Distributed Operating Systems lecture 1

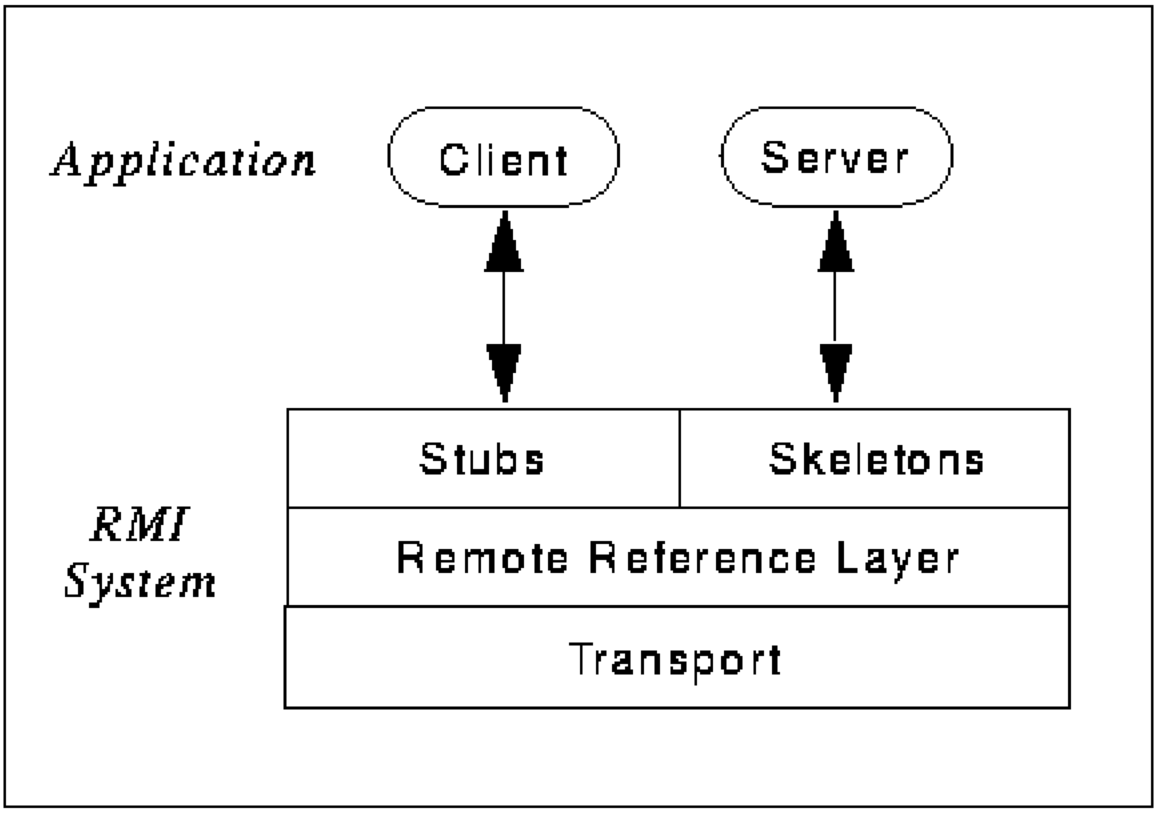
Remote Method Invocation

Loosely and tightly coupled multiprocessing

Network OS vs Distributed OS - ch 16, 17

Types of Transparency - 16.2, 17.2

Java's Remote Method Invocation



The problem of reference parameters

How would you solve the problem?

What mechanisms do you need?

Need to be able to transmit objects - object serialization.

Since objects can hold references to other objects this recursively traverses all references to make sure all such objects are transmitted as well.

No problem with different data formats.

The Java VM does the translation with the native format.

<u>Client stub</u>

Implements the same interface (method specification) as one of the server objects.

Does the object serialization and passes the information on to the remote reference layer (via a serialized marshal stream).

If the objects in the call are local they are passed by copy.

If the objects are remote they are passed by reference.

Gets the results (or exceptions) back and turns them back into objects (from the serialized marshal stream).

<u>Server skeleton</u>

What are the three steps this performs?

Constructs objects from the marshal stream.

Makes the method call.

Packages up results or exceptions to the marshal stream.

A client held reference to a remote object is actually a reference to a local stub.

Stubs and skeletons are generated by a special remote method invocation compiler - `rmic`.

> rmic is run on the compiled remote object class

Remote reference layer

Deals with determining where the remote object is.

It may in fact be replicated and the method call has to go to several locations.

Also deals with the differences in the semantics of references to the server.

> some servers are always running
>
> some only run when a method call is made

This level of starting up remote objects is not visible above the remote reference layer.

This layer also deals with reconnection strategies if the remote object becomes inaccessible.

Transport

Deals with connection setup and management and the actual dispatching to remote objects.

On the server side it deals with accepting connections and maintains a table of remote objects which reside in its address space.

Transports can be connectionless - UDP, datagrams

or streams based - TCP.

Finding the remote object

The server registers with a name server.
`public static void java.rmi.Naming.rebind(String name, Remote obj)`

The client finds it by calling
`public static Remote java.rmi.Naming.lookup(String name)`

The basic bootstrap name server is called - `rmiregistry` and is on port 1099 by default.

Remote garbage collection - Birrell, Nelson and Owicki 1994

The first reference to a remote object from a particular client leads to a reference count increment.

When the client end of this object is unreferenced a message is sent to decrement the count.

Object can be garbage collected when no more live references to it and no more local references.

CORBA - Common Object Request Broker Architecture

There are other forms of RMI.

Designed to be used over heterogeneous object systems.

Need a language neutral object model.

Distributed Operating Systems

A distributed system is:

"one on which I cannot get any work done because some machine I have never heard of has crashed".

<u>Tightly-coupled and Loosely-coupled systems</u>

Tightly-coupled

    shared memory

    must be the same OS

    processes can communicate via shared memory

Loosely-coupled

    network connection

    could be different OSs, or different parts of the OS

    processes must communicate via messages

<u>What advantages does a network of processes offer?</u>

More work can get done

Ability to share devices, programs and data

Greater reliability

Easier to expand

Two main approaches

Network OS 16.1

Communications layer on top of a *normal* OS.

Possibly different OS.

User is aware of different machines.

Some can copy files across the network but not share them. e.g. ftp

In this case the file location is explicitly known.

Others can share files but the location is still part of the name.

Distributed OS 16.2

Aim to have the system look like one machine.

Locations of everything are hidden from the users/programmers.

Location transparency

No (obvious) connection between the name of a resource (process, file) and its position on the system.

Migration transparency

Resources can move around the system.

- Files
- Processes - for load balancing or resource access requirements, or maintenance
- (RPC is an example of computation migration)

Need location transparency to implement.

Mach and ports

Mach does everything with messages sent to object ports

(not to be confused with machine port numbers)

so we have location transparency and possible migration transparency.

Replication transparency

Replicating resources without the user being aware of it.

Safety.

Speed of access.

Need location and migration transparency to implement.

Parallelism transparency

Compiler and OS combine to maximise the amount of work done in parallel.

Automatically - the best mix of work on which processors depending on the configuration of the system.

Different levels of parallelism

Coarse grained to fine grained parallelism.

processes -> threads -> methods -> … -> expressions fine grained parallelism is only profitable with unconventional machine designs

otherwise the communication overhead is prohibitive

Dealing with files over a network

Two methods (we have mentioned these before - in DSM)

Caching of data vs remote service 17.3.5

Copy the whole file to the processor where it is requested.

Provide a server which sends chunks of the file as requested.

What are the pros and cons?

caching - usually faster, more efficient, scales better

remote service - simpler to implement because of no consistency problem, uses less local memory (primary or secondary)

Consistency semantics

Now we have copies of file information around the system how do we keep them consistent?

It depends on the semantics we are going to use.

Unix - writes are reflected *instantaneously* to all sharing processes.

Session - a copy is sent on opening and the copies remain independent. Writes only get sent back when the file is closed.