Object interaction using RMI

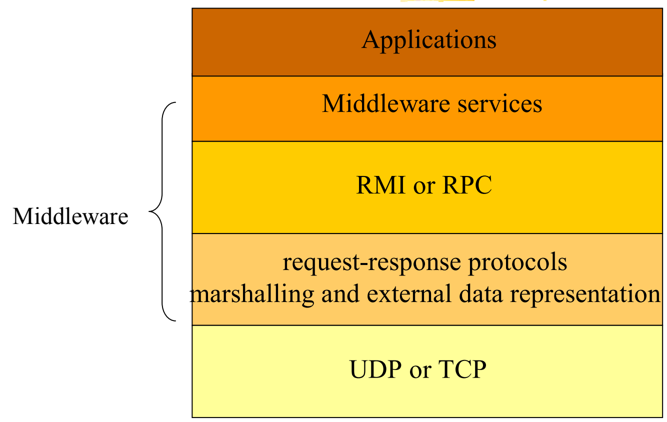
INF 5040/9040 autumn 2008

Lecturer: Frank Eliassen

Plan

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

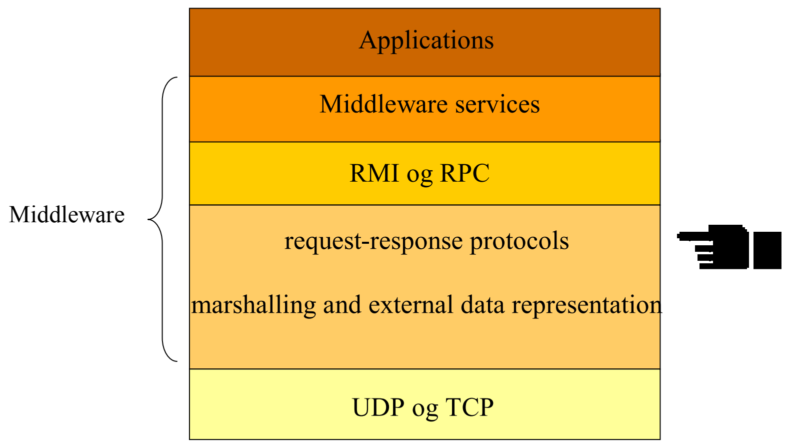
Layering of middleware



Plan

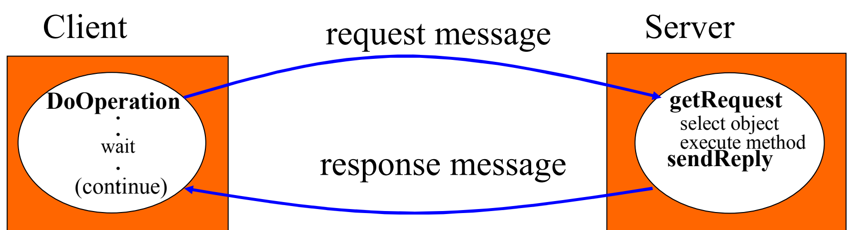
- Principles for realising remote methods invocations (RMI)
- Object-servers
- Multi-threaded object servers
- CORBA RMI
- Java RMI

Layering of middleware



Client-server communication
RPC/RMI exchange protocols

- Usually based on UDP or TCP



•**Message structure**

message type
request id
ROR
method name
arguments

Failure model for RPC/RMI protocols

- Protocol can be exposed to
  - omission failure
  - process crash failure
  - message order not guaranteed (UDP)
- Failure is detected as *timeout* in the primitive DoOperation:
  - recovery actions depend on the offered delivery guarantee

Failure and recovery for RPC/RMI protocols (I)

- 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 and recovery for RPC/RMI protocols (II)

- 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 protocols: TCP vs UDP

- UDP has limited packet size
  - => need for fragmentation/defragmentation protocols
- RPC/RMI 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

Classification of RPC/RMI 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

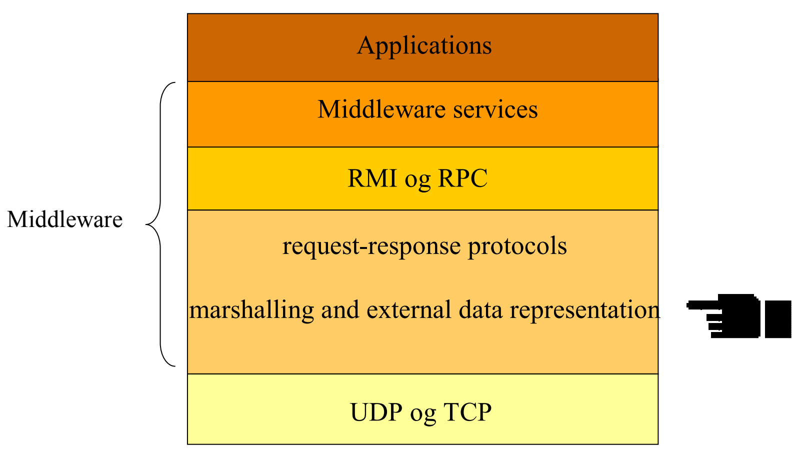
- Reliability semantics of RPC/RMI under partial failures

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

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

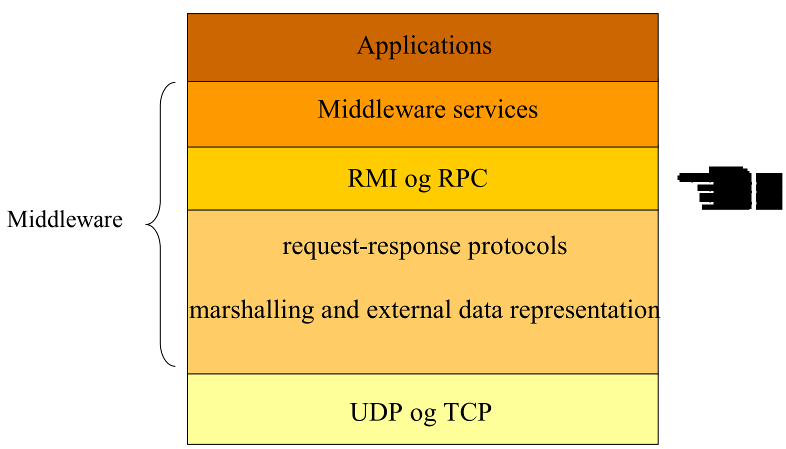
Layering of middleware



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)

Layering of middleware



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

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 of responses
  - Transmits request-messages to the server and receive response messages.

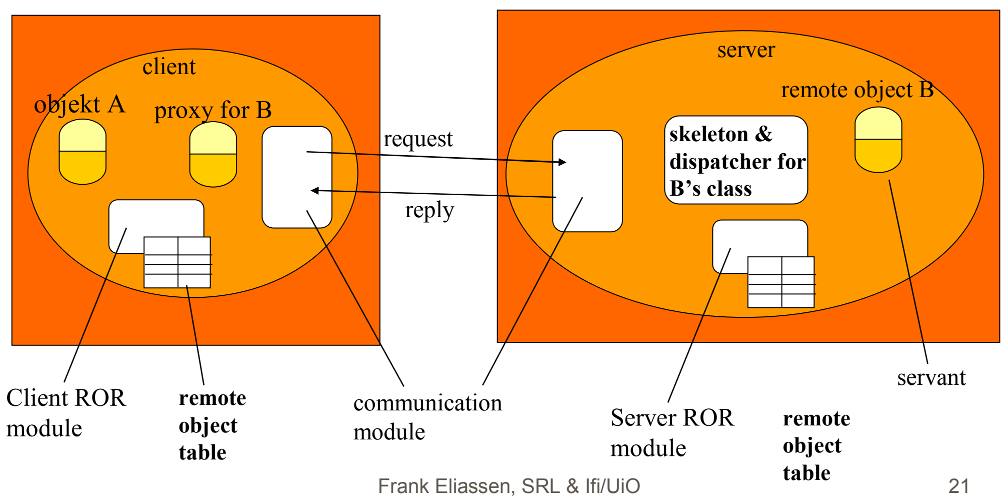
Elements of the RMI software (II)

- 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)

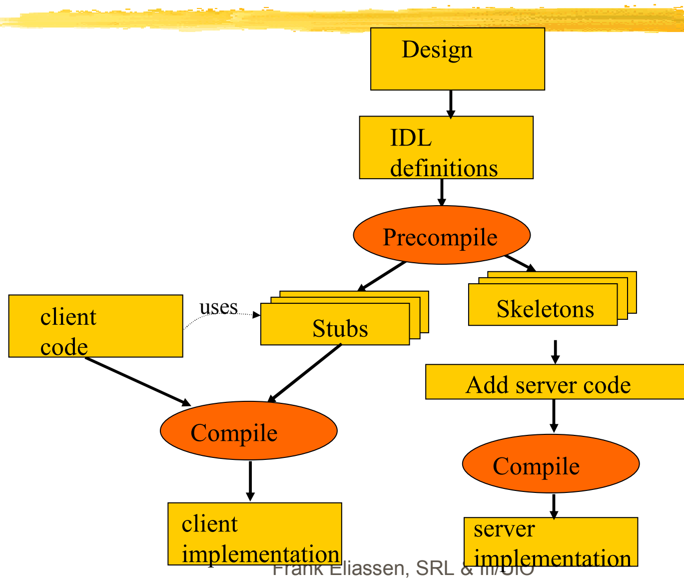
- RMI interface processing: Skeleton
  - A server has one skeleton for each class representing a remote object
  - Implements the methods in the remote interface
  - A skeleton method unmarshals the arguments in the request message and invokes the corresponding method in the remote object.
  - It waits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

Objects and modules involved in RMI



Frank Eliassen, SRL & Ifi/UiO 21

Generation of proxies, dispatchers and skeletons

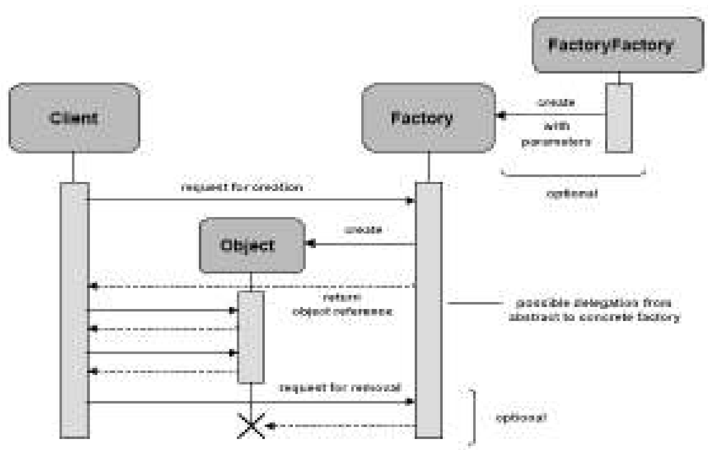


Frank Eliassen, SRL & Ifi/UiO 22

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

Factory pattern



source: S. Krakowiak, Middleware Architecture with Patterns and Frameworks, http://sardes.inrialpes.fr/~krakowia/MW-Book/

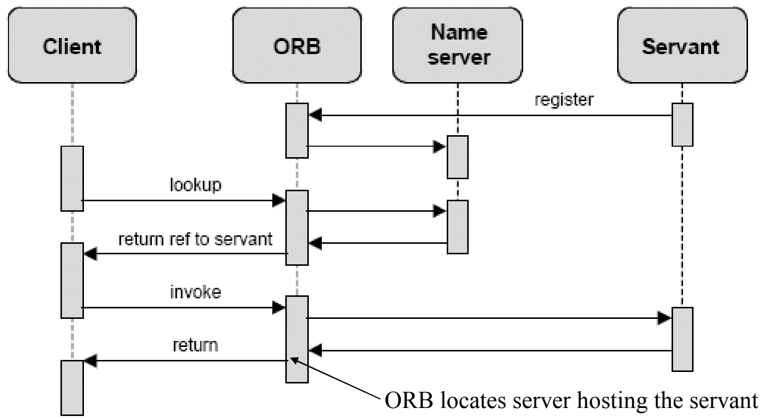
RMI name resolution, binding, and activation

- Name resolution
  - corresponds to mapping a symbolic object name to an ROR
  - performed by a name service (or similar)
- Binding in RMI
  - corresponds to locating the server holding a remote object based on the ROR of the object and placing a proxy in the client process's address space
- 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)

Locating the server of a remote object

- Corresponds to mapping an ROR to a communication identifier.
  - integrated in ROR
    - Address can be extracted directly from the object reference
  - location service
    - A location service 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

Remote method invocation



source: S. Krakowiak, Middleware Architecture with Patterns and Frameworks, http://sardes.inrialpes.fr/~krakowia/MW-Book/

Implicit and explicit binding

```
Distr_object* obj_ref;              // Declare a system wide object reference
obj_ref = lookup(obj_name);         // Initialize the reference to a distrb. obj
obj_ref->do_something();            // Implicit bind and invoke method
```

```
Distr_object* obj_ref;              // Declare a system wide object reference
Local_object* obj_ptr               // Declare a pointer to a local object
obj_ref = lookup(obj_name);         // Initialize the reference to a distrb. obj
obj_ptr = bind(obj_ref);            // Explicitly bind and get pointer to local proxy
obj_ptr->do_something();            // Invoke a method on the local proxy
```

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

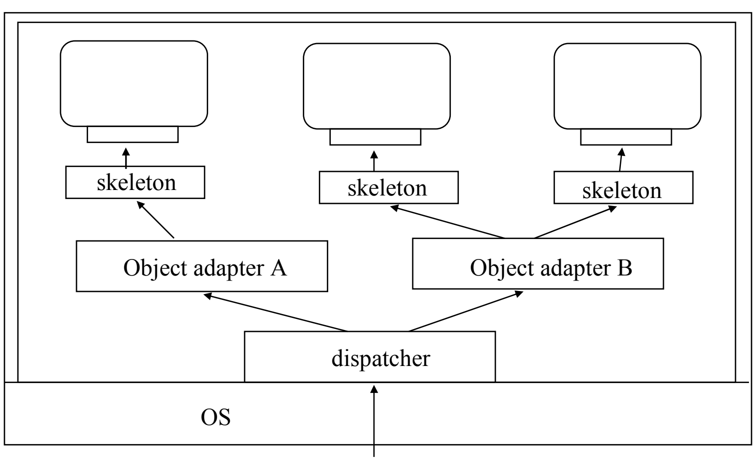
- Services realized as objects that the server encapsulates
  - Services can be added or removed by creating and removing remote objects
- Object servers act as places where objects can live
- Object servers activate remote objects on demand
  - Several ways to activate an object

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

- When an object is activated, which processing resources 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

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))



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:

Internet address	port number	adapter name	object key	interface of remote object

Plan

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

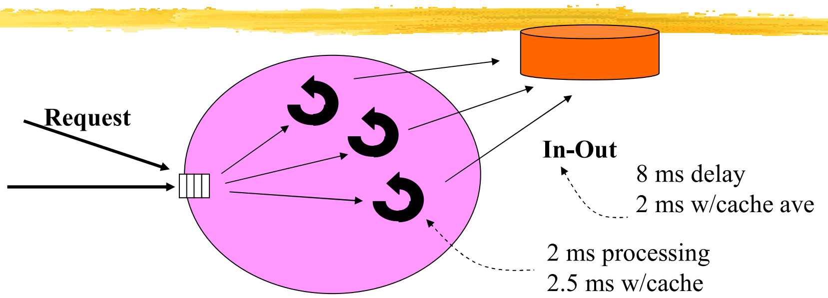
Why use several processes or threads in distributed systems?

- Better performance
- Better exploitation of resources

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

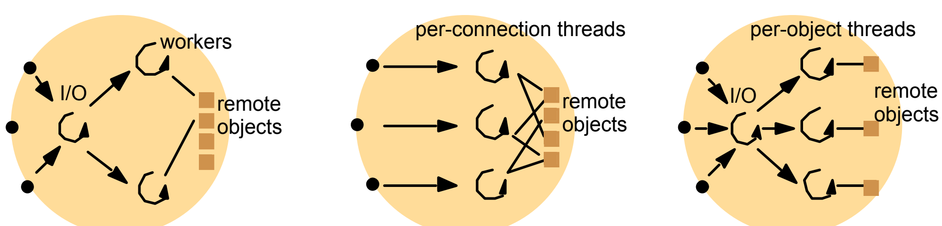
- When an object is activated, which processing resources should be assigned to its implementation?
  - Create a new process or thread?
  - Are there several ways this can be done?
  - Is there a best way (cf. activation policies)?

Multi-thread server



	# 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



a. Thread-per-request b. Thread-per-connection c. Thread-per-object

Plan

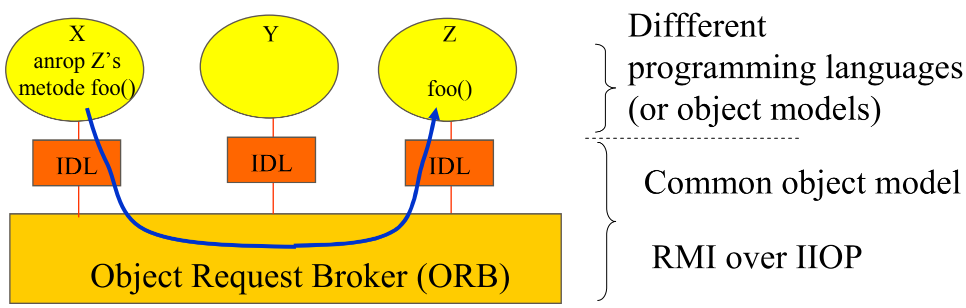
- Principles for realising remote methods invocations (RMI)
- Object-servers
- Multi-threaded object servers
- CORBA RMI
- Java RMI

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.



CORBA IDL

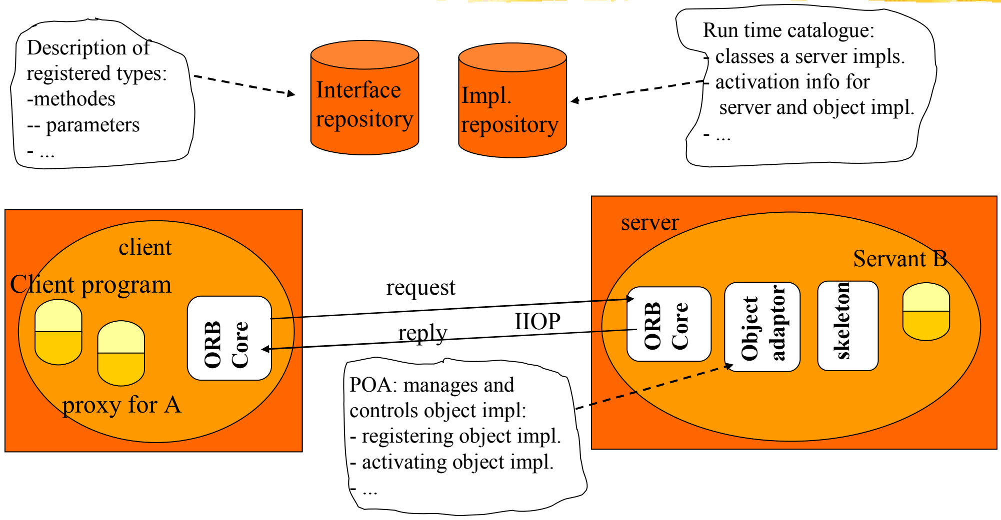
- Language for specifying CORBA object types
- Can express all concepts in the CORBA object model
- OMG/IDL is
  - not dependent on a specific programming language
  - syntactically oriented towards C++
  - not computationally complete
- Different bindings to programming languages available

CORBA object model and IDL

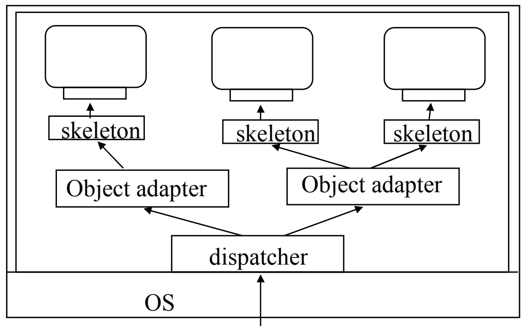
Object type — *Implicit supertype Object* — *supertype* — *parameter-direction* — *return value* — *exception*

```
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;
};
```

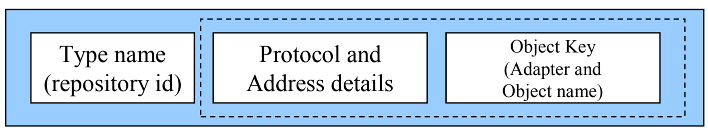

CORBA architecture



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



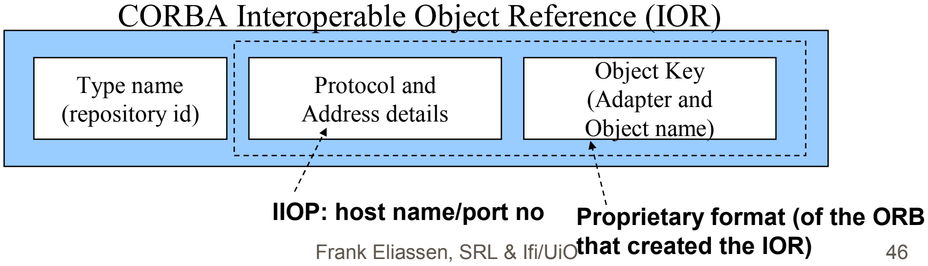
CORBA Interoperable Object Reference (IOR)



CORBA RMI binding (I)

- Binding 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

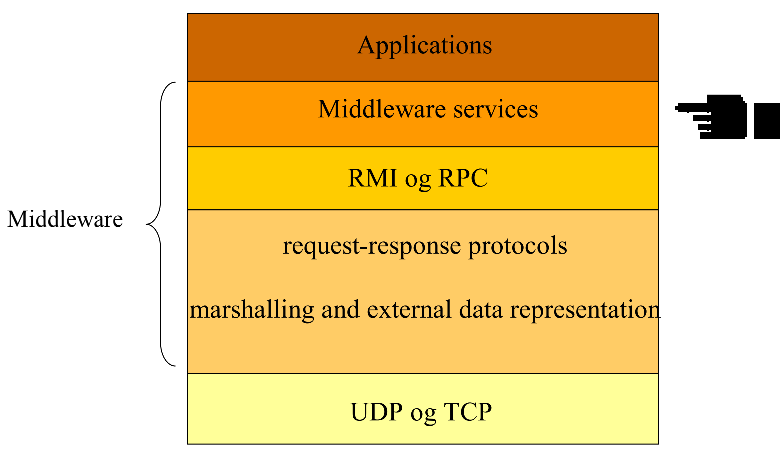
CORBA Interoperable Object Reference (IOR)



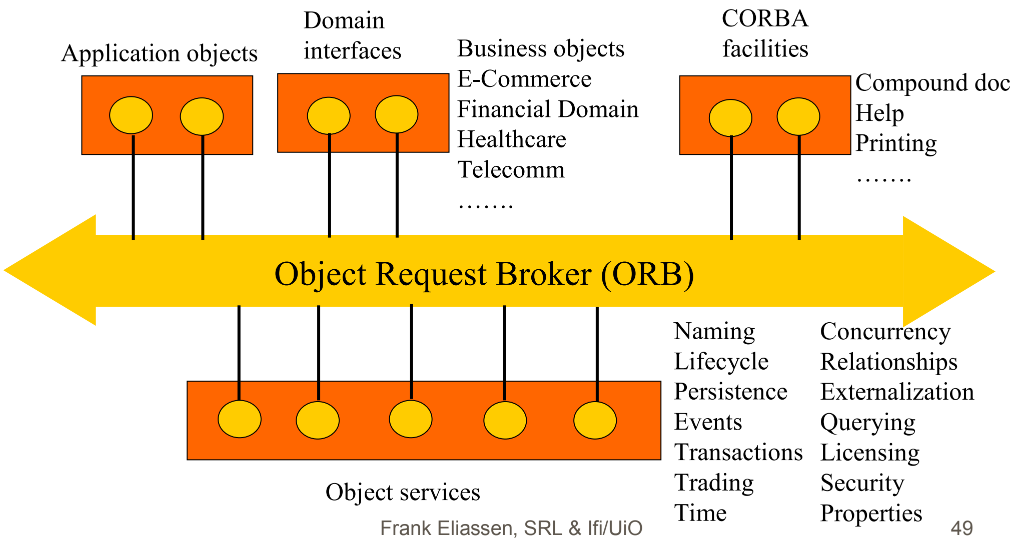
CORBA RMI binding (II)

- 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
  - An *activator* (implementation repository) can automatically start a server process when a client is using a persistent object reference and terminate the server again after a certain idle time
  - The location of the *activator* is encoded into the IOR of the object IOR. The actual location of the server process must be resolved via the activator.
  - A persistent POA must be registered at an activator.
  - A persistent POA creates persistent IORs and knows how to activate persistent objects that it manages

Layering of middleware



CORBA Services



Plan

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

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*

Java Object Model

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

Frank Eliassen, SRL & Ifi/UiO 53

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

Frank Eliassen, SRL & Ifi/UiO 54

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

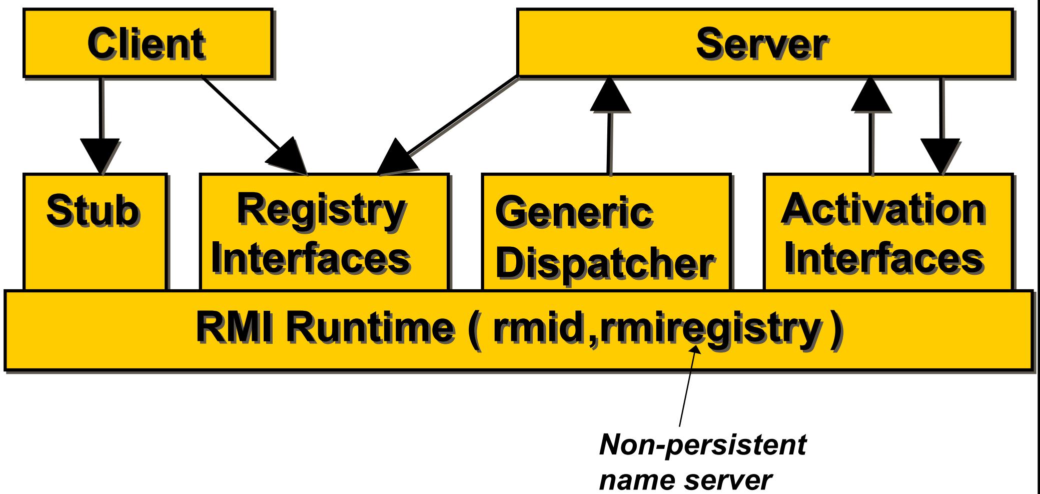
Java RMI parameter passing

- 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

Architecture of Java RMI



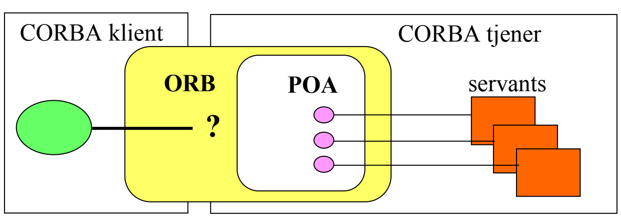
Summary

- 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

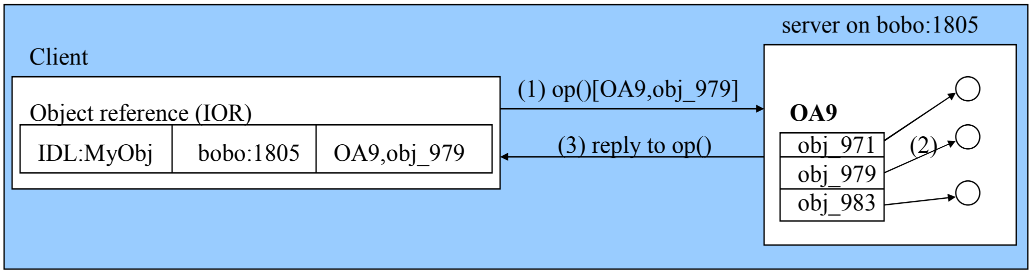
Extra slides

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



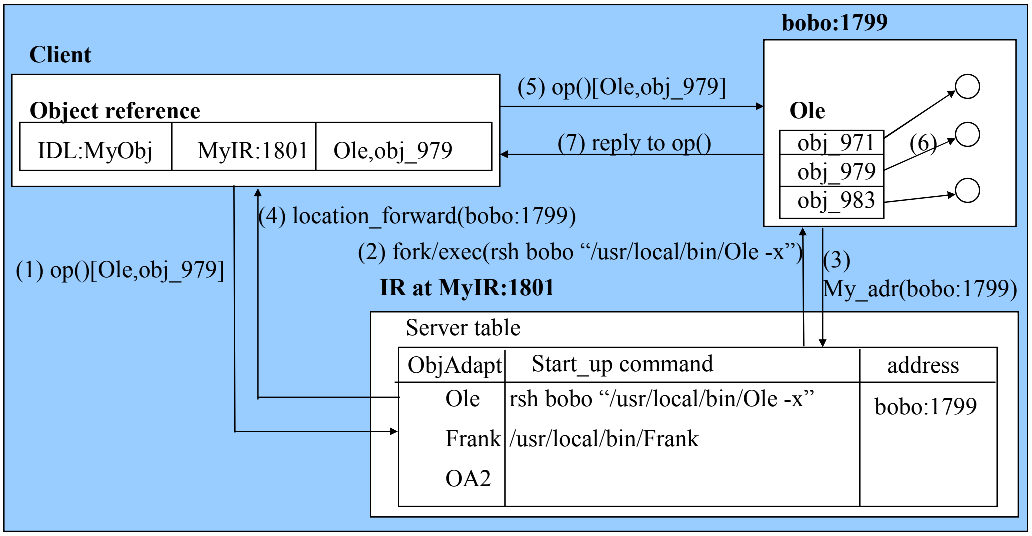
Locating transient IORs over IIOP



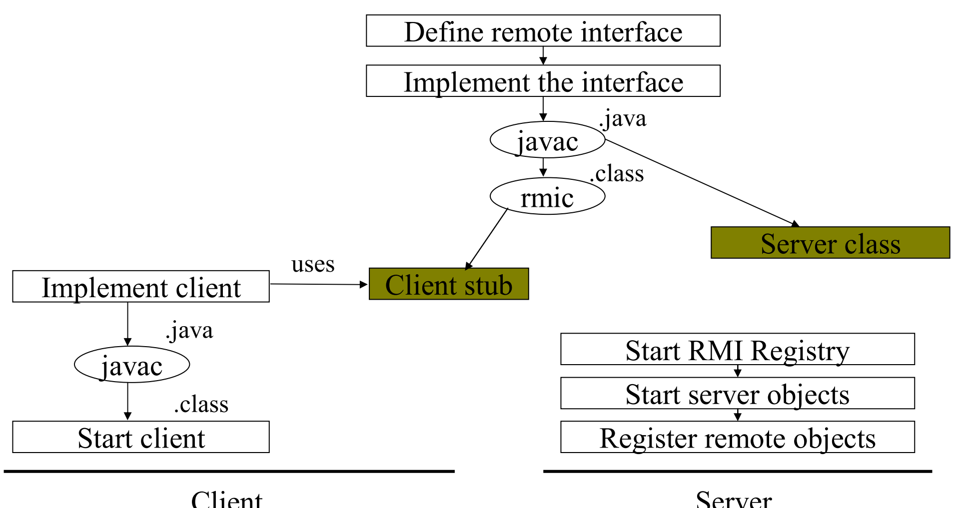
Locating persistent IOR over IIOP

- 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



Java RMI development process



Frank Eliassen, SRL & Ifi/UiO 64