Desarrollo de Aplicaciones en Red

José Rafael Rojano Cáceres
http://www.uv.mx/rrojano

El modelo de comunicación

General concepts

- As we saw in a Distributed System the logical and physical component are separated, so on, we need to communicate it in some way.
- Up to now we have talk about:
  - *Message Passing*
  - *Client/Server*
  - *RPC*

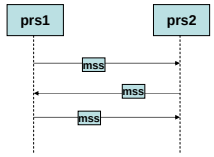
Models of communication

- *Message Passing*
- *Client/Server*
- *Peer to Peer*
- *Mom (Message oriented Middleware)*
- *RPC*
- *Distributed Objects*
  - RMI
  - Corba
  - DCOM
- *Mobil Agents*

Each model define the way in which components interact each other
- The pattern of communication
- The functional role

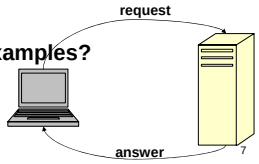
Message Passing

- It defines some primitives like send, receive, etc.
- An layer abstraction for them are sockets.
- Their functionalities includes communication:
  - *Synchronous*
  - *Asynchronous*
  - *Queues*



Client / Server

- In this model there are processing of information in both client and server
- The role for serve includes offering different services
- The communication can be:
  - *Synchronous*
  - *Queue*
- Can you give some examples?
  - *…*



7

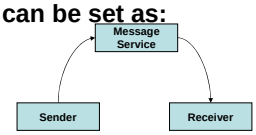
Peer to Peer

- In this model both process can be seen as server or client
- The role is designed interactively
- The communication as well as client / server can be:
  - *Synchronous*
  - *Queue*

8

MOM (Message Oriented Middleware)

- This model implements a message service
- The message are persistent
- The communication is:
  - *Asynchronous*
    - *Queue*
- The communication can be set as:
  - *Point to point*
  - *Multicast*



MOM (2)

- *It facilitates communication between distributed applications*
- *MOM is mainly oriented to asynchronous communications, while p2p or RPC are synchronous*

10

Middleware

- *There is not a set of standardize functionalities for it, because middleware change fast, but in general we can identify:*
  - Presentation services: forms manager, printing manager, hypermedia linker
  - Communication services: P2P, RPC, Message Passing
  - Control services: transaction manager, scheduler
  - Information services: directory, relation database manager, repository manager
- *MOM falls in communication services category*

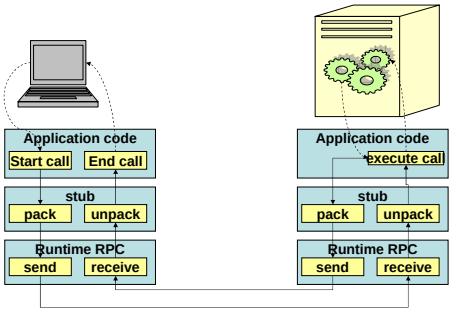
11

RPC (1)

- The remote procedure call defined a communication:
  - *Synchronous*
  - *Queue*
- Process of communication
  - *The client process package the parameters in a message and send to server waiting the answer*
  - *The server unpack parameters, execute locally the call, get the result, again pack it and send it back*

12

RPC (2)

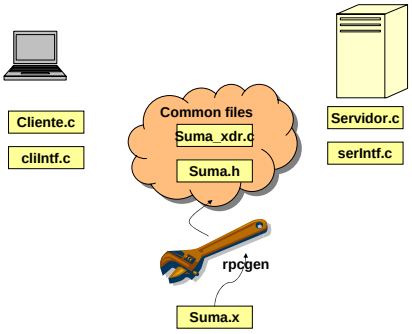


RPC (3)

- ***A stub is generated through RPC software***
- ***Stub task at the Client are:***
  - **Localize the server**
  - **Package and unpackage parameters**
  - **Send message and wait for the answer**
- ***Stub task at the server is:***
  - **Similar task that client**
- ***Stubs are independent of implementation only depends of the interface***

An example with RPC

How RPC look like?



Suma.x

```
struct peticion {
        int a;
        int b;
};

program SUMAR {
        version SUMAVER {
                int SUMA(peticion) = 1;
        } = 1;
} = 99;
```

Suma.h

```
#ifndef _SUMA_H_RPCGEN
#define _SUMA_H_RPCGEN
#include <rpc/rpc.h>
struct peticion {
        int a;
        int b;
};
#define SUMAVER ((u_long)99)
#define SUMA ((u_long)1)
extern int * suma_1(peticion *, CLIENT *);
extern int * suma_1_svc(peticion *, struct
svc_req *);
#endif /* !_SUMA_H_RPCGEN */
```


Servidor.c

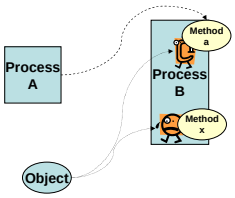
```
#include "suma.h"
int *suma_1_svc(peticion *argp, struct svc_req
*rqstp)
{
        static int result;
        result = argp->a + argp->b;
        return(&result);
}
```

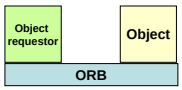
Cliente.c

```
#include "suma.h"
main( int argc, char* argv[] )
{
        CLIENT *clnt;
        int *res;
        peticion suma_1_arg;
        char *host;
        if(argc < 2) {
           printf("usage: %s server_host\n",
argv[0]);
           exit(1);
        }
        host = argv[1];
        /* find server*/
        clnt = clnt_create(host, SUMAR,
SUMAVER, "udp");
        if (clnt == NULL) {
                clnt_pcreateerror(host);
                exit(1);
        }
        suma_1_arg.a = 5;
        suma_1_arg.b = 2;
        res = suma_1(&suma_1_arg, clnt);
        if (res == NULL) {
                clnt_perror(clnt, "call failed:");
        }
        printf("The result is %d\n", *res);
        clnt_destroy( clnt );
}
```

Distributed Objects

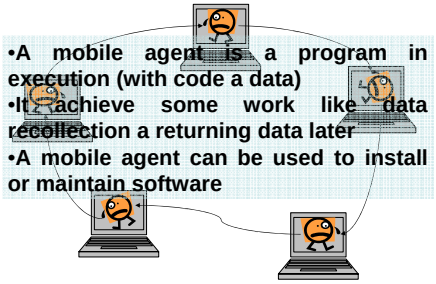
- *JAVA RMI*
  - Remote Method Invocation
- *CORBA*
  - Common Object Request Broker





19

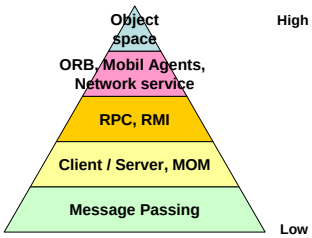
Mobile Agents



- A mobile agent is a program in execution (with code a data)
- It achieve some work like data recollection a returning data later
- A mobile agent can be used to install or maintain software

20

Abstraction Level



High

Low

21

Object Space

- *Object spaces form the basic of distributed objects by supplying the location.*
- *In Small talk there are two types of Object Space:*
  - Export sets: maintain lists of object exported by space, not effective for remote reference
  - Full object space: support full reference
- *Remote references are proxies which identify particular object in particular object spaces (location) and explain how to get a message to the object*
- *In small talk symbolic references allow object spaces to implement primitive naming service.*

22

Denominación y servicio de nombres

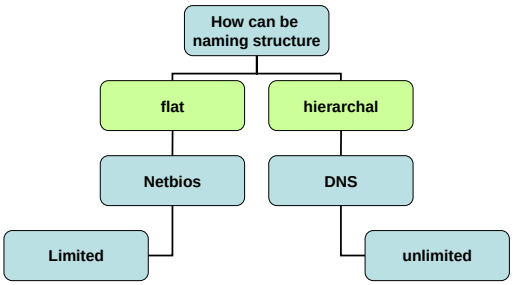
23

Naming (denominación)

- This is the name that receive the process of association between logical objects and physical objects.
  - *For example: naming files for user is different to management files by the OS (tracks and sectors)*
  - *In a distributed system it's necessary naming the machine also*

24

Naming structure



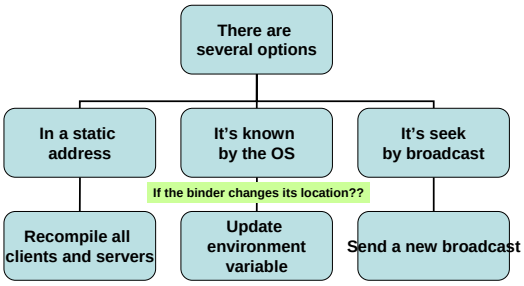
25

Translating names

- A process in charge to translate names is the *binder*. The process of translation is know as *binding*.
- The binder as the task of:
  - *Resolution: translate name for id*
  - *Inclusion: adding new names*
  - *Erasing: deleting old names*
  - *Modification: alter existing names*
- The binder must provide fail-over
- When accessing an object must be established *capabilities* (rights).
  - *What operation can be done*

26

Where the binder is?



27

Characteristic in a binder

- It's desirable that the binder has:
  - *Transparent location (static)*
    - *The name does not reveal location*
  - *Independent location (dynamic)*
    - *The name of the object does not change when it's relocated*
  - *An static address and knows possition for the OS, are static*
  - *Broadcasting is dynamic*

28

Remote Procedure Call

In detail

29

RPC (1)

- Recalling communication takes place at the lower levels through message passing.
- The protocol used is RR (request/response)
- RPC is a mechanism to implement the *transparency* in the systems.
- The RPC is not more that just a remote operation customized by the mask of a procedural interface.
- RPC has its origin with Birrell & Nelson, 1984 as mechanism to solve the passing data

30

RPC (2)

- Recall the mechanism to do the call is based in a *stub*.
- For the client the stub:
  - *Take the parameters from local stack*
  - *Do all the pass seen at page 12*
- At the server
  - *Take the parameters from message and put in local stack*

But the process is not as simple as I show it the last slides…..

31

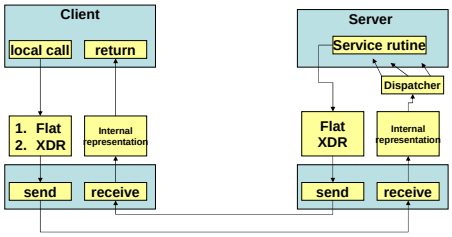
Task for RPC

- Service Interface
  - *It's on charge to marshalling the of parameters*
  - *It's written in conventional language*
- Seek of server
  - *As we saw this process is know as binding*
- Management communication
  - *The task is transmit and receive*

32

Service Interface

- It's very important for the whole service



33

Passing of parameters

- Flatting data mean transform a data structure in a collection of bytes.
- It is not a easy task because of internal data codification (ascii, ebcdic, …)
- Data representation (Big a Little Endian)
- For this it's necessary an independent representation
  - *XDR (external data representation) proposed by Sun, Xerox, ASN.1*

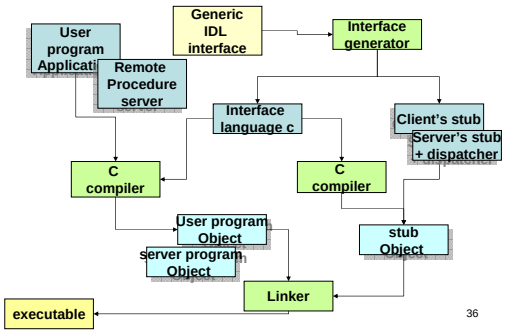
34

Stub generation

- With the interface defined can be implemented the stub.
- Notice that implementation in client and server can be different because these can be written in different language.
- For that it's necessary to define the interface in a language IDL (Interface Definition Language)
  - *The IDL in general provides semantic that languages doesn't offer*
- Having this definition it can be generated automatically the stub for server and for client

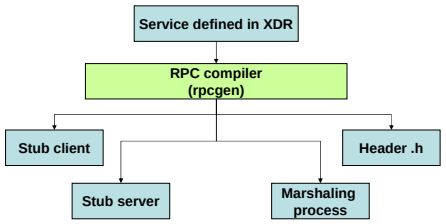
35

Client & Server Stubs



36

Example RPC SUN



Remeber the binder is local to each client

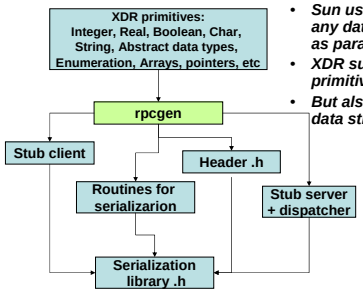
The interface

```
//Definition in XDR, for service call "Archiver"
CONST MAX=1000;
typedef int ID_File;
typedef int Ptr_File;
typedef int Length;
struct Data_s { Length length_data;
                char buffer[MAX];
                int error};
struct Writting_Args { ID_File file;
                Ptr_File position;
                Data_s data};
struct Reading_Args { ID_File file;
                Ptr_File position;
                Length leng_data};
program Archiver{
  version VER{
          void write_data(WriteArg)=1;
          Data_s read_data(ReadArg)=2;
  }=2;
}=9999;
```

- *XDR*
  - Contains the definition of the operations
  - The ID
  - Number of service
- *With the XDR is generated*
  - Client's Stub
  - Main server program + dispatcher
  - Serialization routines
  - header

Policies (1=w, 2=r, 9999=save

Marshaling



- *Sun uses XDR to represent any data structure to pass as parameter*
- *XDR support some primitives data type*
- *But also can be defined new data structures for the user*

Binding for RPC

- RPC does not has a binding service in stead there is binder for machine (*portmapper*)
- The binder is a server listening in an static port
- The client must specify address, the stub seek the server through the local binder

- To be continued…….

Reference

- http://www2.sims.berkeley.edu/courses/is206/f97/GroupB/mom/

43