Object interaction using RMI
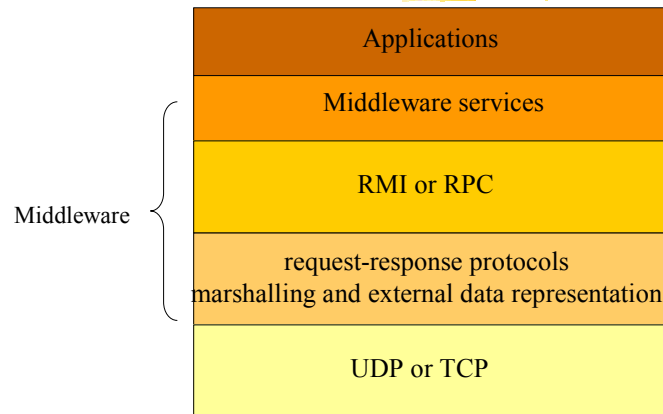
INF 5040 høst 2006

Lecturer: Frank Eliassen

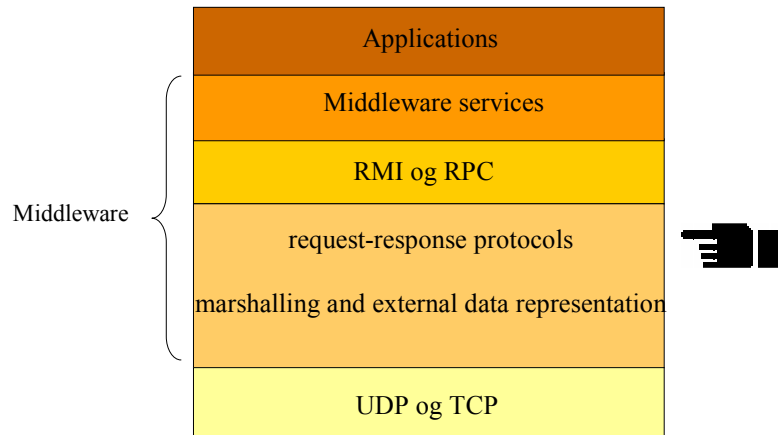## Plan

- Principles for realising remote methods invocations (RMI)
- Objekt-servers
- Multi-threaded object servers
- CORBA RMI
- Java RMI

Layering of middleware

Applications
Middleware services
RMI or RPC
request-response protocols marshalling and external data representation
UDP or TCP

Middleware

Plan

➢ Principles for realising remote methods invocations (RMI)
➢ Objekt-servers
➢ Multi-threaded object servers
➢ CORBA RMI
➢ Java RMI

Layering of middleware

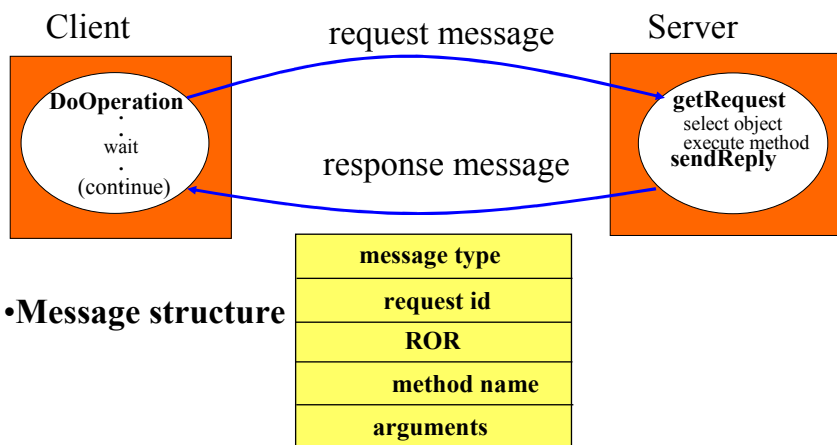| |
|---|
| Applications |
| Middleware services |
| RMI og RPC |
| request-response protocols |
| marshalling and external data representation |
| UDP og TCP |

Middleware

Client-server communication
RPC/RMI exchange protocols

➢ Based on UDP or TCP

Client   request message   Server

DoOperation
·
wait
·
(continue)

getRequest
select object
execute method
sendReply

response message

•**Message structure**

message type
request id
ROR
method name
arguments

INF 5040 høst 2007

3

Failure model for RPC/RMI exchange protocols over UDP (I)

➢ Protocol can be exposed to
  ▪ omission failure
  ▪ process crash failure
  ▪ message order not guaranteed
➢ Failure is detetced as *timeout* in the primitive DoOperation:
  ▪ recovery actions depend on the offered delivery guarantee

Failure model for RPC/RMI exchange protocols over UDP (II)

➢ Timeout DoOperation
  ▪ Send request message repeatedly until
    – response is avaialble, or
    – assume server has failed (max no of retrans.)
➢ Duplicate request messages
  ▪ occur when request message is sent more than once
  ▪ can lead to operations being executed more than once for the same requestl
  ▪ => must be able to filter duplicate requests (role of request id)
➢ Lost response messages
  ▪ server has already sent response message when it recieves a duplicate request message
  ▪ => may have to execute the operation again to get the right response
    – OK for operations that are "idempotent"

Failure model for RPC/RMI exchange protocols over UDP(III)

- Logs (histories):
  - used by servers offering operations that are not "idempotent"
  - contains response messages already sent
- Disadvantage of logs:
  - storage requirement
- if a client is allowed to do only one request at a time to the same server, the log can be limited in size (bounded by the number of concurrent clients)
- at reception of the next request message from the same client the server may delete the last response message for that client from the log
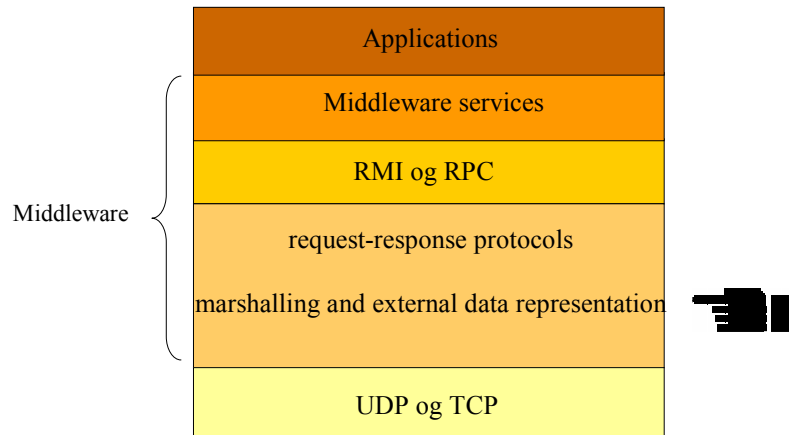
RPC/RMI exchange protocols over TCP

- UDP has limited packet size
  - => need for fragmentation/defragmentation protocols
- RPC/RMI exchange protocols over TCP avoids this problem
  - TCP ensures reliable delivery of byte streams
- Problem:
  - much overhead if the connection has to be created at each request
    => need for optimization (leave connection open for later reuse)
  - upper bound on number of concurrent TCP-connections could cause problems

Layering of middleware

Applications
Middleware services
RMI og RPC
request-response protocols
marshalling and external data representation
UDP og TCP

Middleware { (brace spanning Middleware services, RMI og RPC, request-response protocols, marshalling and external data representation)

Marshalling
External data representation

- ➢ "marshalling"
  - serialize data structures to messages (sequence of data values)
  - translate sequence of data values to an external representation
- ➢ "unmarshalling"
  - inverse of "marshalling"
- ➢ External data representation (representation "on the wire")
  - a representation of data during transfer of the message
  - Sun XDR (representation of most used data types)
  - ASN.1/BER (ISO standard, based on "type-tags", open)
  - NDR (used in DCE RPC)
  - CDR (used in CORBA RMI, binary layout of IDL types)
  - Java Object Serialization (JOS)
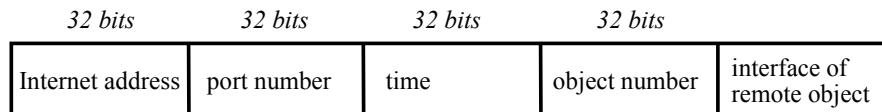 - XML (used i SOAP: "RMI" protocol for Web Services)

6

INF 5040 høst 2007

Marshalling
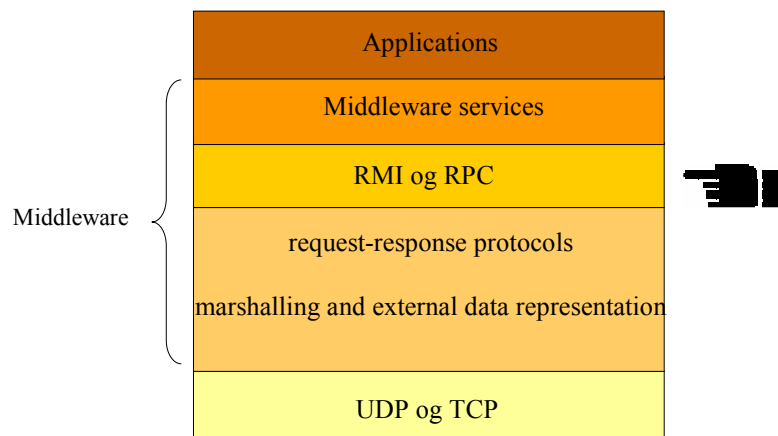## Object-references

➢ Remote-object-reference (ROR)
- Identifier for remote objects that is valid in a distributed system
- Must be generated in a way that ensures uniqueness over timie and space (=> a ROR can not be reused)
- Example:

32 bits	32 bits	32 bits	32 bits	
Internet address	port number	time	object number	interface of remote object

Layering of middleware

Applications
Middleware services
RMI og RPC
request-response protocols
marshalling and external data representation
UDP og TCP

Middleware

Classification of RPC/RMI exchange protocols

- Classification after (Spector, 1982):
  - basis for implementing different types of RMI and RPC
- Request (R) protocol
  - Only Request-message. No response message from server
  - No confirmation that operation has been performed
- Request-Reply (RR) protocol
  - Reply-message confirms that the Request-message has been performed
  - A new request from the client confirms reception of Reply-message
- Request-Reply-Acknowledge (RRA) protocol
  - separate message from client to confirm reception of Reply-message
  - tolerates loss of Ack-message
    - Ack with a given *request id* confirms all lower requests ids

RPC/RMI invocation semantics

- Reliability semantics of RPC/RMI under partial failures

	Fault tolerance measures		Invocation semantics
Retransmit Request message	Duplication filtering	Re-excute method *or* retransmit Reply	
No (R)	____	____	Maybe
Yes (RR)	No	Re-execute method	At-least-once
yes (RR)	Yes	Retransmit Reply	At-most-once

8

RMI invocation semantics in object and RPC middlewares

- ➤ RMI in CORBA and Java have "at-most-once" invocation semantics under partial failures
  - ▪ referred to as synchronous requests
- ➤ CORBA allows other forms of synchronization that provides other invocation semantics
  - ▪ One-way operations: maybe-semantics
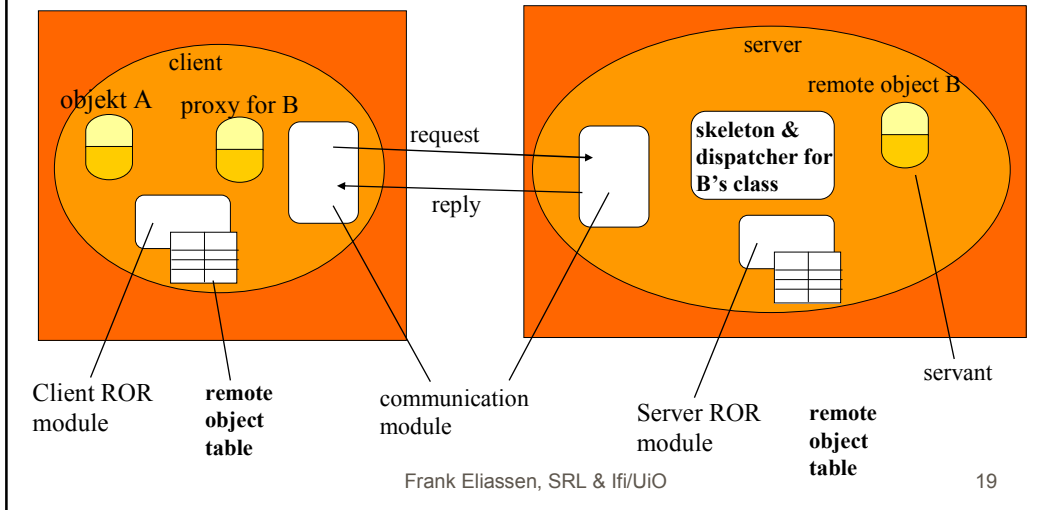- ➤ SUN RPC: at-least-once semantics

Implementation of RMI

- ➤ Three main tasks :
- ➤ Interface processing
  - ▪ Integration of the RMI mechanism into a programming language.
  - ▪ Basis for realizing access transparency
- ➤ Communication
  - ▪ message exchange (request-reply protocol)
- ➤ Object location, binding and activation
  - ▪ Locate a suitable server process that hosts the remote object and bind to the server
  - ▪ Activate an object-implementation
  - ▪ Basis for realizing location transparency

Objects and modules involved in RMI

Client ROR module	Server ROR module
remote object table	**remote object table**
communication module	

server

client

objekt A · proxy for B

remote object B

skeleton & dispatcher for B's class

request

reply

servant

Elements of the RMI software (I)

➢ RMI interface processing: Client proxy
  - Local "proxy" object for each remote object a client holds a ROR ("stand-in" for remote object).
  - The class of the proxy-object has the same interface as the class of the remote object. Can perform type checking on arguments
  - Performs marshalling of requests and unmarshalling or responses
  - Transmits request-messages to the server and receive response messages.

Elements of the RMI software (II)

➢ RMI interface processing: Dispatcher

- A server has one dispatcher for each class representing a remote object.
- Receives requests messages
- Uses *method id* in the request message to select the appropriate method in the skeleton and passes on the request message
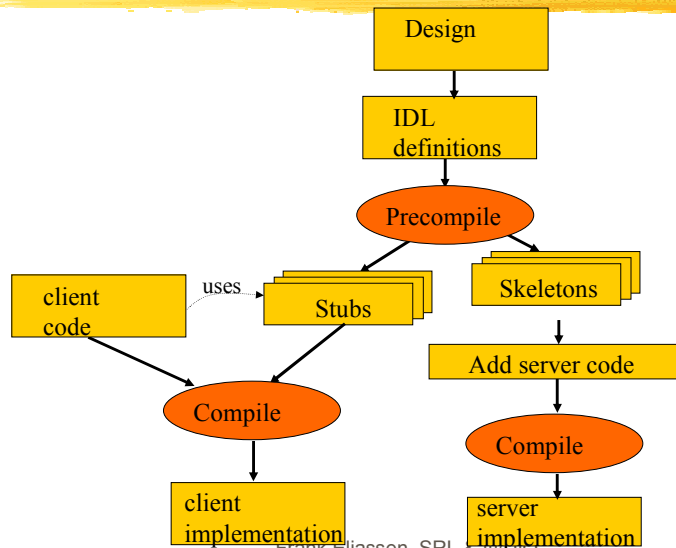
Elements of the RMI software (III)

➢ RMI interface processing: Skeleton

- A server has one skeleton for each class representing a remote object
- Implements the methods in the remote interface
- A skeleton method unmarshals the arguments in the request message and invokes the corresponding method in the servant.
- It waits for the invocation to complete and then marshals the result, together with any exceptions, in a reply message to the sending proxy's method.

Generation of proxies, dispatchers and skeletons

```
                        Design
                          │
                          ▼
                        IDL
                      definitions
                          │
                          ▼
                     Precompile
                     ╱         ╲
                    ╱           ╲
   client         Stubs        Skeletons
   code    uses                   │
     │      ┄┄┄┄┄              ▼
     │        │            Add server code
     │        │                 │
     ▼        ▼                 ▼
      Compile                Compile
         │                      │
         ▼                      ▼
      client                 server
   implementation         implementation
```

Server and client programs

➢ Server program contains
  ▪ the classes for the dispatchers and skeletons
  ▪ the implementation classes of all the servants that it supports
  ▪ an initialization section: creates and initializes at least one servant
    – additional servants may be created in response to client requests
  ▪ register zero or more servants with a *binder*
  ▪ potentially one or more *factory methods* that allow clients to request creation of additional servants
➢ Client program contains
  ▪ the classes and proxies for all the remote objects that it will invoke

RMI binding, object location, and activation

➢ Binding in RMI
- corresponds to mapping a symbolic object name to an ROR
- performed by a "binder" (name service or similar)

➢ Object location in RMI
- corresponds to mapping an ROR to a communication identifier

➢ Activation in RMI
- corresponds to creating an active object from a corresponding passive object (e.g., on request). Performed by an *activator*
  - register passive objects that are available for activation
  - activate server processes (and activate remote object within them)

Object location

➢ Object location : corresponds to mapping an ROR to a communication identifier.
- integrated in ROR
  - Address can be extracted directly from the object reference
- location service
  - A name server is used by the client proxy at each request
- cache/broadcast
  - Each client has cache of bindings (ROR, comm. identifier)
  - If ROR not in cache, perform broadcast with ROR
  - Servers that host the object respond with comm.identifier
- forward pointers or address hint (to e.g., location service)
  - Used at object migration
- Combinations of the above

Plan

> Principles for realising remote methods invocations (RMI)

> Object-servers

> Multi-threaded object servers

> CORBA RMI

> Java RMI

Object-server:
Server tailored to support distributed objects

> Services realized as objects that the server encapsulates

- Services can be added or removed by creating and removing remote objects

> Object servers act as places where objects can live

> Object servers activate remote objects on demand

- Several ways to activate an object

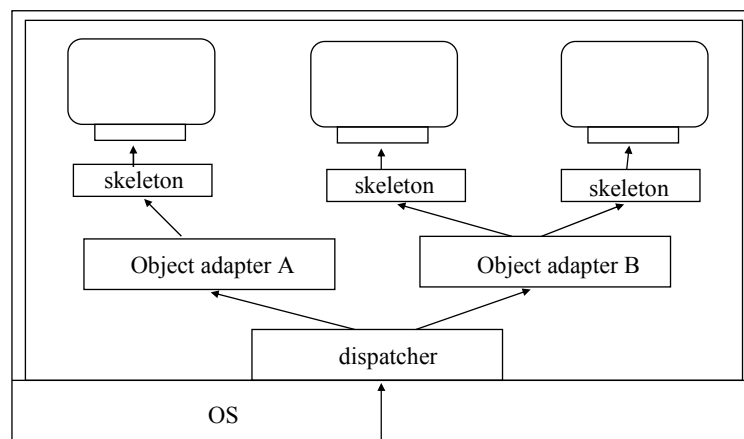# Object servers must assign processing resources to objects when they are activated

- When an object is activated, which processing resoures should be assigned to the implementation?
- Activation policy
  - A particular way of activating an object
  - Different dimensions
    - How to translate between ROR and local implementation?
    - Should the server be single-threaded or multi-threaded?
    - If multi-threaded, how to assign threads to objects and requests? One thread per object? One per request?
    - Transient vs persistent objects, etc
- No single activation policy that fits all needs
  - Object servers should support several concurrent activation policies
  - Objects can be grouped according to which activation policy they are governed by

Frank Eliassen, SRL & Ifi/UiO 29

Organization of object servers that support different activation policies

- **Object-adapter:** software that implements a specific activation policy (supported by CORBA Portable Object Adapter (POA))

Frank Eliassen, SRL & Ifi/UiO 30

Plan

- Principles for realising remote methods invocations (RMI)
- Objekt-servers
- Multi-threaded object servers
- CORBA RMI
- Java RMI

Why use several processes or threads in distributed systems?

- Better performance
- Better exploitation of resources

Object-servers must assign processing resources to objects when objects are activated

➢ When an object is activated, which processing resources should be assigned to its implementation?

- Create a new process or thread?
- Are there several ways this can be done?
- Is there a best way (cf. activation policies)?

Threads are cheaper to manage than processes

➢ execution environment (address room) expensive to create

➢ threads are much cheaper to create and terminate

➢ threads can share runtime environment
(e.g., global variables, open files, timers, signals, ….)

➢ BUT: processes define *protection domains*
- a thread in a process can (normally) not access data and resources in an other process

Multi-thread server

Request

In-Out
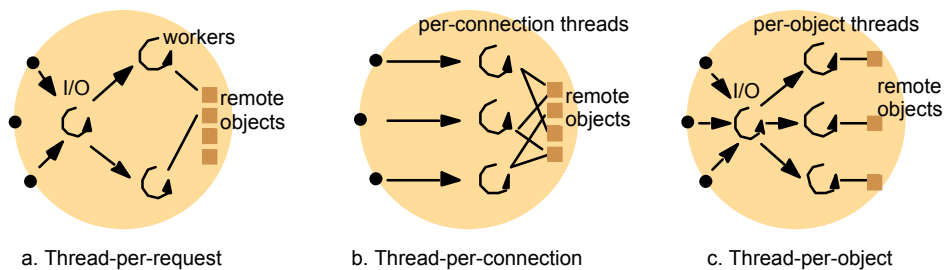8 ms delay
2 ms w/cache ave

2 ms processing
2.5 ms w/cache
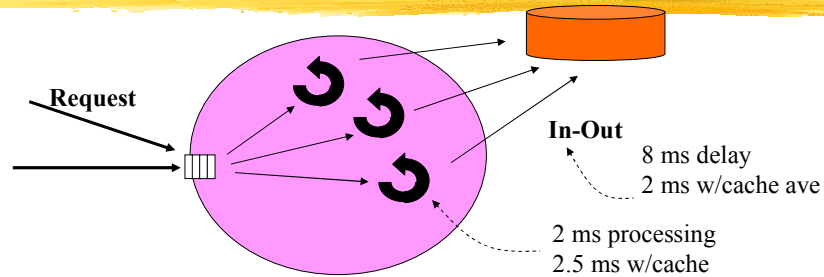
	# processors	# threads	Max. calls/sec
no disk caching	1	1	100
no disk caching	1	2	125
disk caching	1	2	400
disk caching	2	2	444
disk caching	2	3	500

Alternative threading-policies for object activation

workers
I/O
remote objects

per-connection threads
remote objects

per-object threads
I/O
remote objects

a. Thread-per-request

b. Thread-per-connection

c. Thread-per-object

Plan

> Principles for realising remote methods invocations (RMI)

> Objekt-servers

> CORBA RMI

> Java RMI

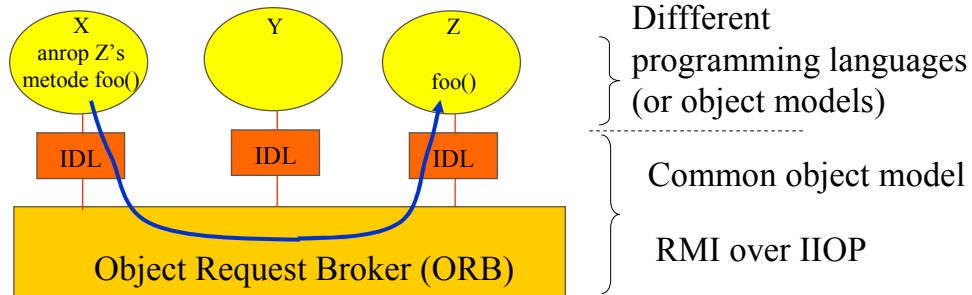> Multi-threaded object servers

---

CORBA middleware

> Offers mechanisms that allow objects to invoke remote methods and receive responses in a transparent way
  - location transparency
  - access transparency

> The core of the architecture is the Object Request Broker (ORB)

> Specification developed by members of the Object Management Group (www.omg.org)

CORBA RMI

Clients may invoke methods of remote objects without worrying about:
object location, programming language,
operating system platform, communcation
protocols or hardware.

X
anrop Z's
metode foo()

Y

Z
foo()

IDL   IDL   IDL

Object Request Broker (ORB)

Diffferent
programming languages
(or object models)

Common object model

RMI over IIOP

Layering of middleware

| Applications |
| CORBA Middleware services |
| RMI og RPC |
| request-response protocols |
| marshalling and external data representation |
| UDP og TCP |

Middleware

20

CORBA Services

Application objects

Domain
interfaces

Business objects
E-Commerce
Financial Domain
Healthcare
Telecomm
....

CORBA
facilities

Compound doc
Help
Printing
....

Object Request Broker (ORB)

Naming
Lifecycle
Persistence
Events
Transactions
Trading
Time

Concurrency
Relationships
Externalization
Querying
Licensing
Security
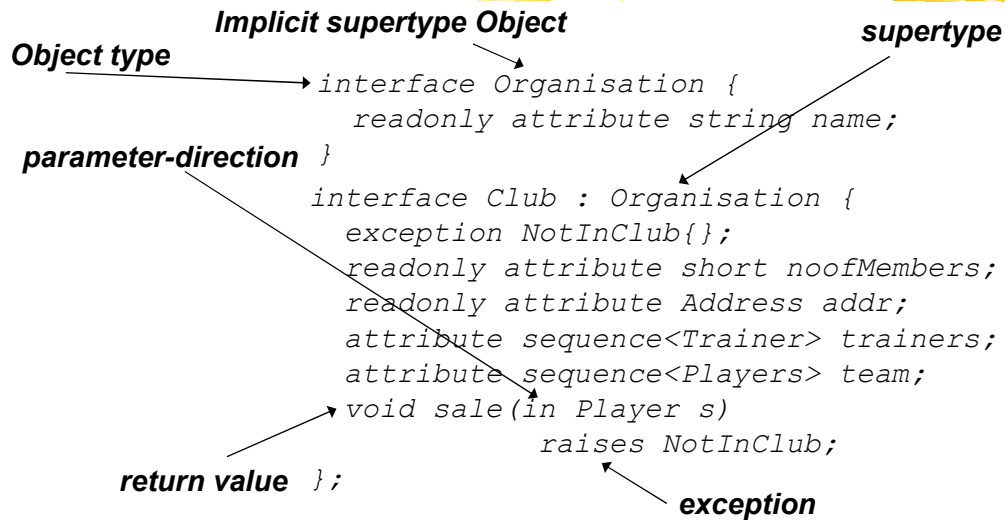Properties

Object services

Frank Eliassen, SRL & Ifi/UiO

41

CORBA IDL

➢ Language for specifying CORBA object types
➢ Can express all concepts in the CORBA object model
➢ OMG/IDL is
  ▪ not dependent on a specific programming language
  ▪ syntactically oriented towards C++
  ▪ not computationally complete
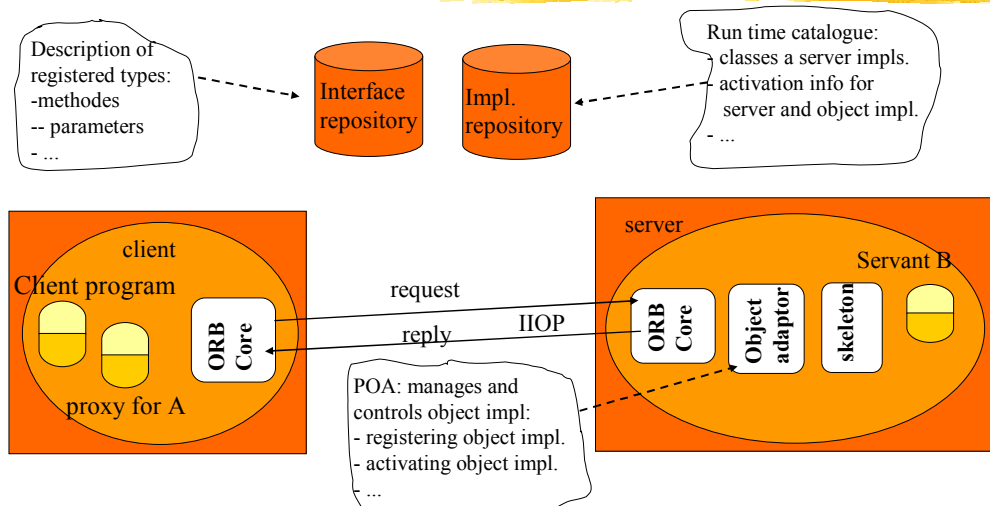➢ Different bindings to programming languages available

Frank Eliassen, SRL & Ifi/UiO

42

CORBA object model and IDL

Implicit supertype Object

Object type

supertype

parameter-direction

```
interface Organisation {
    readonly attribute string name;
}
interface Club : Organisation {
    exception NotInClub{};
    readonly attribute short noofMembers;
    readonly attribute Address addr;
    attribute sequence<Trainer> trainers;
    attribute sequence<Players> team;
    void sale(in Player s)
                  raises NotInClub;
};
```

return value

exception

Frank Eliassen, SRL & Ifi/UiO   43

CORBA architecture

Description of registered types:
-methodes
-- parameters
- ...

Interface repository

Impl. repository

Run time catalogue:
- classes a server impls.
- activation info for server and object impl.
- ...

client

Client program

server

Servant B

request

reply

IIOP

ORB Core

ORB Core

Object adaptor

skeleton

proxy for A

POA: manages and controls object impl:
- registering object impl.
- activating object impl.
- ...
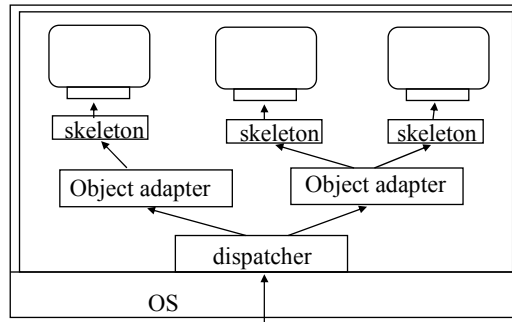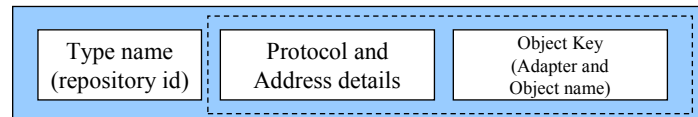
Frank Eliassen, SRL & Ifi/UiO   44

CORBA ROR format (IOR) reflects the organization of object servers


CORBA Interoperable Object Reference (IOR)

Type name (repository id)	Protocol and Address details	Object Key (Adapter and Object name)

CORBA RMI Object Location (I)

➢ Object location in RMI corresponds to mapping object references (ROR) to "servants"
- servant: implementation of one or more CORBA objects

➢ ROR in CORBA: Interoperable Object Reference (IOR)

➢ Location process:
- based on information encoded in the object reference

CORBA Interoperable Object Reference (IOR)

Type name (repository id)	Protocol and Address details	Object Key (Adapter and Object name)
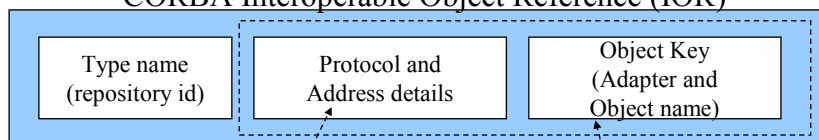
IIOP: host name/port no   **Proprietary format (of the ORB that created the IOR)**

CORBA RMI Object Location (II)

➤ Transient IOR
- Valid only as long the corresponding server process is available
- After the server process has terminated, the IOR will never be valid again
- The location of the server process is encoded into the IOR of the object.

➤ Persistent IOR
- continue to function (denote same CORBA object) even when the server process terminates and later starts up again Frank Eliassen, SRL & Ifi/UiO 47

Plan

➤ Principles for realising remote methods invocations (RMI)

➤ Objekt-servers

➤ Multi-threaded object servers

➤ CORBA RMI

➤ Java RMI

Frank Eliassen, SRL & Ifi/UiO 48

INF 5040 høst 2007

24

➤ Java Remote Method Invocation (RMI)

Java RMI

➤ Remote Method Invocation (RMI) supports communication between different Java Virtual Machines (VM), and possibly over a network

➤ Provides tight integration with Java

➤ Minimizes changes in the Java language/VM

➤ Works for homogeneous environments (Java)

➤ Clients can be implemented as *applet* or *application*

25

Java Object Model

➢ Interfaces and Remote Objects
➢ Classes
➢ Attributes
➢ Operations/methods
➢ Exceptions
➢ Inheritance

Java interfaces to remote objects

➢ Based on the ordinary Java interface concept
➢ RMI does not have a separate language (IDL) for defining remote interfaces
➢ Pre-defined interface `Remote`
➢ All RMI communication is based on interfaces that extends java.rmi.Remote
➢ Remote classes implement java.rmi.Remote
➢ Remote objects are instances of remote class

Java remote interface: Example

interface name   ***declares the Team interface as "remote"***

```
interface Team extends Remote {
public:
   String nama()throws RemoteException;
   Trainer[] trained_by() throws RemoteException;
   Club club() throws RemoteException;
   Player[] player() throws RemoteException;
   void chooseKeeper(Date d)  throws RemoteException;
   void print() throws RemoteException;
};
```

remote operation

Java RMI parameter passing

➢ Atomic types transferred *by value*
➢ Remote objects transferred *by reference*
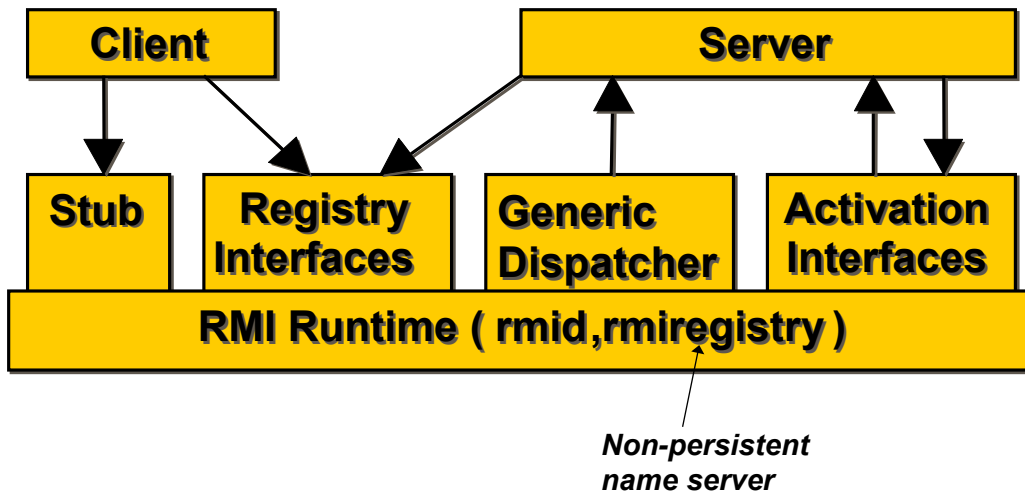➢ None-remote objects transferred *by value*

```
class Address {
  public:
  String street;
  String zipcode;
  String town;
};
interface Club extends Organisation, Remote {
  public:
  Address addr() throws RemoteException;
  ...
};
```

Returns a copy of the Address-object

27

Architecture of Java RMI

Client		Server

Stub	Registry Interfaces	Generic Dispatcher	Activation Interfaces

RMI Runtime (rmid,rmiregistry)

Non-persistent name server

Summary

➢ Implementation of RMI
- proxies, skeletons, dispatcher
- interface processing, binding, location, activation

➢ Invocation semantics (under partial failure)
- maybe, at-least-once, at-most-once
- Reliability of RMI is at best "at-most-once"

➢ Multi-threaded servers
- can in some cases be used to increase the throughput (method calls/time unit) if, e.g., I/O is the bottleneck

➢ Principles of CORBA
- Clients may invoke methods of remote objects without worrying about: object location, programming language, operating system platform, communcation protocols or hardware.

➢ Principles of Java RMI
- Similar to CORBA but limited to a Java environment

Extra slides

CORBA
Portable object adapter (POA)

- ➢ Enables portability of object implementations across differnt ORBs
- ➢ Supports light weight transient object and persistent object identifiers (e.g., for objects stored in databases)
- ➢ Supports transparent object activations
- ➢ Extensible mechanism for activation policies
- ➢ Several POAs in one single server

CORBA klient	CORBA tjener
ORB ?	POA servants
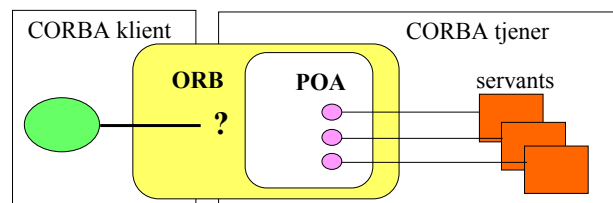
Locating transient IORs over IIOP
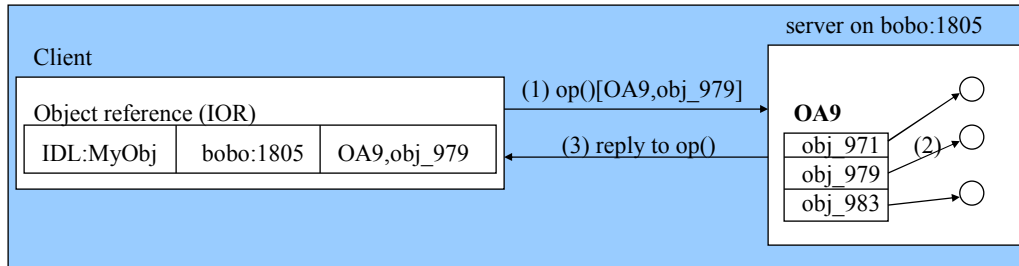
server on bobo:1805

Client

Object reference (IOR)

IDL:MyObj	bobo:1805	OA9,obj_979

(1) op()[OA9,obj_979]

(3) reply to op()

OA9

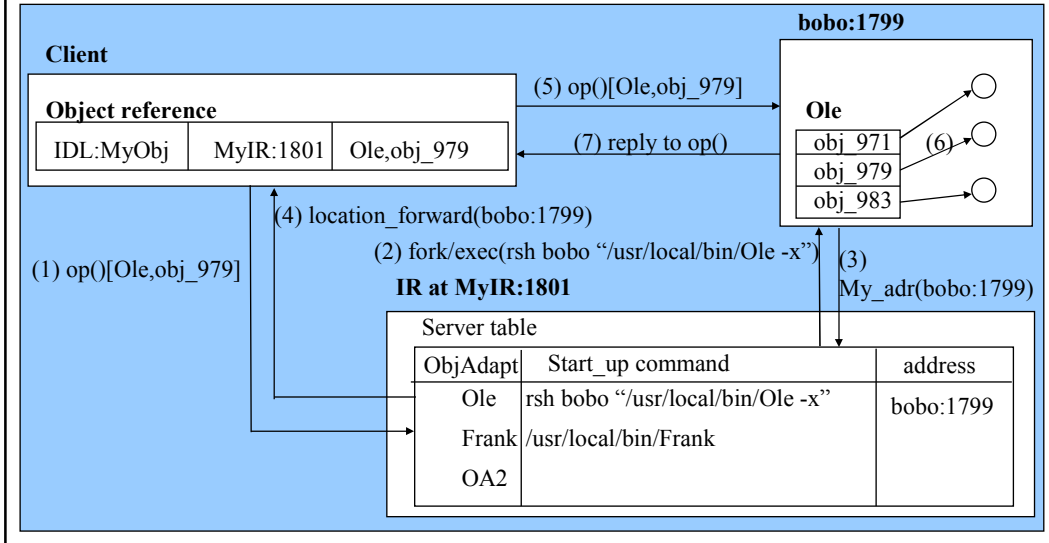| obj_971 |
|---------|
| obj_979 |
| obj_983 |

(2)

Locating persistent IOR over IIOP

- ➢ uses Implementation Repository (IR) as activator.
- ➢ IR:
  - ▪ handles process/thread-creation and -termination, a.m.
  - ▪ is not portable (specific to an ORB implementation)
  - ▪ not standardized
    - – tailored to specific environments
    - – not possible to write specifications that cover all environments
    - – communication between an ORB and its IR is not visible to the client
  - ▪ Object migration, scaling, performance, and fault tolerance are dependent on IR
  - ▪ Implemented usually as a process at a fixed address
    - – a set of host machines that is configured under the same IR is denoted a "location domain"

Locating persistent IOR over IIOP

bobo:1799

Client

Object reference

IDL:MyObj	MyIR:1801	Ole,obj_979

(5) op()[Ole,obj_979]

(7) reply to op()

Ole

| obj_971 |
| obj_979 |
| obj_983 |

(6)

(4) location_forward(bobo:1799)

(2) fork/exec(rsh bobo "/usr/local/bin/Ole -x")

(1) op()[Ole,obj_979]

(3) My_adr(bobo:1799)

IR at MyIR:1801

Server table

ObjAdapt	Start_up command	address
Ole	rsh bobo "/usr/local/bin/Ole -x"	bobo:1799
Frank	/usr/local/bin/Frank	
OA2		

Java RMI development process

Define remote interface

Implement the interface

.java

javac

.class

rmic

Server class

Implement client → uses → Client stub

.java

javac

.class

Start client

Start RMI Registry

Start server objects

Register remote objects

Client

Server

Frank Eliassen, SRL & Ifi/UiO

62