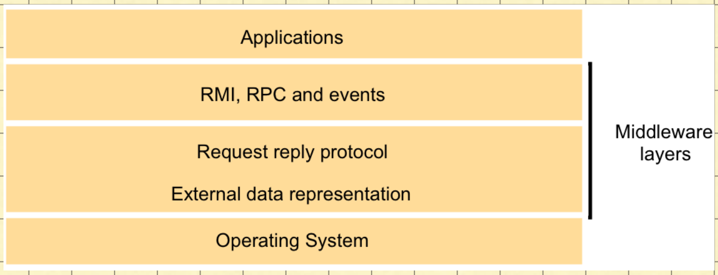
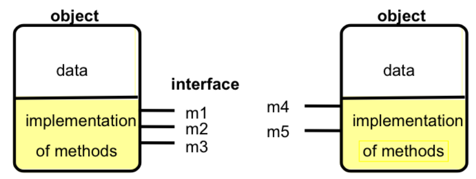
Middleware layers



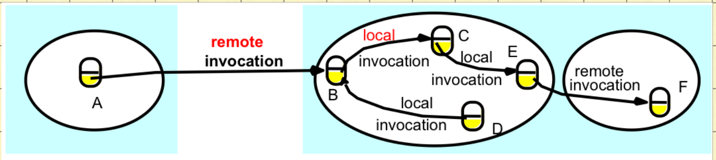
Objects



- Objects = Data (attributes) + Operations (methods)
  - encapsulating Data and Methods
  - State of Objects: value of its attributes
- Interact via interfaces:
  - define types of arguments and exceptions of methods

The object (local) model

- OO Programs:
  - (state is distributed in) a collection of interacting objects
- Interfaces
  - the only means to access data, make them remote?
- Actions
  - via method invocation
  - The state of the receiver may be changed
  - A new object may be instantiated
  - interaction, chains of invocations
  - may lead to exceptions, specified in interfaces
- Garbage collection
  - reduced effort, error-free (Java, not C++)



- Objects distributed (client-server models)
- Extend with
  - Remote object reference
  - Remote interfaces
  - Remote Method Invocation (RMI)

Remote object reference

- Object references
  - used to access objects which live in processes
  - can be passed as arguments, stored in variables,...
- Remote object references
  - object identifiers in a distributed system
  - must be unique in space and time
  - error returned if accessing a deleted object
  - can allow relocation (as in CORBA)

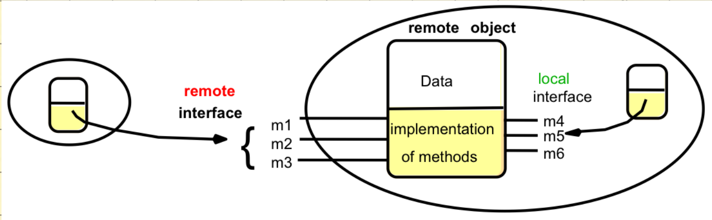
- An identifier for an object that is valid throughout the distributed system
  - must be unique
  - may be passed as argument, hence need external representation

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

Remote interfaces

- The class of a remote object implements the methods of its remote interface (e.g. as public instance methods in Java)
- Interfaces specify externally accessed
  - variables and procedures
  - no direct references to variables (no global memory)
  - local interface separate
- Parameters
  - input, output or both,
  - instead of call by value, call by reference
- No pointers
- No constructors

Remote object and its interfaces



- CORBA: Interface Definition Language (IDL)
- Java RMI: as other interfaces, keyword *Remote*

Handling remote objects

- Exceptions
  - raised in remote invocation
  - clients need to handle exceptions
  - timeouts in case server crashed or too busy
- Garbage collection
  - distributed garbage collection may be necessary
  - combined local and distributed collector
  - cf Java reference counting

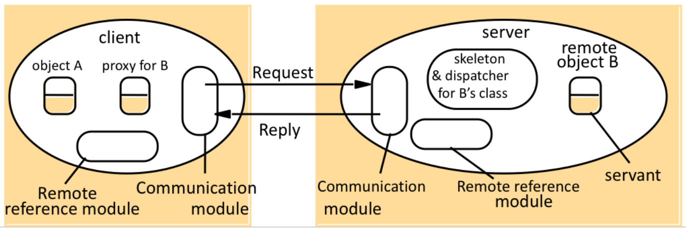
Design issues for RMI

- Invocation semantics
  - Local invocations are executed only once. This is not the case for remote invocations
- Maybe: invocation not guaranteed
- At least once: either a result or an exception (retransmission of request messages): Sun RPC
- At most once: Java and CORBA

Fault tolerance measures			*Invocation semantics*
Retransmit request message	*Duplicate filtering*	*Re-execute procedure or retransmit reply*	
No	Not applicable	Not applicable	*Maybe*
Yes	No	Re-execute procedure	*At-least-once*
Yes	Yes	Retransmit reply	*At-most-once*

Implementation of RMI

- Application-level Object A invokes a remote method in application-level object B
- Communication module: implements the request-reply protocol
  - use unique message ids (new integer for each message)
  - implement given RMI semantics
  - Receiver module selects dispatcher for the class of the object to be invoked passing on its local reference which it gets from the remote reference module in return for the remote object id in the request message



- Proxies: makes RMI transparent to clients by behaving like a local object to the invoker. Instead of executing an invocation, it sends a message
  - One proxy for each of the remote objects
- Remote reference module: translates between local and remote object references and creates remote object references and proxies
  - Remote object table: entries for all remote objects held by the process and entries for all local proxies
- Servants: an instance of a class which provides the body of a remote object. This eventually handles the remote requests passed on by the corresponding skeleton. Lives in a server process.
- A server has one dispatcher and one skeleton for each class representing a remote object.
- Dispatcher receives the request from communication module. Uses the *methodid* to select the appropriate method in the skeleton. Dispatcher and proxies use the same allocation of *methodIds* to the methods of a remote interface
- Skeleton: The class of a remote object has a skeleton which implements the methods in the remore interface. Unmarshalls the arguments in the request and invokes the corresponding method in the servant. Then sends reply to the sending proxy's method

Binding and activation

- The binder
  - mapping from textual names to remote object references
  - used by clients as a look-up service (cf Java RMIregistry)
- Activation
  - objects active (within running process) and passive (=implementation of methods + marshalled state)
  - activation = create new instance of class + initialise from stored state
- Activator
  - records location of passive and active objects
  - starts server processes and activates objects within them

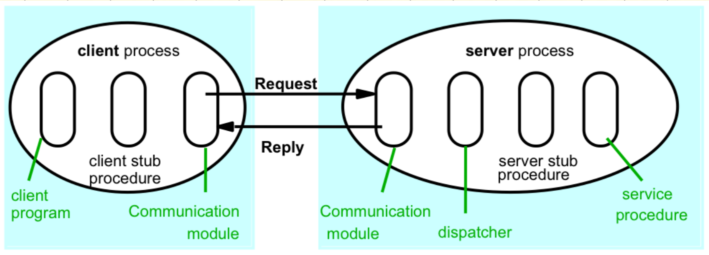
Object location issues

- Persistent object stores
  - stored on disk, state in marshalled form
  - readily available
  - cf Persistent Java
- Object migration
  - need to use remote object reference and address
- Location service
  - assists in locating objects
  - maps remote object references to probable locations

Remote Procedure Call (RPC)

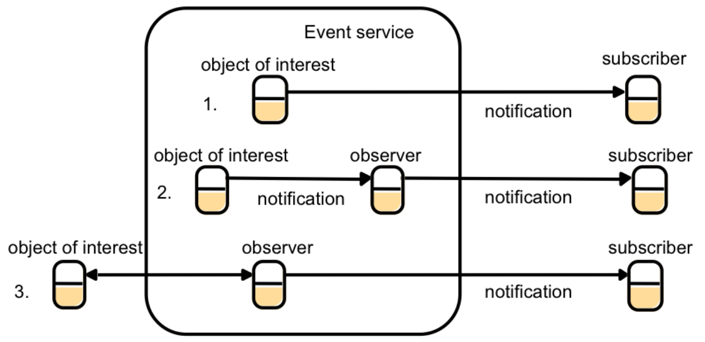
- RPC
  - historically first, now little used
  - over Request-Reply protocol
  - usually at-least-once or at-most-once semantics
  - can be seen as a restricted form of RMI
  - cf Sun RPC
- RPC software architecture
  - similar to RMI (communication, dispatcher and stub in place of proxy/skeleton)

RPC client and server



Implemented over Request-Reply protocol.

Events and Notifications


Event service
object of interest
subscriber
1.
notification
object of interest
observer
subscriber
2.
notification
notification
object of interest
observer
subscriber
3.
notification