Capítulo 4
Objetos distribuidos e invocación remota

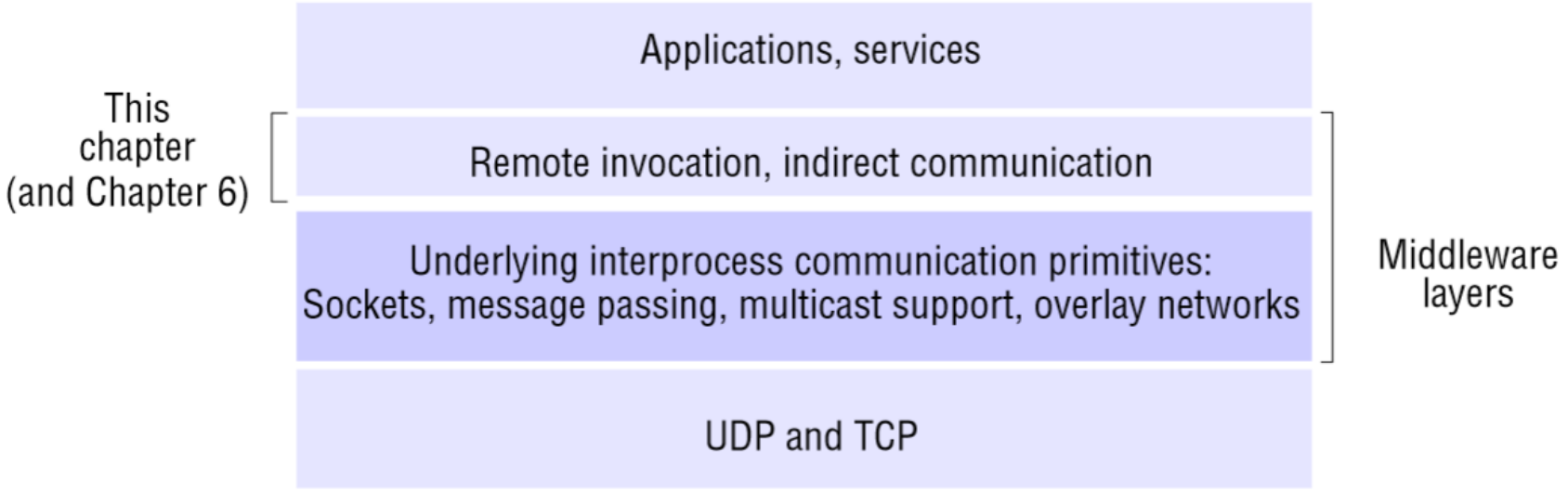
Sistemas Distribuidos

Universidad Nacional de Asunción
Facultad Politécnica
Ingeniería Informática

Profesor: Ing. Fernando Mancía

Aplicación Distribuida: Aplicaciones que se componen de programas cooperantes corriendo en procesos distintos. Es necesario invocar operaciones en otros procesos, generalmente en computadores diferentes.

Figure 5.1 Middleware layers



RPC. Uno de los modelos más antiguos y amigables para los programadores: la extensión del modelo de llamada a procedimiento convencional a sistemas distribuidos (llamada a procedimiento remoto), permite a los programas de cliente llamar procedimientos de forma transparente en el servidor.

RMI. Es una extensión de la invocación a métodos locales que permite que un objeto que vive en un proceso invoque los métodos de un objeto que reside en otro proceso.

Protocolo Petición Respuesta. Representa un patrón de paso de mensajes y soporte de intercambio bidireccional para la comunicación cliente-servidor. Proporciona un soporte de bajo nivel para ejecución remota, RPC y RMI.

Middleware. Software que proporciona un modelo de programación sobre bloques básicos arquitectónicos, esto es: procesos y paso de mensajes.

Emplea protocolos basados en mensajes entre procesos para proporcionar abstracciones de un nivel mayor, tales como:
" invocaciones remotas y eventos".

Proporciona:

a) transparencia de ubicación
b) independencia de detalles de protocolos de comunicación,
c) independencia de los sistemas operativos, e
d) independencia el hardware de los computadores.

Se puede tener implementaciones en lenguajes diferentes

Transparencia frente a ubicación:
En RPC, el cliente que llama a un procedimiento no puede discernir si el procedimiento se ejecuta en el mismo proceso o en un proceso diferente
Tampoco necesita conocer la ubicación del servidor. Análogamente…

En RMI el objeto que realiza la invocación no sabe si el objeto que invoca es local o no. No requiere conocer su ubicación.

En los P. distribuidos basados en eventos, los objetos que generan eventos y los objetos que reciben notificaciones de esos eventos tampoco necesitan estar al corriente de sus ubicaciones respectivas.

Protocolos de comunicación: Los protocolos que dan soporte a las abstracciones del middleware son independientes de los protocolos de transporte subyacentes. Por ejemplo el protocolo petición-respuesta puede estar implementado tanto sobre UDP como sobre TCP.

Hardware de los computadores: Se emplean en el empaquetado y desempaquetado de mensajes. Éstos ocultan las diferencias de arquitectura en el hardware, como el ordenamiento de los bytes.

Sistemas operativos: Las abstracciones de mayor nivel que provee la capa de middleware son independientes de los sistemas operativos subyacentes

Utilización de diversos lenguajes de programación: Diversos middleware se diseñan para permitir que las aplicaciones distribuidas sean escritas en más de un lenguaje de programación. En particular *CORBA* permite a los clientes escritos en un lenguaje invocar métodos en objetos que viven en programas servidores escritos en otro lenguaje. Esto se obtiene empleando un *lenguaje de definición de interfaz* o IDL *(interface definition language)* para definir interfaces.

Protocolo Petición Respuesta

Protocolos de Petición Respuesta

Request Reply Protocols

Comunicación cliente-servidor

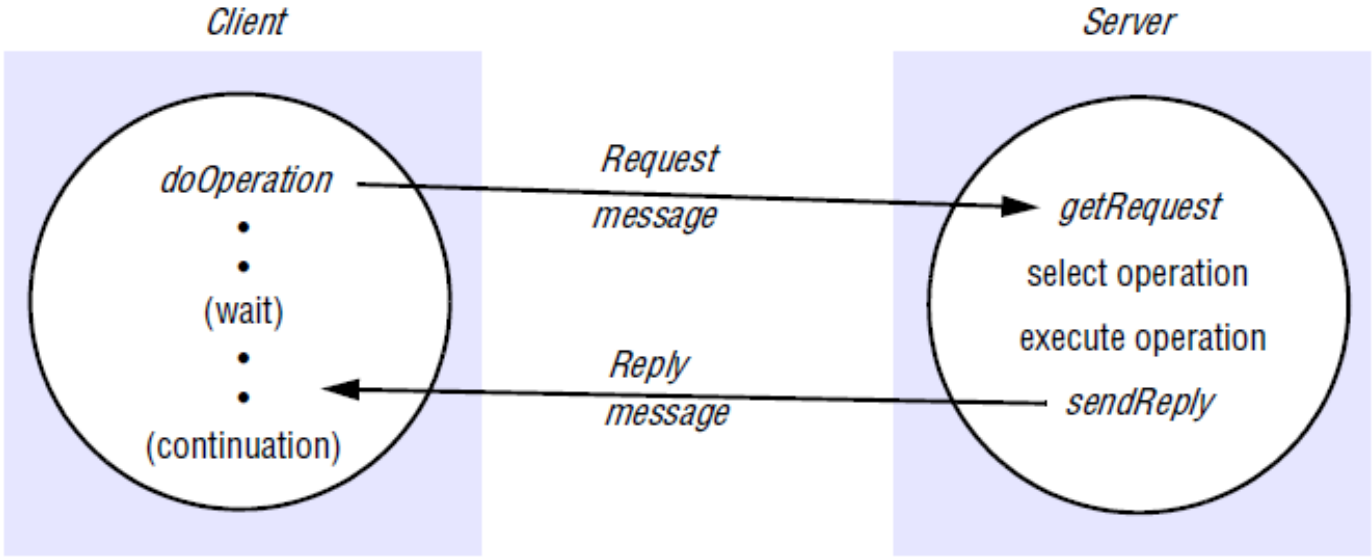
Comunicación cliente-servidor, está orientada a soportar los roles y el intercambio de mensajes de las interacciones típicas cliente-servidor.

En el caso normal, la comunicación petición-respuesta es síncrona, ya que el proceso cliente se bloquea hasta que llega la respuesta del servidor.

Esta comunicación también puede ser **fiable** ya que la respuesta del servidor es, en efecto**, un acuse de recibo para el cliente**.

La comunicación cliente-servidor asíncrona es una alternativa que puede ser útil en situaciones donde los clientes pueden recuperar las respuestas.

Figure 5.2 Request-reply communication



public byte[] ***doOperation*** *(RemoteObjectRef o, int methodId, byte[] arguments)*
sends a request message to the remote object and returns the reply.
The arguments specify the remote object, the method to be invoked and the arguments of that method.

public byte[] ***getRequest*** *();*
acquires a client request via the server port.

public void ***sendReply*** *(byte[] reply, InetAddress clientHost, int clientPort);*
sends the reply message reply to the client at its Internet address and port.

Figura 4.12
Operaciones del protocolo petición-respuesta

Figure 5.4 Request-reply message structure

messageType	*int (0=Request, 1= Reply)*
requestId	*int*
remoteReference	*RemoteRef*
operationId	*int or Operation*
arguments	*// array of bytes*

Protocolo petición-respuesta

Modelo de fallos del protocolo petición-respuesta. Si las tres primitivas *hazOperacion, damePeticion, enviaRespuesta* se implementan con datagramas UDP, adolecerán de los mismos fallos de comunicación que cualquier otro ejemplo de aplicación de UDP. Esto es:

. Sufrirán fallos de omisión.
. No se garantiza que los mensajes lleguen en el orden de emisión.

Además, el protocolo puede padecer el fallo de los procesos .Presuponemos que los procesos pueden caer.

Es decir, cuando se detienen, permanecen detenidos. En aquellas ocasiones en las que un servidor falle o se elimine un mensaje de petición o respuesta, ***hazOperacion*** **utiliza un** ***time out*** para esperar el mensaje de respuesta del servidor. La acción a tomar cuando se supera el tiempo límite de espera depende de las garantías de entre ofrecidas.

Protocolo petición-respuesta

Tiempos de espera límite: Existen varias opciones respecto a lo que la primitiva *hazOperación* puede hacer después de superar un time out.

La opción más simple es devolver inmediatamente control indicando al cliente que la primitiva *hazOperacion* ha fracasado. Ésta no es la aproximación normal; se puede alcanzar un time out debido a la pérdida de un mensaje de petición o de respuesta y, en el último caso, la operación se habrá realizado. **Ejemplo: "transferencia electronica de dinero indeterminada"**

Para compensar la posibilidad de pérdida de los mensajes, *hazOperacion* manda el mensaje de petición de forma repetida hasta que bien obtiene una respuesta o bien está razonablemente seguro que el retraso se debe a una falta de respuesta del servidor, más que a mensajes perdidos. En algún momento, cuando *hazOperacion* devuelva el control indicará al cliente mediante una excepción que no ha recibido resultado alguno

Protocolo petición-respuesta

Eliminación de mensajes de petición duplicados: En los casos de mensaje de petición se retransmite, el servidor puede recibir más de uno.

Por ejemplo, el servidor puede recibir el primer mensaje de petición pero precisará más tiempo que el *timeout* del cliente para ejecutar el comando y devolver la respuesta.

Esto puede llevar a que el servidor ejecute una operación mas una vez para la misma petición. Para evitarlo, se diseñó el protocolo para reconocer sucesivos mensajes (del mismo cliente) con el mismo identificador de petición y para eliminar los duplicados. Si el servidor no hubiera enviado aún la contestación, no necesita realizar ninguna operación especial, transmitirá la respuesta cuando termine de ejecutar la operación.

id_cliente # id_peticion

Protocolo petición-respuesta

Pérdida de mensajes de respuesta: Si el servidor ya ha enviado la respuesta cuando recibe una petición duplicada necesitará ejecutar otra vez la operación para obtener el resultado, a menos que haya almacenado el resultado de la ejecución original. Algunos servidores pueden ejecutar sus operaciones más de una vez y obtener el mismo resultado cada vez.

Una *operación* ***idempotente*** es una operación que puede ser llevada a cabo repetidamente con el mismo efecto que si hubiera sido ejecutada exactamente una sola vez. Por ejemplo, una operación que añade un elemento a un conjunto es una operación idempotente porque siempre tendrá el mismo efecto cada vez que se realice, mientras que la operación de añadir un elemento a una secuencia no es una operación idempotente

Un servidor cuyas operaciones sean todas idempotentes no tendrá que tomar medidas especiales para evitar que se ejecuten más de una vez.

Protocolo petición-respuesta

Historial: En aquellos servidores que necesitan retransmitir las respuestas sin tener que vol ejecutar las operaciones, es imprescindible la utilización de un historial. El término *historial o histórico* se utiliza para referirse a una **estructura que contiene el registro de los mensajes de respuesta que han sido transmitidos**. Una entrada en el historial contiene un identificador de petición, un mensaje y un identificador del cliente al que fue enviado.

Su propósito es permitir que el servidor pueda retransmitir los mensajes de respuesta cuando los clientes se lo soliciten. Un problema asociado con el uso del historial es el costo de almacenamiento.

Un historial se volverá de , grande a menos que el servidor pueda decidir cuándo no será necesario seguir almacenando mensajes para su retransmisión.

Protocolo de intercambio RPC

En la implementación de los distintos tipos de RPC se utilizan tres protocolos con diferentes semánticas en presencia de fallos de comunicación.

El protocolo *petición (R).* cuando el procedimiento no devuelve ningún valor y el cliente no necesita confirmación de que ha sido ejecutado. El cliente puede seguir inmediatamente después de haber enviado el mensaje de petición ya que no tiene que esperar un mensaje de respuesta

El protocolo *petición-respuesta (RR).* No se necesitan mensajes especiales de acuse de recibo, ya que los mensajes de respuesta del servidor sirven como confirmación de las peticiones cliente.

El protocolo *petición-respuesta-confirmación de la respuesta (RRA).* está basado en el intercambio de tres mensajes: petición-respuesta-confirmación de la respuesta.

Figura 4.14
Protocolos de intercambio RPC

Name	*Messages sent by*		
	Client	*Server*	*Client*
R	*Request*		
RR	*Request*	*Reply*	
RRA	*Request*	*Reply*	*Acknowledge reply*

HTTP: un ejemplo de protocolo petición-respuesta

HTTP: basado en protocolo petición – respuesta. Cliente: Navegador. Servidor: WebServer.
HTTP Especifica los mensajes involucrados: En ellos, los metodos, los argumentos, resultados, reglas para representar EMPAQUETAR en dichos mensajes de intercambio.

Soporta un conjunto fijo de métodos (GET, PUT, POST, etc.) que son aplicables a todos los recursos. Al contrario de los protocolos anteriores, cada objeto tiene sus propios métodos. Además permite la negociación de contenidos y una autenticación del tipo *clave de acceso.*

Negociación del contenido: las peticiones de los clientes pueden incluir información sobre que tipo de representación de datos pueden aceptar (por ejemplo lenguaje o tipo de medio), haciendo posible que el servidor pueda elegir la representación más apropiada para el usuario.

HTTP: un ejemplo de protocolo petición-respuesta

Autenticación: se utilizan credenciales y desafíos para conseguir una autenticación del estilo *clave de acceso.* En el primer intento de acceso al área protegida por palabra clave, el servidor responde con un desafío aplicable al recurso. Cuando el cliente recibe el desafío pide al usuario un nombre y una palabra de paso, y se los envía al servidor en las siguientes solicitudes.

HTTP se implementa sobre TCP. En la versión original del protocolo, cada interacción cliente-servidor se componía de los siguientes pasos:

. **El cliente solicita una conexión al puerto del servidor por defecto o a otro especificado en la petición aceptada por el servidor.**
. El cliente envía un mensaje de petición al servidor.
. El servidor envía un mensaje de respuesta al cliente.
. Se cierra la conexión.

HTTP: un ejemplo de protocolo petición-respuesta

Métodos HTTP: Cada petición de un cliente especifica el nombre de un método que habrá de ser aplicado al recurso en el servidor y el URL de dicho recurso. El mensaje de respuesta indica el estado de la petición. Las peticiones y las respuestas pueden contener también datos, el contenido de un formulario o la salida de un programa ejecutado en el servidor web.
Los métodos considerados son los siguientes:

GET: pide el recurso cuyo URL se da como argumento. Si el URL se refiere a datos, entonces el servidor responderá enviando de vuelta los datos indicados por el URL. Si el URL se refiere a un programa, entonces el servidor web ejecutará el programa y devolverá su salida al cliente.
Se pueden añadir argumentos al URL; por ejemplo, un GET se puede utilizar para enviar el contenido de un formulario a un programa *cgi* que los tomará como entrada. La operación GET puede condicionarse a la fecha de modificación del recurso indicado. El método GET también puede configurarse para obtener parte de los datos.

HTTP: un ejemplo de protocolo petición-respuesta

HEAD: esta petición es idéntica a GET, sólo que no devuelve datos. Sin embargo, devuelve toda la información sobre los datos, como el tiempo de la última modificación, su tipo o tamaño.

POST: especifica el URL de un recurso (por ejemplO un programa) que puede tratar lo datos aportados con la petición. El procesamiento llevado a cabo sobre los datos depende del programa especificado en el URL. Este método está diseñado para:

. Proporcionar un bloque datos (por ejemplo los obtenidos en un formulario) a un proceso gestión de datos como un servlet o un programa *cgi.*
. Enviar un mensaje a un tablón de anuncios, lista de correo o grupo de noticias.
. Modificar una base de datos con una operación de añadir registro.

HTTP: un ejemplo de protocolo petición-respuesta

PUT: indica que los datos aportados en la petición deben ser almacenados con la URL aportada como su identificador, ya sea como una modificación de datos existentes o como la creación de un recurso nuevo.

DELETE: el servidor borrará el recurso identificado por el URL. El servidor no siempre permitirá esta función, en cuyo caso se devolverá una indicación de fallo.

OPTIONS: el servidor proporciona al cliente una lista de métodos aplicables a un URL (por ejemplo, *GET, HEAD, PUT)* y sus requisitos especiales.

TRACE: el servidor envía de vuelta el mensaje de petición. Se utiliza en procesos de depuración.

Mensaje HTTP request

method	*URL or pathname*	*HTTP version*	*headers*	*message body*
GET	//www.dcs.qmw.ac.uk/index.html	HTTP/ 1.1		

Mensaje HTTP reply

HTTP version	*status code*	*reason*	*headers*	*message body*
HTTP/1.1	200	OK		resource data

HTTP -Referencias

https://www.w3.org/Protocols/

http://httpwg.org/

https://http2.github.io/

RPC

Llamada a procedimiento Remoto
Remote Procedure Call

RPC

La llamada a procedimiento remoto (RPC) representa un adelanto en computación distribuida, con el objetivo de hacer que la programación de sistemas distribuidos parezca similar, si no idéntica, a la programación convencional.

Logra un alto nivel de transparencia de distribución. Esta unificación se logra de una manera muy simple, extendiendo la abstracción de una llamada de procedimiento a entornos distribuidos.

Una llamada a un procedimiento remoto es muy similar a una invocación a un método remoto, en la que un programa cliente llama a un procedimiento de otro programa en ejecución en un servidor.

Los servidores pueden ser clientes de otros servidores para permitir cadenas de RPC.

Un proceso servidor **define en su *interfaz de servicio*** los procedimientos disponibles para ser llamados remotamente. RPC, como RMI, puede implementarse para ofrecer alguna de las semánticas de invocación discutidas, *al menos una vez* o *como máximo una vez.*

RPC se implementa usualmente sobre un protocolo petición-respuesta.

Los contenidos de los mensajes de petición y respuesta son los mismos que los que mostraron para RMI, excepto que se omite el campo *ReferenciaObjeto.*

El software que soporta RPC es similar al mostrado para RMI **excepto que no se requieren módulos de referencias remotas,** dado que las llamadas a procedimientos no tienen que ver con objetos y referencias a objetos.

El cliente que accede a un servicio incluye un ***procedimiento de resguardo*** *p/* cada procedimiento en la interfaz de servicio. Procedimiento de resguardo es similar al de un proxy.

Se comporta como un procedimiento local del cliente, pero en lugar de ejecutar la llamada, empaqueta el identificador del procedimiento y los argumentos en un mensaje de petición, que se envía vía su módulo de comunicación al servidor.

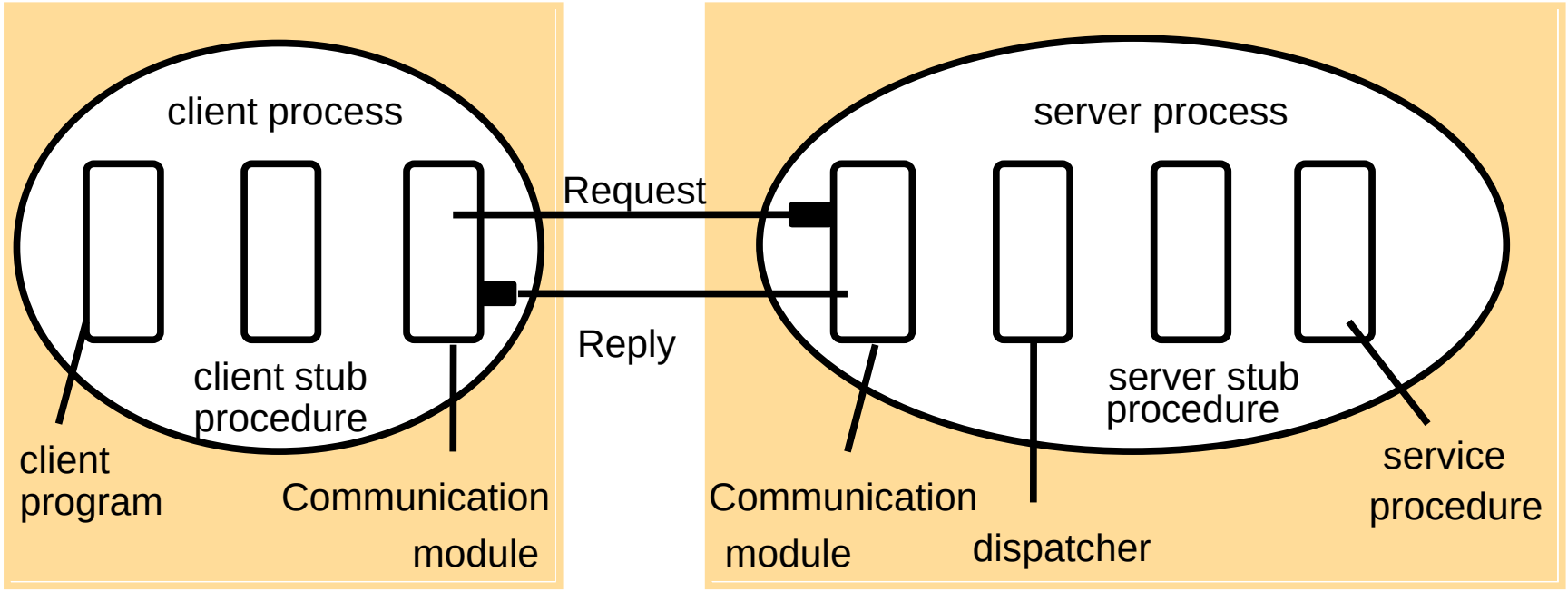
Cuando llega el mensaje de respuesta, desempaqueta los resultados. **El proceso servidor contiene un distribuidor junto a un procedimiento de resguardo de servidor y un procedimiento de servicio para cada procedimiento de la interfaz de servicio.**

El distribuidor selecciona uno de los procedimientos de resguardo según el identificador de procedimiento del mensaje de petición.

Un ***procedimiento de resguardo de servidor*** es como un método de esqueleto en el que se desempaquetan los argumentos en el mensaje de petición, se llama al procedimiento de servicio correspondiente y se empaquetan los datos con el resultado para el mensaje de respuesta.

Los procedimientos de servicio implementan los procedimientos en la interfaz de servicio

Figura 5.7
Papel de los procedimientos de resguardo de cliente y servidor en RPC



SUN RPC

Sun RPC, se diseñó para la comunicación cliente-servidor en el sistema de archivos en red Sun NFS. Sun RPC se denomina a menudo ONC (Computación de Red Abierta -*Open Network Computing)*

RPC se proporciona como parte de los varios sistemas operativos de Sun y otros del tipo UNIX. Los diseñadores tienen la opción de utilizar llamadas a procedimientos remotos sobre UDP o TCP.

Utiliza la semántica de llamada *al menos una vez.*

Como opción existe la posibilidad de difusión de RPC.

El sistema Sun RPC proporciona un lenguaje de interfaz denominado XDR y un compilador de interfaces llamado *rpcgen* cuyo uso está orientado al lenguaje de programación C.

Figura 5.8
Interfaz de archivos en Sun XDR

```
const MAX = 1000;
typedef int FileIdentifier;
typedef int FilePointer;
typedef int Length;
struct Data {
        int length;
        char buffer[MAX];
};
struct writeargs {
        FileIdentifier f;
        FilePointer position;
        Data data;
};
```

```
struct readargs {
        FileIdentifier f;
        FilePointer position;
        Length length;
};

program FILEREADWRITE {
   version VERSION {
        void WRITE(writeargs)=1;
        Data READ(readargs)=2;
   }=2;
} = 9999;
```

Las interfaces en los sistemas distribuidos. En un programa distribuido, los módulos pueden lanzarse en procesos separados.
Para definir módulos, la mayoría de los LP proporcionan medios para que varios módulos puedan comunicarse entre sí: **las interfaces**

No es posible para un módulo que se ejecuta en un proceso acceder a las variables de un módulo que está en otro proceso.

Asimismo, la interfaz de un módulo escrita para RPC o RMI no puede especificar el acceso directo a variables.

Las interfaces en IDL de *CORBA* pueden especificar atributos, lo que parece violar esta regla. Sin embargo, los atributos no son accedidos directamente sino mediante ciertos procedimientos de escritura y lectura que se añaden automáticamente a la interfaz.

Parámetros de entrada:

- *S*e pasan al módulo remoto mediante el envío de los valores de los argumentos en el mensaje de petición.
- Posteriormente se proporcionan como argumentos a la operación que se ejecutará en el servidor.

Los parámetros de salida:

Se devuelven en el mensaje de respuesta y se sitúan como la respuesta de la llamada o reemplazando valores del argumento en el entorno.

Los punteros en un proceso dejan de ser válidos en el remoto. En consecuencia, no pueden pasarse punteros como argumentos o como valores retornados.

Interfaces

Interfaces de servicio: En el modelo cliente-servidor, c/ servidor proporciona procedimientos disponibles para clientes.
El término se emplea para referirse a la especificación de los procedimientos que ofrece un servidor.

Por ejemplo, un servidor de archivos **proporcionará procedimientos** para leer y escribir archivos. **Define los tipos de los argumentos de entrada y salida.**

Interfaces remotas: En el modelo de objetos distribuidos, una *interfaz remota* especifica los métodos de un objeto que están disponibles para su invocación por objetos de otros procesos, y define los tipos de los argumentos de entrada y de salida.

Sin embargo, la gran diferencia es que los métodos en las interfaces remotas pueden pasar objetos como argumentos y como resultados de los métodos.

Lenguajes de definición de interfaces.

- Mecanismo RMI con un lenguaje de programación concreto.
- Incluye una notación apropiada para definir interfaces,
- Debe permitir relacionar los parámetros de entrada y de salida con el uso habitual de los parámetros en ese lenguaje.

JavaRMI es un ejem. en el que se ha añadido un mecanismo RMI a un LP

Esta aproximación es útil cuando se puede escribir cada parte de una aplicación distribuida en el mismo lenguaje.

Es conveniente, permite al programador emplear un solo lenguaje para las invocaciones locales y remotas.

Sin embargo, muchos servicios útiles ya se encuentran escritos en C++ y otros lenguajes.

Sería beneficioso, que pudieran acceder a ellos remotamente, muchos otros programas escritos en gran variedad de lenguajes, incluyendo Java.

Los *lenguajes de definición de interfaces (IDL)*
Permiten que los objetos implementados en lenguajes diferentes se invoquen unos a otros.

Un IDL proporciona: notación para definir interfaces en la cual cada uno de los parámetros de un método se podrá describir como de *entrada* o de *salida* además de su propia especificación de tipo.

Figura 5.2
Ejemplo en CORBA IDL

```
// In file Person.idl
struct Person {
        string name;
        string place;
        long year;
} ;
interface PersonList {
        readonly attribute string listname;
        void addPerson(in Person p) ;
        void getPerson(in string name, out Person p);
        long number();
};
```

Figura 5.5
Semánticas de invocación

Medidas de tolerancia a fallos			*Semánticas De Invocación*
Retransmisión de mensajes	*Filtrado De duplicados*	*Reejecución del procedimiento O retransmisión de la respuesta*	
No	No procede	No procede	*Pudiera ser*
Si	No	Reejecutar proced.	*Al menos una vez*
Si	Si	Retransmitir resp.	*Como máximo una vez*

Semántica de la invocación RPC. Las opciones principales son:

Reintento del mensaje de petición: donde se retransmite el mensaje de petición hasta que, o bien se recibe una respuesta o se asume que el servidor ha fallado.

Filtrado de duplicados: cuando se emplean retransmisiones, si se descartan las peticiones duplicadas en el servidor .

Retransmisión de resultados: si se mantiene una historia de los mensajes de resultados para permitir retransmitir los resultados perdidos sin reejecutar las operaciones en el servidor.

Semántica de invocación "*pudiera ser*": el que invoca no puede decir si un método se ha ejecutado una vez, o ninguna en absoluto.

La semántica *pudiera ser* aparece cuando no se aplica ninguna medida de tolerancia ante fallos. Esta puede padecer de los siguientes tipos de fallo:

- ❑ Fallos de omisión si se pierde la invocación o el mensaje con el resultado
- ❑ Fallos por caída cuando el servidor que contiene el objeto remoto falla

Incógnita! Si el mensaje con el resultado no se recibe tras un timeout y no hay reintentos, no hay certeza si se ha ejecutado el método. Si se pierde el mensaje de invocación, entonces el método no se habrá ejecutado. Por otro lado, puede haberse ejecutado el método y haberse perdido el mensaje con el resultado.

Semántica de invocación *"al menos una vez":* el invocante recibe un resultado, en cuyo caso el invocante sabe que el método se evaluó al menos una vez, a menos que se reciba una excepción informando que no se recibe ningún resultado.

La semántica de invocación *al menos una vez* puede alcanzarse mediante la **retransmisión de los mensajes de petición**, que enmascara los fallos por omisión de los mensajes de invocación del resultado, La semántica *al menos una vez* puede padecer los siguientes tipos de fallos:

- Fallos por caída cuando el servidor que contiene el objeto remoto falla.
- Fallos arbitrarios. En casos donde el mensaje de invocación se retransmite, el objeto remoto puede recibirlo y ejecutar el método más de una vez, provocando que se almacenen o devuelvan valores posiblemente erróneos.

Semántica *como máximo una vez:* el invocante recibe bien un resultado, en cuyo caso el invocante sabe que el método se ejecutó exactamente una vez, o una excepción que le informa de que no se recibió el resultado, de modo que el método se habrá ejecutado o una vez o ninguna en absoluto.

La semántica de invocación *como máximo una vez* puede obtenerse utilizando todas las medidas de tolerancia frente a fallos.

Tanto Java RMI como CORBA observan la semántica de invocación *como máximo una vez,* pero CORBA permite emplear la semántica *pudiera ser* para los métodos que no devuelven resultados.

Sun RPC proporciona la semántica de llamadas *al menos una vez.*

Invocación a Método Remoto
Remote method invocation

El modelo de objetos distribuido

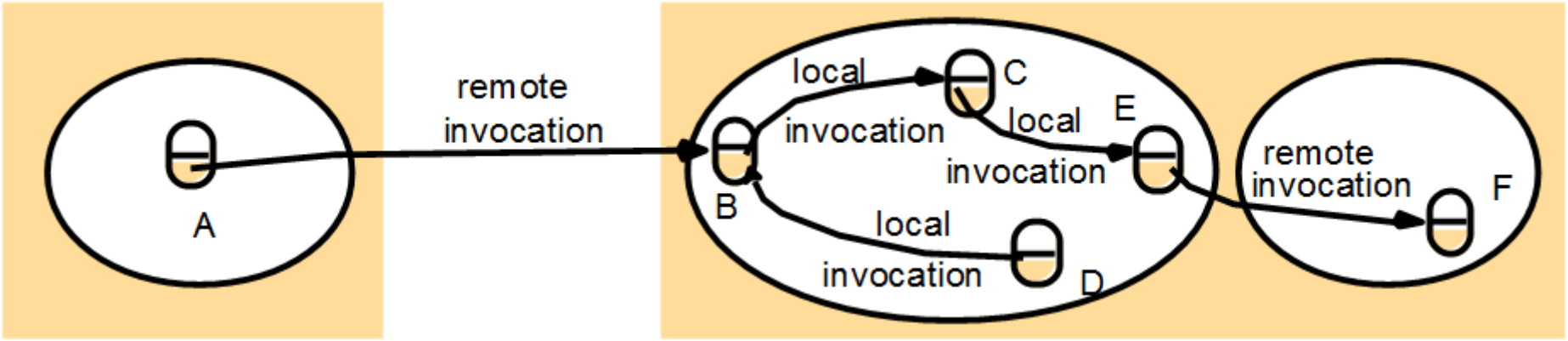
RMI está estrechamente relacionado con RPC, pero extiende al mundo de los objetos distribuidos. En RMI, un objeto llamante puede invocar un método en un objeto potencialmente remoto. Al igual que con RPC, los detalles subyacentes generalmente están ocultos para el usuario.

En RMI, un objeto llamante puede invocar un método en un objeto potencialmente remoto. Al igual que con RPC, los detalles subyacentes generalmente están ocultos para el usuario.

Invocaciones de métodos remotas.
Las invocaciones de métodos entre objetos en diferentes procesos.

Invocaciones de métodos locales.
Las invocaciones de métodos entre objetos del mismo proceso.

Figura 5.3
Invocaciones de métodos locales y remotas



Ejemplo: AppMovil solicita sonidos guardados por otro proceso.

Objetos Remotos : Objetos que pueden recibir invocaciones remotas.

B y F son objetos remotos. Todos los objetos pueden recibir invocaciones locales, a pesar de que sólo puedan recibirlas de otros objetos que posean referencias ellos. Por ejemplo, el objeto C puede tener una referencia al objeto E de modo que puede invocar uno de sus métodos.

Los 2 conceptos fundamentales sgtes son el corazón del *modelo* objetos distribuidos:

Referencia de objeto remoto: otros objetos pueden invocar los métodos de un objeto remoto si tienen acceso a su "*referencia de objeto remoto*"

Interfaz remota: c/ objeto remoto tiene una *interfaz remota* que especifica cuáles de sus métodos pueden invocarse remotamente.

Referencias a objetos remotos. permite que cualquier objeto que pueda recibir un RMI tenga una referencia a objeto remoto.

Una referencia a objeto remoto *"es un identificador que puede usarse a lo largo de todo un sistema distribuido para referirse a un objeto remoto particular único."*

Las referencias a objetos remotos son análogas a las locales en cuanto a que:

- El objeto remoto donde se recibe la invocación de método remoto se especifica mediante una referencia a objeto remoto.
- Las referencias a objetos remotos pueden pasarse como argumentos y resultados de las invocaciones de métodos remotos.

Interfaces remotas. La clase de un objeto remoto implementa los métodos de su interfaz remota. Los objetos en otros procesos pueden invocar solamente los métodos que pertenezcan a su interfaz remota.

Los objetos locales pueden invocar los métodos en la interfaz remota así como otros métodos implementados por un objeto remoto.

El sistema CORBA proporciona IDL, que permite definir interfaces remotas.

En Java RMI, las interfaces remotas se definen de la misma forma que cualquier interfaz Java.

Adquieren su capacidad de ser interfaces remotas al extender una interfaz denominada *Remote.*

Acciones en un sistema de objetos distribuido. Como en el caso no distribuido, una acción se inicia mediante la invocación de un método, que pudiera resultar en consiguientes invocaciones sobre métodos de otros objetos.

Pero en el caso distribuido, los objetos involucrados en una cadena de invocaciones relacionadas pueden estar ubicados en procesos o en computadores diferentes.

Cuando una invocación cruza los límites de un proceso o un computador, se emplea una RMI, y la referencia remota al objeto se hace disponible para hacer posible la RMI.

En la Figura 5.3, el objeto A necesita poseer una referencia a objeto remoto para el objeto B. Las referencias a un objeto remoto pueden obtenerse como resultado de una invocación a un objeto remoto. Por ejemplo, el objeto A podría obtener una referencia remota al objeto F desde el objeto E.

Excepciones. Cualquier invocación remota puede fallar por razones relativas a que el objeto invocado está en un proceso o computador diferente de la del objeto que lo invoca. Por ejemplo, el proceso que contiene el objeto remoto pudiera malograrse o estar demasiado ocupado para responder, o pudiera perderse el mensaje resultante de la invocación.

Es así, que una invocación a un método remoto debiera ser capaz de lanzar excepciones tales como ***timeouts*** debidos a la distribución así como aquellos lanzados durante la ejecución del método invocado. Ejemplos de esto último son: un intento de lectura pasado el fin de un archivo, o un acceso a un archivo sin los permisos adecuados.

CORBA IDL proporciona una notación para las excepciones específicas del nivel de aplicación, y el sistema subyacente genera excepciones estándar cuando ocurren errores debidos a la distribución. Los programas clientes CORBA deben ser capaces de gestionar las excepciones. Por ejemplo, un programa cliente C++ empleará los mecanismos de excepciones de C++.

Figura 5.4
Un objeto remoto y su interfaz remota

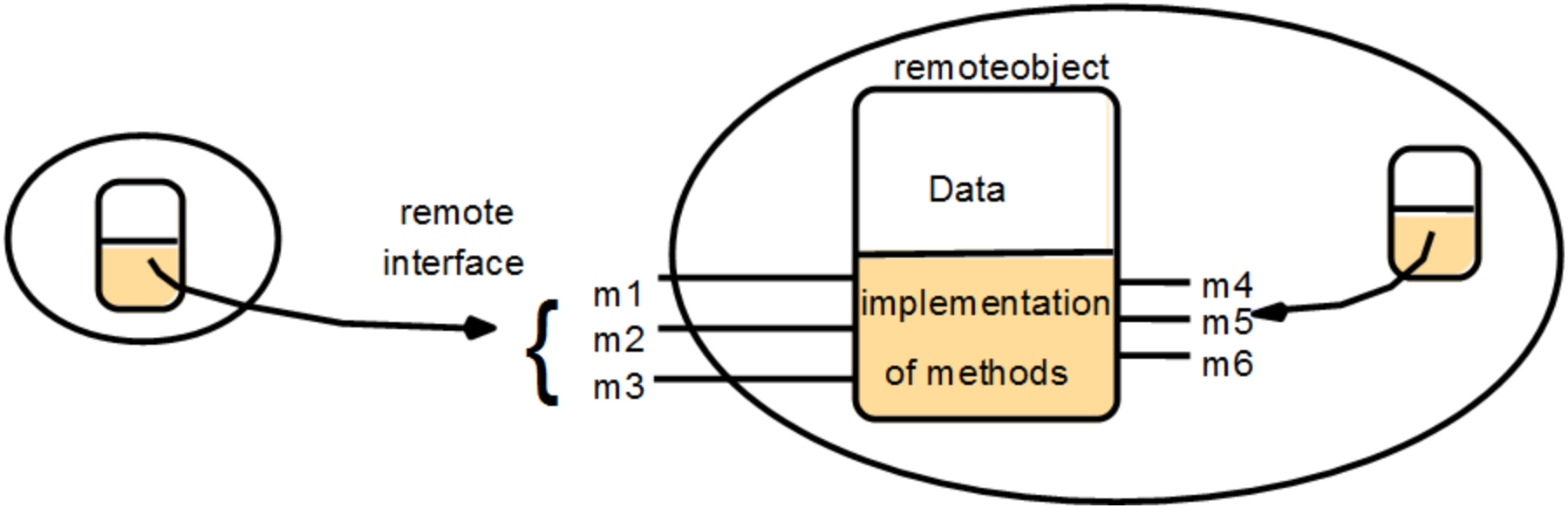
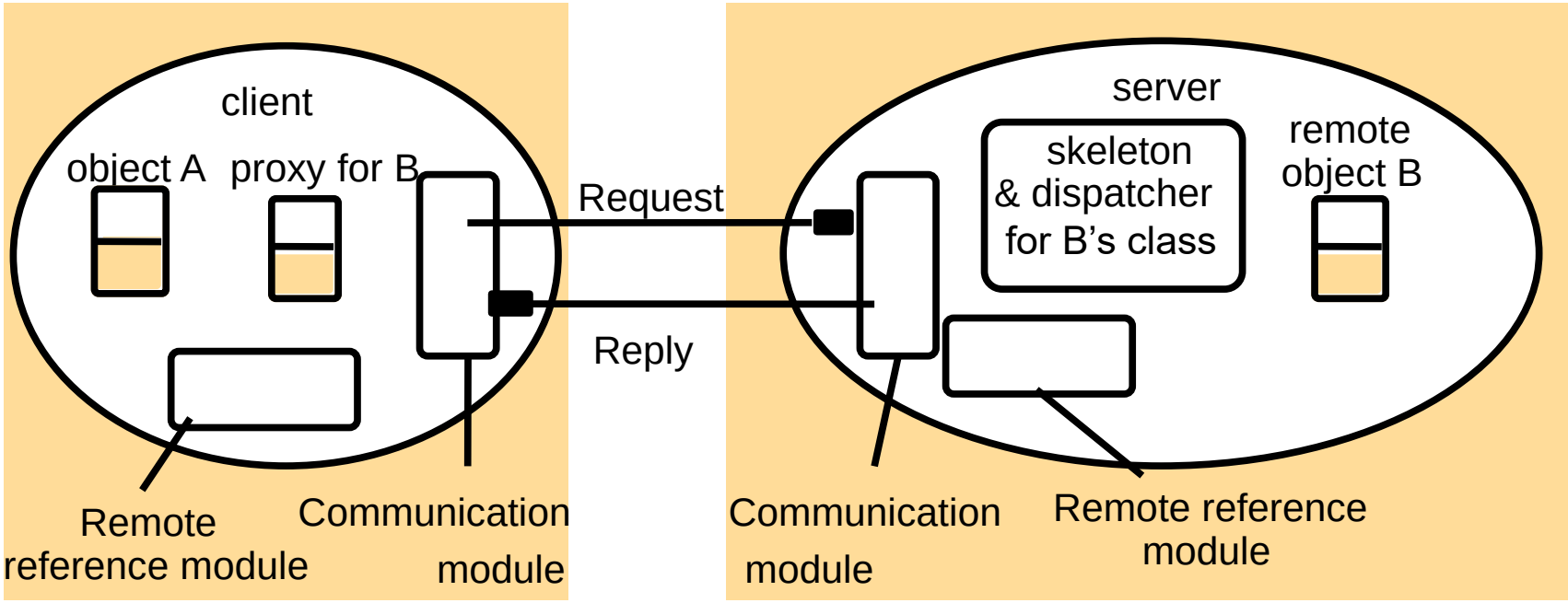


Figura 5.6
El papel de un proxy y un esqueleto en la invocación de métodos remotos



Módulo de comunicación. Los dos módulos cooperantes realizan el protocolo de petición‐respuesta, que retransmite los mensajes de *petición* y *respuesta* entre el cliente y el servidor.

El módulo de comunicación emplea sólo los tres primeros elementos, que especifican: el tipo de mensaje, su *idPeticion* y la referencia remota del objeto que se invoca. El *idMetodo* y todo el empaquetado y desempaquetado es cuestión del software RMI.

Los módulos de comunicación son responsables conjuntamente de proporcionar una semántica de invocación, por ejemplo *como máximo una vez.*

El módulo de comunicación en el servidor selecciona el distribuidor para la clase del objeto que se invoca, pasando su referencia local, que se obtiene del módulo de referencia remota en respuesta al identificador de objeto remoto en el mensaje *petición.*

Módulo de referencia remota.

- Traduce las referencias entre objetos locales y remotos.
- Crea referencias a objetos remotos.

En c/ proceso, el módulo tiene una ***tabla de objetos remotos*** que almacena la correspondencia entre referencias a objetos locales en ese proceso y las referencias a objetos remotos (cuyo ámbito es todo el sistema).

La tabla incluye:

. Una entrada para todo objeto remoto implementado por el proceso. Por ejemplo, en la Figura 5.6, el objeto remoto B estará registrado en la tabla del servidor .

. Una entrada para cada proxy local. Por ejemplo, en la Figura 5.6 el proxy para B estará registrado en la tabla de ese cliente.

Las acciones del módulo de referencia remota ocurren como sigue:

- Cuando se pasa un objeto remoto por primera vez, como argumento o resultado, se le pide al módulo de referencia remota que cree una referencia a un objeto remoto, que se añade a su tabla.

- Cuando llega una referencia a un objeto remoto, en un mensaje de petición o respuesta, se le pide al módulo de referencia remota la referencia al objeto local correspondiente, que se referirá bien a un proxy o a un objeto remoto. En el caso de que el objeto remoto no esté en la tabla, el software RMI crea un nuevo proxy y pide al módulo de referencia remota que lo añada a la tabla.

Los componentes del módulo software de RMI llaman a este módulo cuando realizan el empaquetado y el desempaquetado de las referencias a objetos remotos. Por ejemplo, cuando llega un mensaje de petición, se emplea su tabla para encontrar qué objeto local se va a invocar.

El **software de RMI** consiste en una capa de software entre:

* los objetos del nivel de aplicación y
* los módulos de comunicación y de referencia remota.

Los papeles de los objetos de middleware mostrados en la Figura 5.6 son como sigue:

Proxy: *hace que la* invocación al método remoto sea transparente para los clientes. Para ello se comporta como un objeto local para el que invoca; pero en lugar de ejecutar la invocación, dirige el mensaje al objeto remoto.

Hay un proxy para cada objeto remoto del que el cliente disponga de una referencia de objeto remoto. La clase de un proxy implementa los métodos de la interfaz remota del objeto remoto al que representa. Esto asegura que las invocaciones al método remoto son adecuadas según el tipo del objeto remoto.

Cada Servidor tiene un Distribuidor y un Esqueleto para cada clase que represente a un objeto remoto.

Distribuidor: El distribuidor recibe el mensaje de *petición* desde el módulo de comunicación. Emplea el *idMetodo* para seleccionar el método apropiado del esqueleto, pasándole el mensaje de *petición.* El distribuidor y el proxy emplean los mismos métodos de asignación de cada *idMetodo* para los métodos de la interfaz remota.

Esqueleto: la clase de un objeto remoto tiene un *esqueleto,* que implementa los métodos interfaz remota. Se encuentran implementados de forma muy diferente de los métodos del objeto remoto. Un método del esqueleto desempaqueta los argumentos del mensaje de *petición,* invoca el método correspondiente en el objeto remoto.

Referencias a objetos remotos.

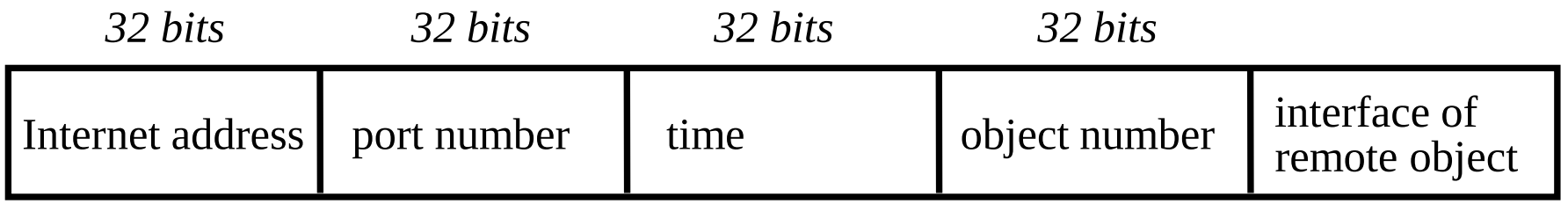
Cuando un cliente invoca un método en un objeto remoto, se envía un mensaje de invocación al proceso servidor que alberga el objeto remoto.

Este mensaje necesita especificar el objeto particular cuyo método se va a invocar.

Una *referencia a un objeto remoto* es un identificador para un objeto remoto que es válida a lo largo y ancho de un sistema distribuido.

En el mensaje de invocación se incluye una referencia a objeto remoto que especifica cuál es el objeto invocado.

Figura 4.10
Representación de una referencia de objeto remoto.



Práctica de Laboratorio

1) Servidor de aplicaciones JEE

Guía de clase práctica:

- http://www.educa.una.py/politecnica/mod/page/view.php?id=68405

2) Objetos distribuidos utilizando JavaRMI

Guía de clase práctica:

- http://www.educa.una.py/politecnica/mod/page/view.php?id=68406

Código fuente:

- https://gitlab.com/fmancia/sd/tree/master/lab-rmi