NET.OBJECTDAYS 2000

Java and Distributed Systems

Dr. Stephan Fischer
GMD-IPSI
Dolivostr. 15
D-64293 Darmstadt
sfischer@darmstadt.gmd.de

Contents

- **Remote Method Invocation**
- **Java and CORBA**
- **Jini**
- **Discussion**

Java RMI (1)

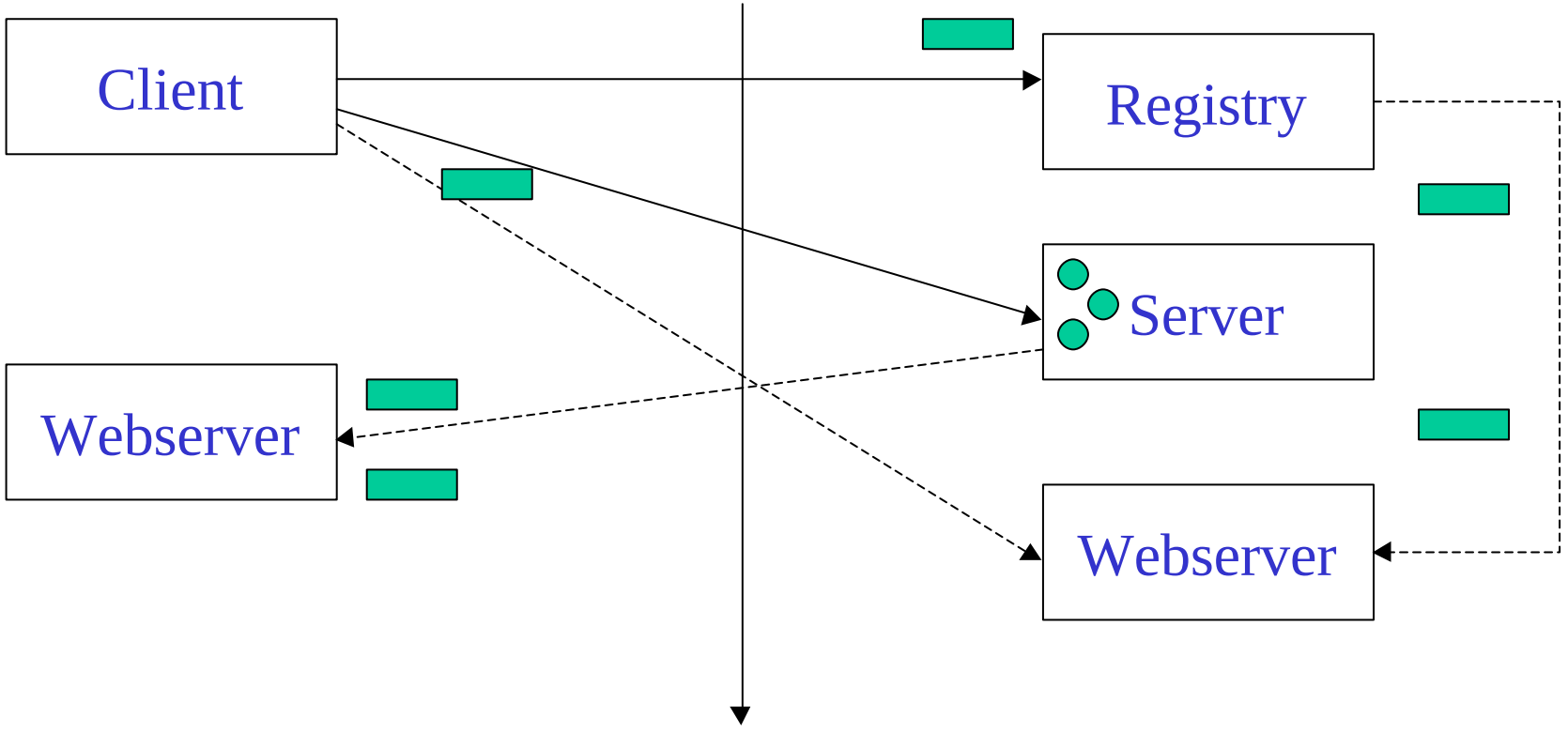
- RMI applications: Client/server architecture
- Client requests service which is executed transparently on server
- Typical server creates remote objects and offers them via references to client
- Client accesses objects by references
- RMI: Mechanisms for Client/server communication: Distributed applications/systems

Java RMI (2)

- Properties of distributed objects
  - localization of objects
    - application registers for remote object (RMIregistry)
    - exchange references to remote objects
  - communication with remote objects
    - hidden from application by RMI
    - appear like local calls
  - class loading of remote objects

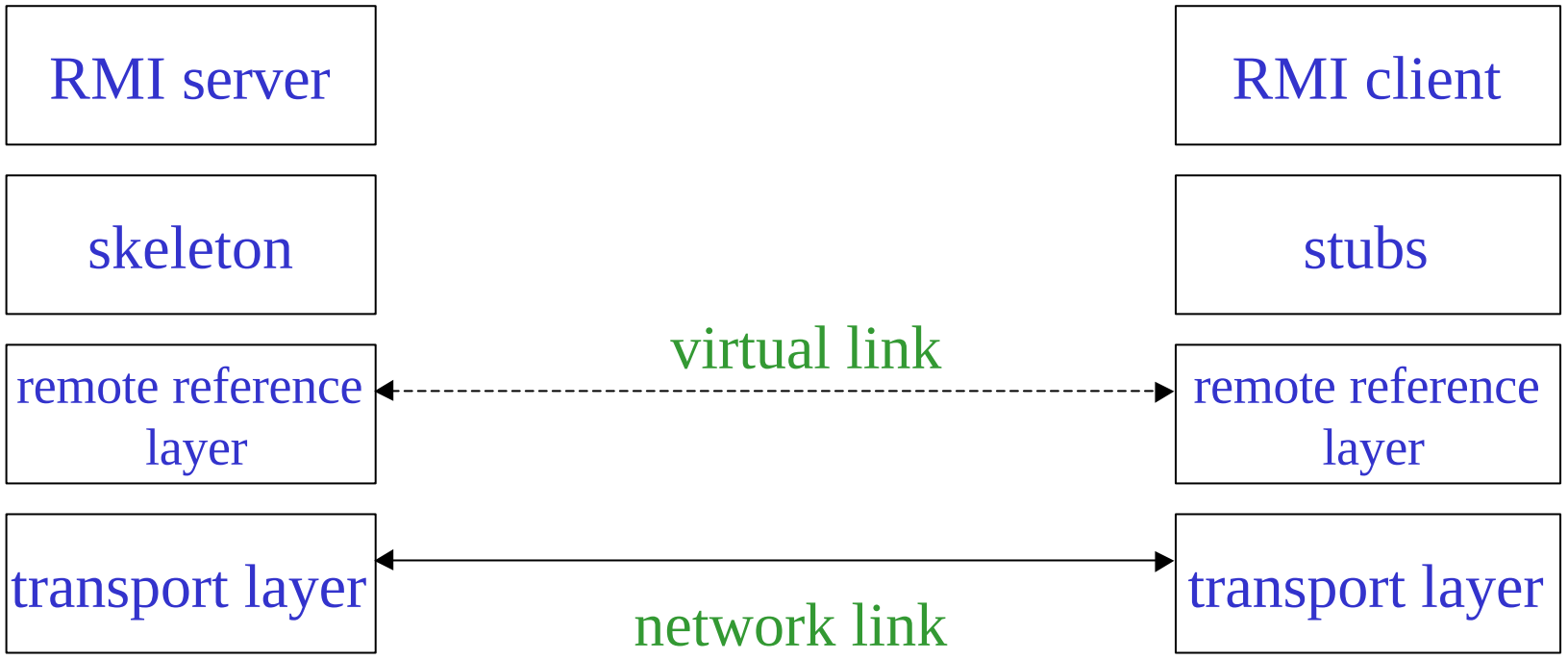
Java RMI (3)

- RMI functionality



Java RMI (4)

- RMI: Architecture



Java RMI (5)

- Skeletons
  - server-side component interacting with server-RRL
  - receive method call requests of client-RRL
  - extract arguments and call server method
  - accept return values and marshals them

Java RMI (6)

- Stub
  - represents remote object at client side
  - defines interfaces to support implementation of remote object
  - is referenced like local objects
  - client-side RRL passes stream (objects, arguments) to stub
  - serializes parameter data and passes to stream
  - deserializes received objects

Java RMI (7)

- Remote Reference Layer
  - referencing protocol, independent from stub or skeleton models
  - client-side: Server-specific information, passed to server-side RRL
  - provides referencing semantics and processes data according to semantics before passing to skeleton for method call

Java RMI (8)

- Transport layer
  - ... manages links between client and server
  - ... realizes abstractions:
    - endpoint: used to reference address space of VM
    - channel: link between address spaces. Manages link between client and server
    - link
    - transport: channel establishment between local address space and remote end-point

Java RMI (9)

- Properties of RMI
  - callback operations
  - dynamic class loading
  - remote interfaces, objects and methods
  - object persistence
  - object activation

Java RMI (10)

- Callback operations
  - two-way communication between objects
  - implementation of interface `Remote` to use object reference passed from one computer to another
  - object export to be able to receive remote method calls (extending `UnicastObject`)

Java RMI (11)

- Callback: Example bank transaction
  - client/server model: Client requests account value from server
  - extended model: Client requests account value from server, server requests password
  - correct answer only provided if password correct

Java RMI (12)

- Dynamic class loading
  - RMI: System of distributed Java-to-Java applications
    - migrate objects from address space 1 to address space 2 (different from CORBA!)
  - transmit (static) data and (dynamic) behavior, as well as class definitions (serialization and RMI class loader)
  - required: Implementation of interface `Serializable`

Java RMI (13)

- Remote interfaces, objects and methods
  - remote object implements remote interface
    - which extends `java.rmi.Remote`
    - each interface method contains `java.rmi.RemoteException` within `throws`-clause
  - uses stub object to access remote object instead of local copy
  - application calls local stub which calls remote skeleton

Java RMI (14)

- Object persistence
  - remote method call requires object passing
    - implementation of interfaces `Serializable` or `Externalizable`
  - rules
    - pass remote objects with a reference, realized as stub (client-side proxy) implementing remote interfaces which remote object implements
    - pass local objects as copies (using serialization)

Java RMI (15)

- Object activation
  - normally: Server generating object instance is running inside VM to obtain reference to remote object
  - large-scale systems: Deactivate objects which are not needed, activate necessary objects
  - RMI: Activation mechanism
    - assign name to object by registry
    - activate object later by referencing using registry
    - advantage: Application which generates instances of remote object can terminate before object is needed.
    - use RMI Actication System Daemon (rmid)

Java RMI (16)

- Object activation
  - client requests object name using registry
  - registry responds with remote reference which causes rmid call
  - client calls remote method using rmid (not Registry!) using remote reference
  - rmid passes call to instance of object implementation
  - start rmid as background process: `start rmid`
  - object: implement interface `Activatable` instead of `UnicastObject`

Java RMI (17)

- Distributed applications in RMI
  - development and implementation of components of distributed application
  - compilation of source files and generation of stubs
  - creation of network connectivity of classes using WWW server
  - start of application

Java RMI (18)

Development and implementation of components of distributed application

- Architecture definition: Creation of local and remote components
  - definition of remote interfaces
  - implementation of remote objects
  - implementation of clients

Java RMI (19)

Compilation of source files and generation of stubs

- compilation of source files using javac
  - implementation of remote interfaces, server and client classes
- compilation of new source files using rmic
  - creation of stubs for remote objects

Java RMI (20)

Start of application

- start of RMI registry for remote objects: `rmiregistry`
  - mapping of object names to objects, resolving of external references
- start of server
- start of client

RMI Example (1)

Application

- client calls remote service
- server accepts request and processes service
- server returns result to client

RMI Example (2)

- Definition of interface

```
import java.rmi.*;
public interface AccountUpdate extends
  Remote {
  public int updateAccount(int number)
  throws RemoteException;
}
```

RMI Example (3)

- Implementation class

```
import java.rmi.*;
import java.rmi.server.*;
import java.io.Serializable;
public class AccountUpdateImpl extends
   UnicastRemoteObject implements AccountUpdate,
   Serializable {
   public AccountUpdateImpl() throws RemoteException {}
   public int updateAccount (int number) throws
   RemoteException {
        //implemented later
        return 0;
   }
}
```

RMI Example (4)

- Stub and skeleton classes
  - (compilation of classes using javac)
  - stub and skeleton classes access implementation classes
  - `rmic AccountUpdateImpl` creates ...
    - AccountUpdateImpl_Skel.class
    - AccountUpdateImpl_Stub.class

RMI Example (5)

- Server application

```
import java
import java.rmi.registry.*;
public class Account{
   public static void main (String args[]) {
       if (args.length != 2) {
             System.err.println("Call: java
               Account <Server> <Port>");
             System.exit(1);
       }
       String server = args[0];
       int port = Integer.parseInt(args[1]);
```

RMI Example (6)

```
//New Security Manager
System.setSecurityManager(new
      RMISecurityManager());
try {
      //Location of Registry
      LocateRegistry.createRegistry(port);
      //Instance of Account application
      AccountUpdateImpl aui = new
            AccountUpdateImpl();
      //Bind object instance to registry
      String urlString = "//" + server +
            ":" + port + "/AccountUpdate";
```

RMI Example (7)

```
                // rebind
                Naming.rebind(urlString, aui);
            }catch (Exception e) {
                System.out.println("Error");
                e.printStackTrace();
                System.out.println(e.getMessage());
            }
        }
    }
```

RMI Example (8)

- Problem: RMI uses `System.setSecurityManager`
  - file `java.policy` has to be extended:

```
grant codeBase "file:<directory>"
  {permission
  java.security.AllPermissions;
};
```

RMI Example (9)

- Client application

```
import java.rmi.*;
public class Client {
  public static void main (String args[]) {
      AccountUpdate au = null;
      if (args.length != 2) {
            System.err.println("Eingabe: java
              Client <Server> <Port>");
            System.exit(1);
      }
      String server = args[0];
      int port = Integer.parseInt(args[1]);
```

RMI Example (10)

```
//New Security Manager
System.setSecurityManager(new
  RMISecurityManager());
try {
  //Binding of object instance to registry
  String url = "//" + server + ":" + port +
     "/AccountUpdate";
  au = (AccountUpdate)Naming.lookup(url);
} catch (Exception e) {
   System.err.println("No Connection"+e);
}
```

RMI Example (11)

```
        try {
              //update account
              int result = au.updateAccount(10);
        } catch (Exception e) {
              System.err.println("Error");
        }
    }
}
```

Java and CORBA (1)

Common Object Request Broker Architecture

- framework for distributed systems
- to support heterogeneous architectures
  - hardware (CPU, file system)
  - operating system and programming language
  - RMI: focuses on homogeneous architectures (both communication partners use Java)
- management of system resources providing different services

Java and CORBA (2)

Object Management Group (OMG)

- consortium of engineers developing architecture to
  - enable transparent cooperation of programming and operating system environments
- result of specifications: CORBA
- Object Request Broker (ORB): Heterogeneous object components can collaborate over networks and operating system borders
- interfaces to CORBA objects: Interface Definition Language (IDL)

Java and CORBA (3)

- CORBA and other programming languages
  - CORBA objects
    - can be at arbitrary locations within network
    - cooperate with objects of other platforms
    - can be implemented in various programming language. Constraints: Mapping from OMG-IDL to programming language necessary.
      - Java, C, C++, ...

Java and CORBA (4)

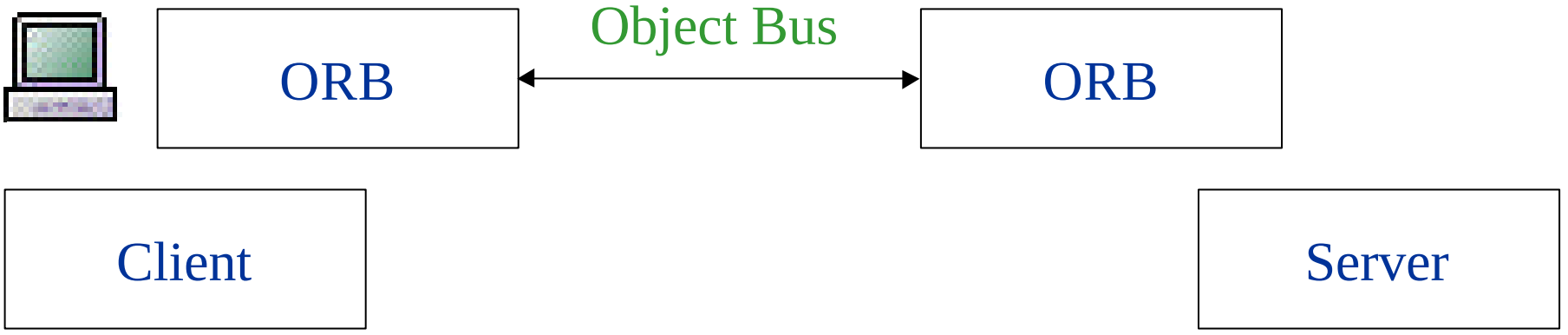
Commercial CORBA Components

- implementation of ORB
- IDL compiler
- implementation of Common Object Services (COS, CORBAServices)
- application-dependent environments (CORBAFacilities)
- since CORBA2.0: Internet-Inter-ORB protocol (IIOP)

Java and CORBA (5)

Object Request Broker (ORB)

- link objects of isolated address spaces
- use marshaling to bundle parameters, return values, and exceptions



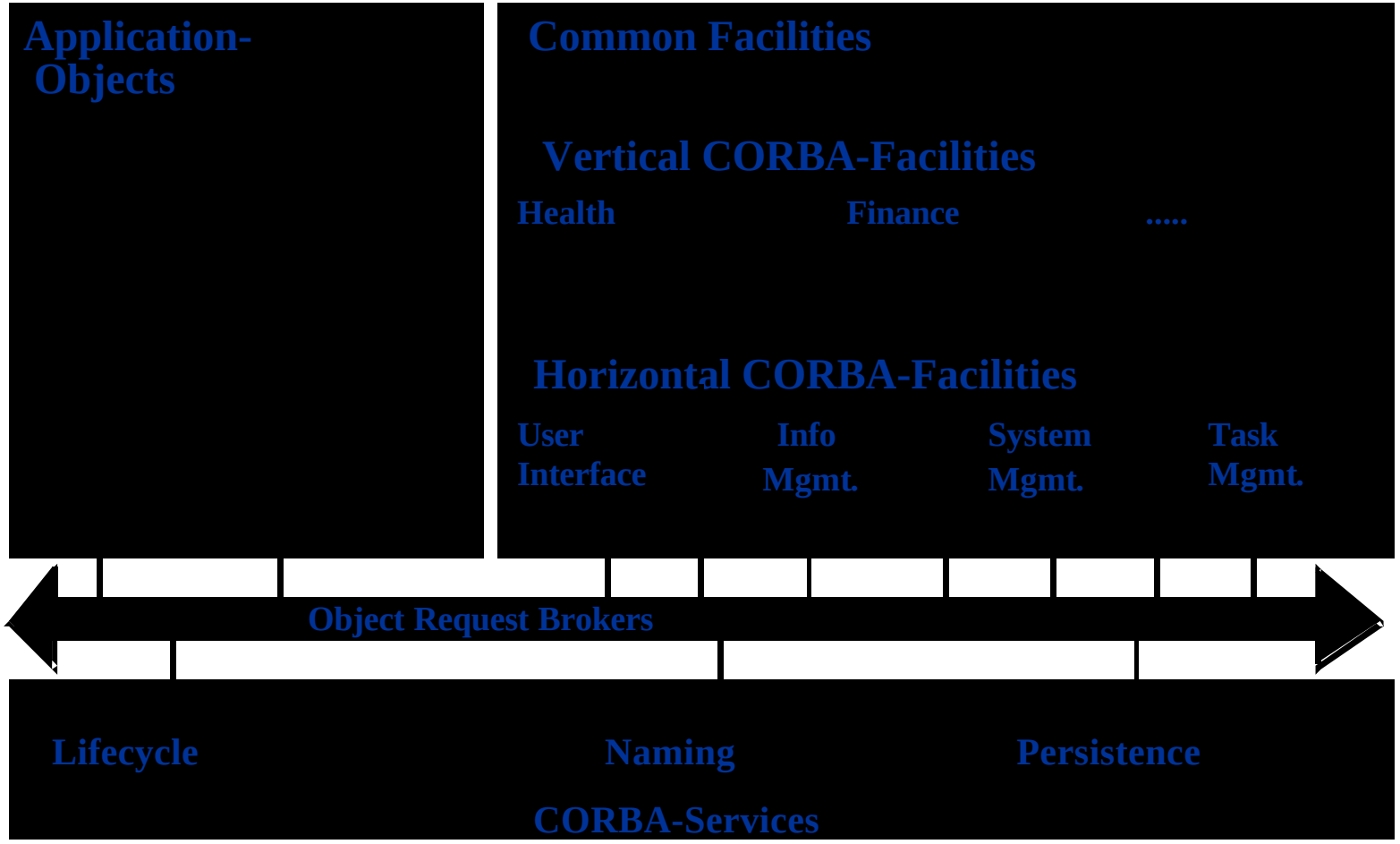
Java and CORBA (6)

- ORB examples
  - ORB resident within client and object implementation
  - server-based ORB
  - system-based ORB
  - runtime-library-based ORB

Java and CORBA (7)

- Common Object Services (COS) support ORB
  - naming service
  - event service
  - security service
  - transaction service

Java and CORBA (8)



Java and CORBA (9)

- Interface Definition Language (IDL)
  - interface for ORB, COS, and Common Facilities
  - describes neutral interface language for service implementations
    - simple data types (basic values). Example: Short, Long
    - complex data types (constructed values). Example: Struct, Sequence

Java and CORBA (10)

- Example

```
module demo {
  interface text {
     readonly attribute string message;
     long connect (in long id);
  };
};
```

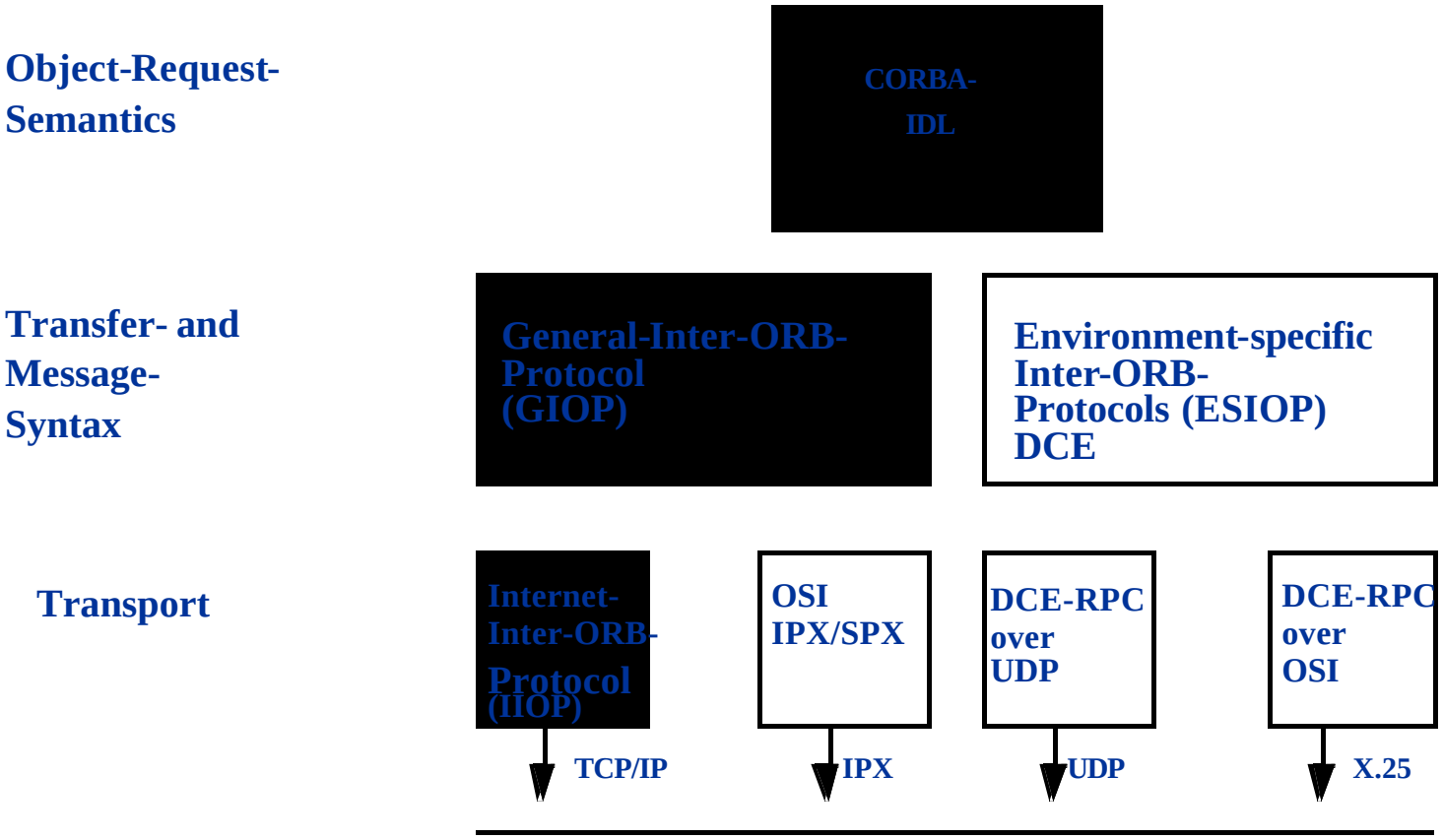
Java and CORBA (11)

- Internet-Inter-ORB Protocol (CORBA2.0)
  - Universal Networked Objects (UNO)
    - using TCP/IP and General-Inter-ORB protocol (GIOP)
  - GIOP defines communication of ORBs

Java and CORBA (12)

- UNO use interoperable message format: Common Data Representation (CDR)
  - example: conversion IDL to network format, byte order
  - difference to External Data Representation (XDR): Variable byte order for systems with same word format

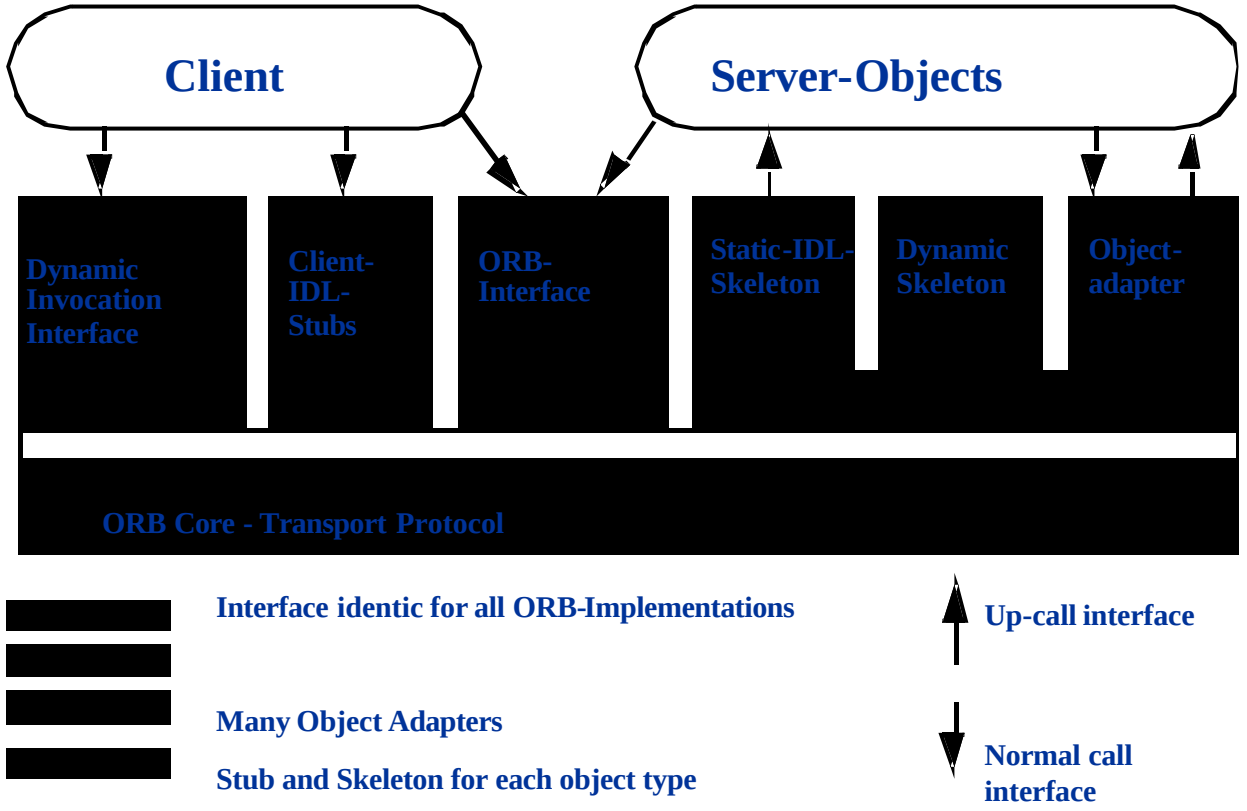
Java and CORBA (13)



Java and CORBA (14)

- Using CORBA: Steps
  - design development
  - object development using IDL interfaces, grouping to modules
  - creation of stubs and skeletons using IDL compiler
  - implementation of interfaces
  - server start and object initialization using naming service

Java and CORBA (15)



Java and CORBA (16)

- CORBA objects
  - are described using IDL-file (definition of object type)
  - IDL interface declares operations, exceptions, and type attributes (values)
  - operation: Consists of signature (name, parameters, result, and exceptions)

Java and CORBA (17)

- Example:

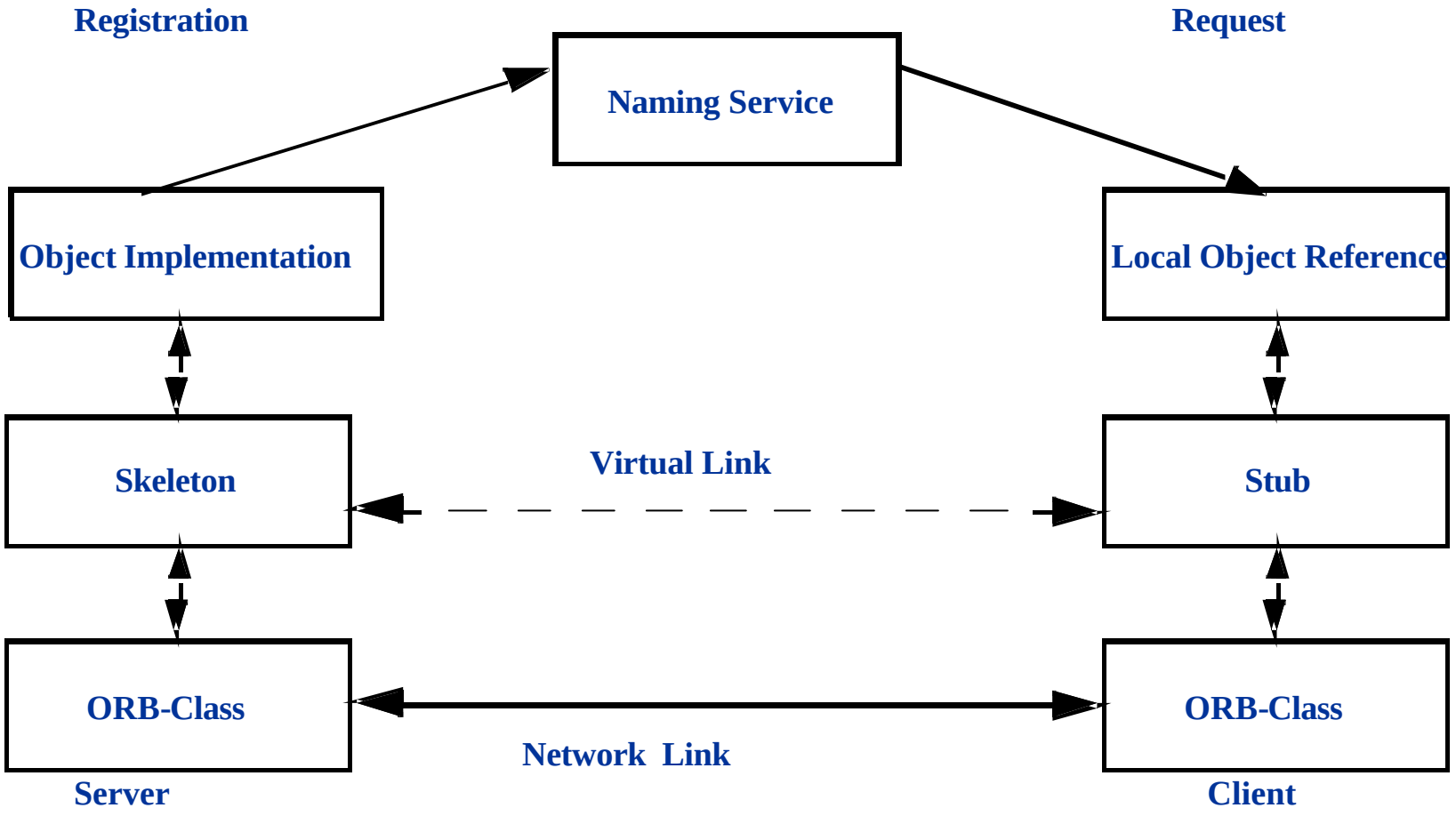
```
module AccountApp {
   interface AA {
       string hello();
   };
};
```

- idltojava: Creates package `AccountApp`, and interface `AA.java` within directory `AccountApp`

Java and CORBA (18)

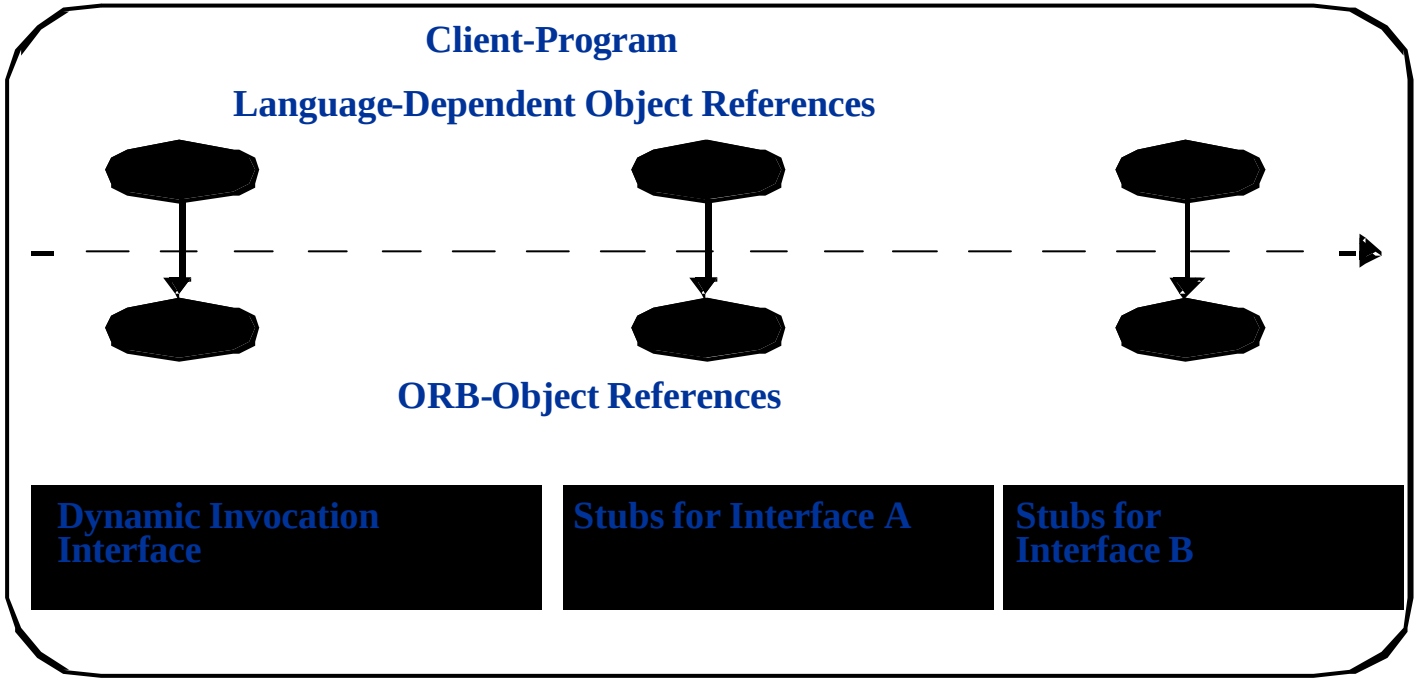
- Stubs and skeletons
  - extend common ORB class (object communication)
  - use Naming Service to offer/retrieve information about object

Java and CORBA (19)



Java and CORBA (20)

- CORBA Client

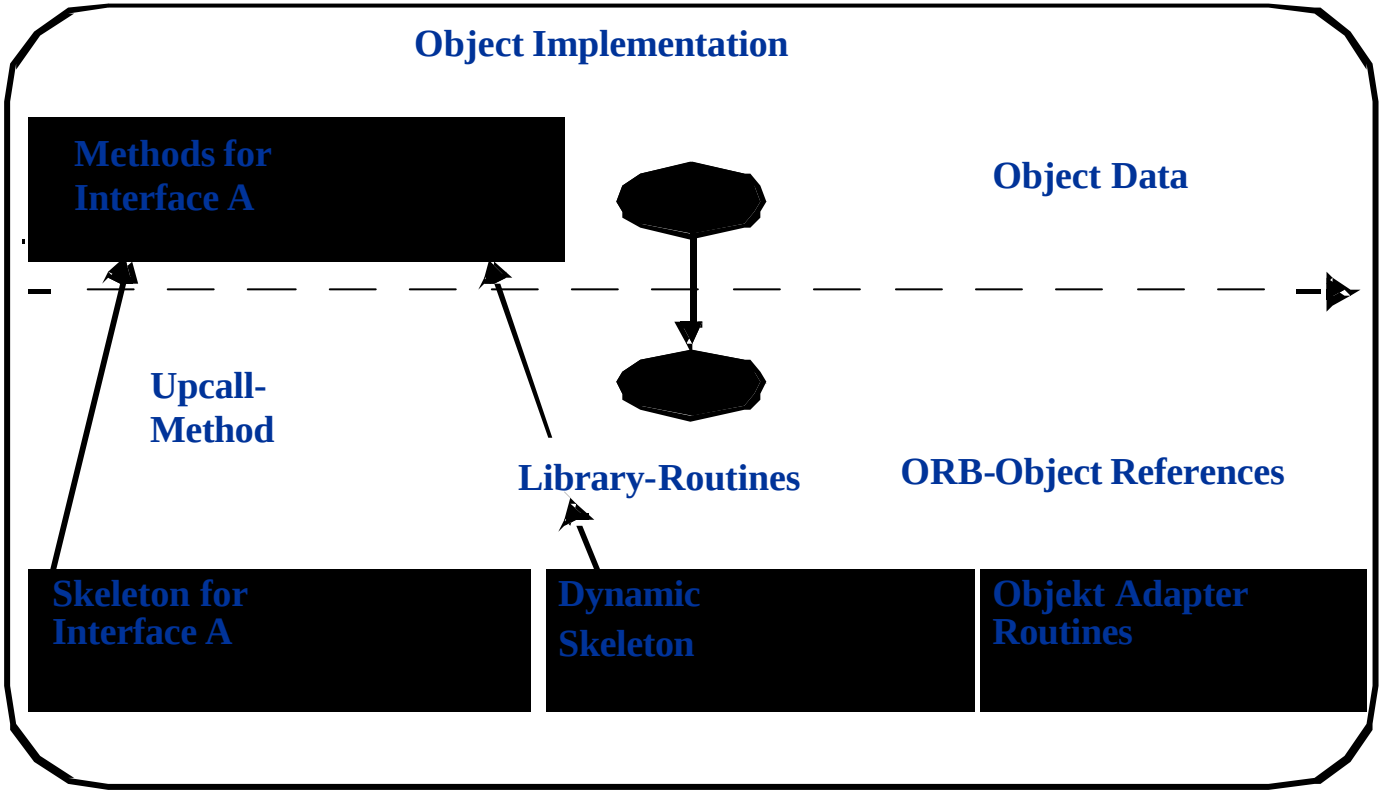


Java and CORBA (21)

- CORBA Server
  - offers references to CORBA objects (Naming Services)
    - transient objects
    - persistent objects (ORB demon starts necessary server)
    - Java IDL: exclusively transient objects

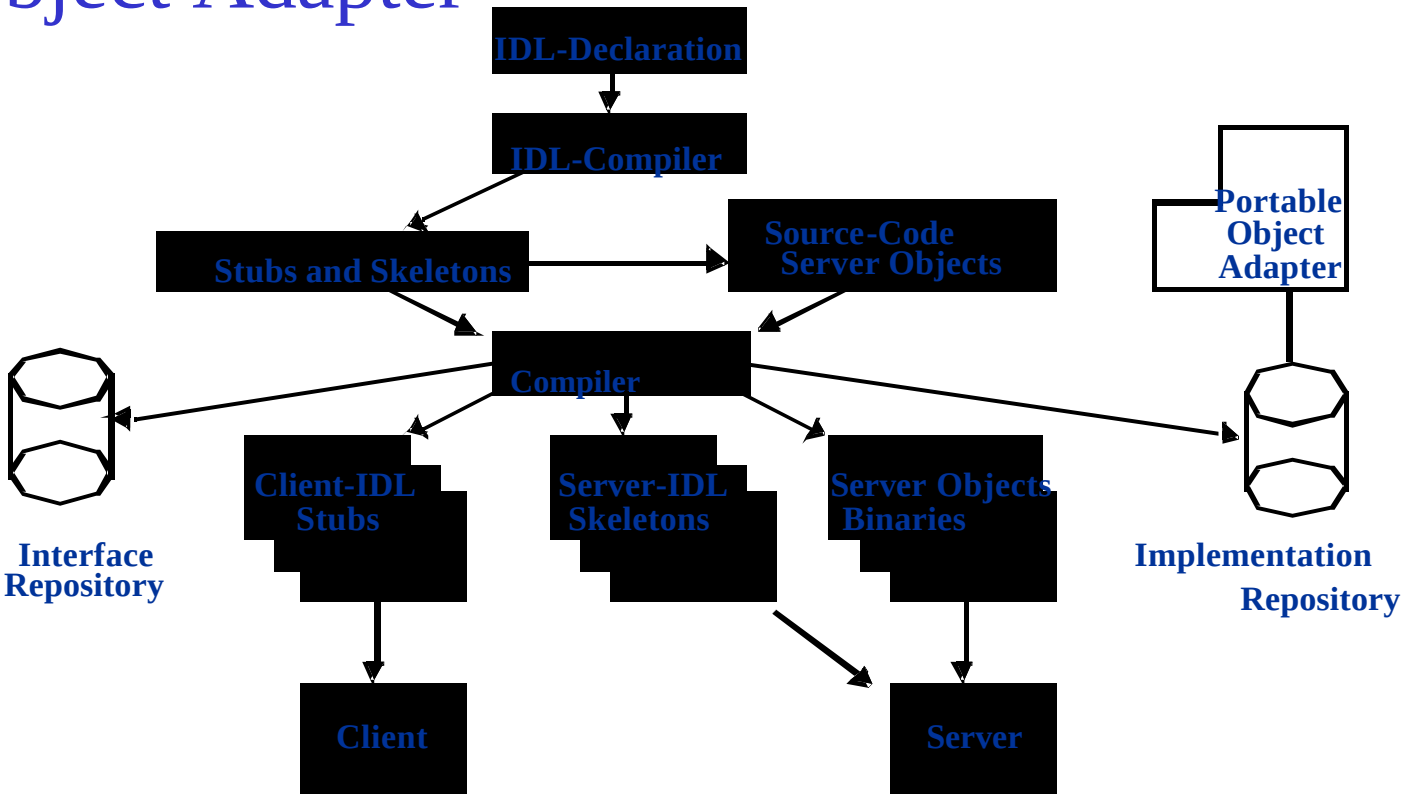
Java and CORBA (22)

- CORBA-Server (2)



Java and CORBA (23)

- Object Adapter



Java and CORBA (24)

- Java IDL
  - IDL file contains elements defining name space
  - IDL declaration are case insensitive
  - IDL does not support overloading and overwriting of operations, although (simple and multiple) heredity possible

Java and CORBA (25)

- IDL Modules
  - definition of group of IDL interfaces
  - nesting of modules possible
  - translated to Java package
  - example
    - module App {
      - interface AA {
        - » string hello();
      - };
    - };

Java and CORBA (26)

- IDL Interfaces
  - contain attributes, exceptions, and operations
  - attributes: Examples
    - `boolean` (IDL): `byte` (Java)
    - `enum`, `struct` (IDL): `class` (Java)
    - `long` (IDL): `int` (Java)
    - `long long` (IDL): `long` (Java)
    - `string` (IDL): `java.lang.String` (Java)
    - `unsigned short` (IDL): `short` (Java)

Java and CORBA (27)

- IDL interfaces: translation by idltojava
  - interface class with same name as interface name
  - implementation base class (skeleton code for server)
  - stub class
  - helper class (restrict object reference to stub object requested by client)
  - holder class (reference to IDL interface object if interface passed as argument)

Java and CORBA (28)

- IDL attributes: Translation by idltojava
  - creates getter and setter method
  - `readonly`: omit setter method
  - example
    - IDL: `attribute long zaehler;`
    - Java:
      - `int zaehler();`
      - `void zaehler(int arg);`

Java and CORBA (29)

- IDL operations
  - identifier `in`: call by value
  - identifier `out`: call by reference (Java: class `<JavaType>Holder` to encapsulate data variable containing containing parameter. Passing of class reference)
  - identifier `inout`: call by value and call by reference
  - identifier `raises`: define exception handling

Java and CORBA (30)

- IDL Exceptions
  - passed as object references
  - extend class `org.omg.CORBA.UserException`
  - example:
    - exception PwdException {
      - string reason;
    - };
    - interface password {
      - void getPassword(out string pwd) raises (PwdException)
    - };

Java and CORBA (31)

- IDL struct
  - container class to group data
  - example:
    - struct Account {
      - long number;
      - long amount;
    - };

Java and CORBA (32)

- IDL typedef
  - define new IDL types
  - are not mapped directly to Java: IDL compiler replaces typedef with IDL type
  - example:
    - typedef string Password;

Java and CORBA (33)

- IDL sequence
  - definition of one-dimensional arrays (bounded and intervals)
  - marshaling: Check if array boundaries are OK
    - if not: MARSHAL-exception
  - create helper and holder class within Java
  - example
    - typedef sequence <string, 10> passwords;

Java and CORBA (34)

- IDL array
  - bounded arrays
  - example:
    - typedef string passwords[10];
- IDL enum
  - enumeration: mapped to final Java class
  - example
    - enum PasswordList {pwd1, pwd2};

Java and CORBA (35)

- Application example: Steps
  - implementation of IDL interface
  - compilation using `idltojava`
  - compilation of created Java classes using `javac`
  - implementation of implementation classes
  - creation of implementation server
  - creation of client application
  - start naming service `tnameserv`
  - start server which is registered at naming service
  - start client

Java and CORBA (36)

- IDL interface

```
module AccountUpdate {
      //Update Account
      interface Functions {
         long updateAccount(in long number);
      };
};
```

Java and CORBA (37)

- Compilation of IDL file
  - `idltojava -fno-cpp AccountUpdate.idl`
    - `-fno-cpp`: switch off C/C++ preprocessor
- Compilation of Java classes
  - `javac AccountUpdate\*.java`

Java and CORBA (38)

- Generation of implementation classes
  - abstract file `_FunctionsImplBase.java`
  - provide body for each interface method

```
/*
 * File: ./ACCOUNTUPDATE/FUNCTIONS.JAVA
 * From: ACCOUNTUPDATE.IDL
 * Date: Wed September 27 15:02:54 2000
 *   By: idltojava Java IDL 1.2 Aug 18 1998 16:25:34
 */
package AccountUpdate;
public interface Functions
        extends org.omg.CORBA.Object, org.omg.CORBA.portable.IDLEntity
          {
        int updateAccount(int number);
}
```

Java and CORBA (39)

- Implementation of interface

```
//Implementation of Interface Functions.java
import AccountUpdate.*;
public class FunctionsImpl extends _FunctionsImplBase {
       public FunctionsImpl() {
       }
       public int updateAccount(int number){
          //realize function here
          return 0;
       }
}
```

Java and CORBA (40)

- Implementation of server component
  - server class
    - registers implemented object at ORB and Naming Service
    - provides link to implementation class
    - not created by `idltojava`!!

Java and CORBA (41)

```
//Server-Class
import AccountUpdate.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
public class Server {
        public static void main (String args[]){
            try {
                //Creation of Server-ORB
                ORB orb = ORB.init(args, null);
                //create implementation object
                FunctionsImpl fimpl = new FunctionsImpl();
                orb.connect(fimpl);
                //Handle for Name Server
                org.omg.CORBA.Object oref =
                  orb.resolve_initial_references
                  ("NameService");
```

Java and CORBA (42)

```
            NamingContext nc =
              NamingContextHelper.narrow(oref);
            //Bind Object Reference
            NameComponent nco= new NameComponent("AC", "");
            NameComponent path[]={nco};
            nc.rebind(path, fimpl);
            //Await client calls
            java.lang.Object sync = new java.lang.Object();
            synchronized (sync) {
              sync.wait();
            }
        }catch (Exception e) {
            System.err.println("Error: "+e);
        }
    }
}
```

Java and CORBA (43)

- Implementation of client component
  - localizes reference to `Functions` object using naming service
  - reference: CORBA reference which has to be narrowed to correct reference type
  - name used by server: "AC"

Java and CORBA (44)

```
//Client Class
import AccountUpdate.*;
import org.omg.CosNaming.*;
import org.omg.CORBA.*;
public class Client {
       public static void main (String args[]){
           try {
               //Create Client-ORB
               ORB orb = ORB.init(args, null);
               //Create handle for Name Server
               org.omg.CORBA.Object oref =
                 orb.resolve_initial_references
                 ("NameService");
               NamingContext nc =
                 NamingContextHelper.narrow(oref);
```

Java and CORBA (45)

```
            //Find Object Reference
            NameComponent nco= new NameComponent("AC", "");
            NameComponent path[]={nco};
            //Use helper class for casting
            Functions fun =
              FunctionsHelper.narrow(nc.resolve(path));
            //call function
            int result = fun.updateAccount(444);
        }catch (Exception e) {
            System.err.println("Error: "+e);
            e.printStackTrace();
        }
    }
}
```

Java and CORBA (46)

- Compilation of Java classes
  - `javac -d . FunctionsImpl.java Server.java Client.java`
- Call naming service tnameserv
  - `tnameserv -ORBInitialPort 2000`
  - returns Interoperable Object Reference (IOR) and port
    - IOR can be used to localize CORBA object refs

Java and CORBA (47)

- Start client and server
  - `java Server -ORBInitialPort 2000`
  - `java Client -ORBInitialPort 2000`

Jini (1)

- Introducing Jini
  - infrastructure for distributed objects becoming "real"
  - computers are inexpensive, small, and everywhere
- Jini: Goals
  - enabling users to share services and resources over network
  - providing users easy access to resources anywhere on the network while allowing the network location of the user to change
  - simplifying the task of building, maintaining, and altering a network of devices, software, and users

Jini (2)

- Jini: Parts
  - **infrastructure**: A set of components that provides an infrastructure for federating services in a distributed system. JVM, RMI, security, lookup
  - **programming model** that supports and encourages production of reliable distributed services and events. Events, Leasing.
  - **services** that can be made part of a federated Jini system and which offer functionality to any other members. Transactions, user defined (printing, storage, ...)

Jini (3)

- Java and Jini
  - Java Virtual Machine provides consistent processor
  - Object Serialization allows objects to be sent via streams
  - RMI allows objects to call objects on different processors and even transfer objects (proxies and dynamic downloading)
  - Java security model allows fine grained control on what programs can do

Jini (4)

- Jini and distributed services
  - discovery/join: Starting from scratch to join a group
  - lookup
  - leasing: Reserving a service for a set period of time
  - events: sending events across processors
  - transactions: Grouping a set of operations to one atomic operation
  - service interfaces: Either standard or "homegrown"

Jini (5)

- Flow of typical interactions
  - register a service
    - ask for a lookup service (static port, or broadcast)
    - lookup responds
    - service joins the lookup service. Passes remote interface proxy
  - use a service
    - ask for a lookup service
    - lookup responds
    - query for a particular service, get proxy
    - contact service using proxy (no more lookup service)

Jini (6)

- Jini details
  - discovery uses IP broadcast to look for a Lookup service
  - Lookup can handle name value pairs or list attributes
    - service name, for example "bank name"
    - service characteristics, for example printer: 1200 dpi, color, ...
  - Leases can be renewed if leasee needs more time
  - RMI provides activation which helps recovering and starting services

Jini (7)

- Transaction service
  - interface `TransactionManager`: Used for managers of the two-phase commit protocol for top-level transactions
    - abort (long id), commit (long id)
    - create (long lease), join (long id, TransactionParticipant part, long crashCount)
  - interface `TransactionParticipant`: Objects involved in a transaction
    - abort (TransactionManager man, long id)
    - commit(TransactionManager man, long id)
    - prepare(TransactionManager man, long id)

Jini (8)

- Jini and CORBA
  - CORBA defines IDL to create interfaces to objects. Subsequent objects can be implemented in any language. Jini is Java only.
  - CORBA only supports passing primitive types, not objects such as Jini does. Version 2.2 of IIOP is supposed to support this.
  - In general Jini shares capabilities with a CORBA system. They can communicate if needed.

Jini (9)

- Jini and Universal Plug and Play (UPnP)
  - UPnP uses protocols, not passing object interfaces.
  - UPnP addresses many of the same problems and has similar goals as Jini.
  - Perhaps not as ambitious as Jini, UPnP will integrate well with Windows operating systems.

Jini (11)

- Jini and HP CHAI
  - intelligent interaction with embedded devices
  - uses Java within real-time environment ChaiVM
  - goal: Universal Plug-and-Play
  - change core Java to run in real-time
  - similar approach like Jini

Jini (12)

- Jini issues
  - Despite being a very dynamic system, Jini still requires static and well defined interfaces.
  - Jini has many processes to configure and run (rmid, reggie, mahalo)
  - Current lack of Integrated Development Evironments for doing Jini development
  - Performance and bottlenecks. Current implementation can be slow. Lookup service could be a bottleneck.
  - Too ambitious, not ambitious enough?

References

- S. Fischer and A. El Saddik. Open Java: Von den Grundlagen zu den Anwendungen. Springer Verlag, Heidelberg, 1999 (in German)
- H. Bader and W. Huber: Jini. Addison-Wesley, 1999 (in German)
- J.P. Redlich: CORBA2.0, Addison-Wesley, 1996 (in German)
- JavaSoft: http://www.javasoft.com
- Sun: http://www.sun.com/jini
- Jini Community: http://www.jini.org/
- Jini Corner: http://www.artima.com/jini
- Chai: http://www.internetsolutions.enterprise.hp.com/chai/
- UPNP: http://www.UPnP.com
- Inferno: http://www.lucent.com/inferno/