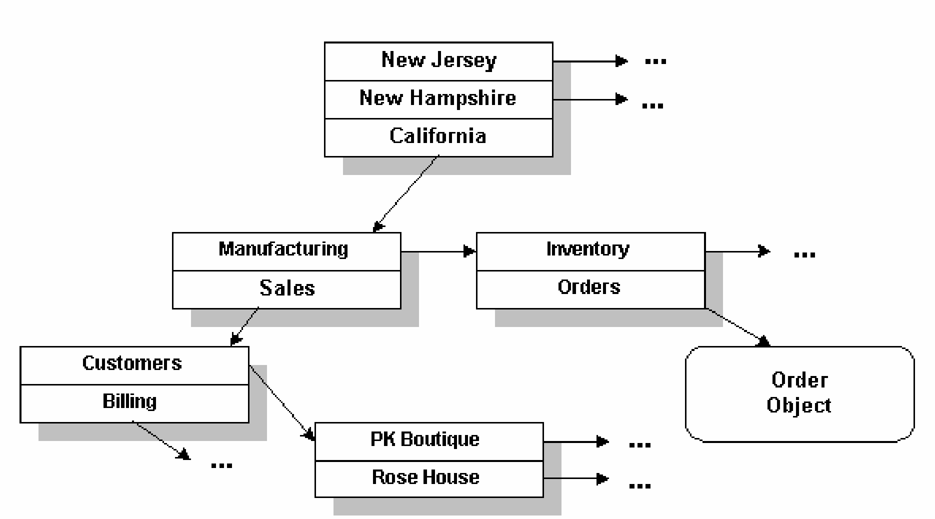
- Introduction to CORBA
- Developing CORBA based applications with J2SE 6
  - Writing the interface: The CORBA IDL and the mapping to Java
  - Implementing the servant and the client
  - Running the application
- More on object references and naming
  - Passing references as parameters and return values
  - Using stringified references
  - **The Naming Service in details**
    - **Naming contexts**
    - **The corbaname url**

The CORBA Naming Service: Introduction

- The CORBA Naming Service provides a mapping from a (human-readable) name to an object's IOR
- It supplies operations to:
  - Create / delete / modify naming contexts (i. e. directories),
  - Bind (that is advertise) an IOR in the Naming Service with a specified name,
  - Resolve (that is lookup) an IOR associated with a specified name

Politecnico
di Milano

The CORBA Naming Service: Example



- Example: namespace organization by geographic region, then by department
- Each shadowed box is implemented by a `NamingContext` object
- `NamingContext` objects are traversed to locate a particular name
- For example, the logical name `California/Manufacturing/Orders` can be used to locate the `Order` object

The CORBA Naming Service: Objects and Data Structures

- A naming context is itself a CORBA object (defined by `CosNaming::NamingContext` in IDL)
- A `NamingContext` object contains a list of `CosNaming::NameComponent` that have been bound to application objects or to other `NamingContext` objects
- A `CosNaming::NameComponent` data structure contains two strings, an id string and a kind string
- If object ‘c’ lives in naming context nc2, and nc2 lives in naming context nc1, a reference to ‘c’ can be obtained with:

```
NameComponent[] cName;
cName = new NameComponent[3];

cName[0] = new NameComponent();
cName[0].id = “nc1”;
cName[0].kind = “”;
cName[1] = new NameComponent();
cName[1].id = “nc2”;
cName[1].kind = “”;
cName[2] = new NameComponent();
cName[2].id = “c”;
cName[2].kind = “”;

org.omg.CORBA.Object obj =
         nc.resolve (cName);
```

NamingContextExt

- Interface `NamingContextExt` extendes the `NamingContext` interface providing methods to more easily convert from strings to `NameComponents` and viceversa

```
NameComponent[] to_name(String sn)
String to_string(NameComponent[] n)
Object resolve_str(String sn)
```

- Example

```
org.omg.CORBA.Object objRef =
  orb.resolve_initial_references("NameService");
NamingContextExt ncRef =
  NamingContextExtHelper.narrow(objRef);
NameComponent path1[] = ncRef.to_name("Finance");
ncRef.bind_new_context(path1);
NameComponent path2[] =
  ncRef.to_name("Finance/RossiAccount");
ncRef.rebind(path2, ref);
```

The corbaname URL

- The string_to_object method of the ORB can be used to refer to objects bound to the nam service through the "corbaname" URL
- Example:

```
org.omg.CORBA.Object objRef =
  orb.string_to_object("corbaname::loca
  lhost:1050#Finance/RossiAccount");
```

Esercizio

- Si vuole implementare un servizio per l'accesso da remoto alla base dati di una biblioteca civica
  - Tale base di dati mantiene informazioni sui libri e sul nome dei clienti a cui ogni libro sia stato prestato
- Il servizio fornisce metodi per:
  - Aggiungere un libro alla biblioteca (servizio amministrativo)
  - Memorizzare il fatto che un libro è stato prestato
  - Memorizzare il fatto che un libro è stato restituito
  - Indagare relativamente allo stato di un libro (disponibile/prestato)
- Estensione
  - Aggiungere un metodo attraverso il quale un cliente possa registrarsi per essere informato quando un libro torna disponibile (registrazione del client come listener)