

Unidad 4

Objetos distribuidos e invocación remota

Sistemas Distribuidos

Universidad Nacional de Asunción, Facultad Politécnica

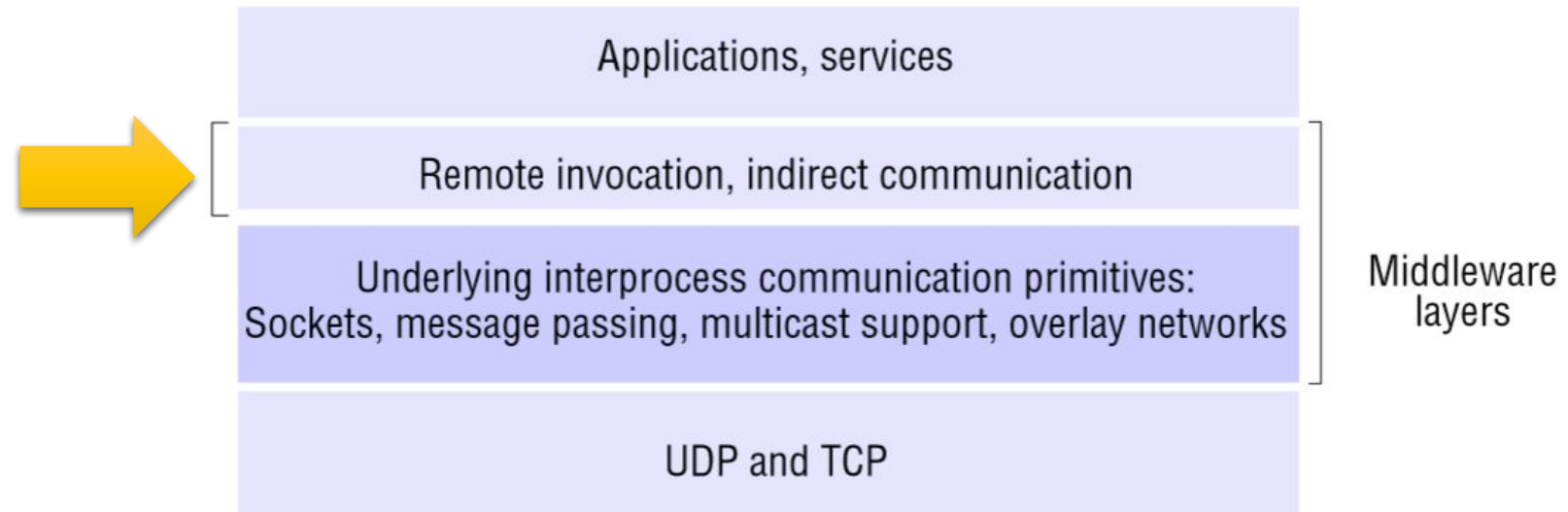
Ingeniería Informática

Ing. Fernando Mancía, Ing. Juan Ferreira

Contenido teórico basado en el capítulo 5 y 8 del libro: George Coulouris, Jean Dollimore, Tim Kindberg, and Gordon Blair. (2011). Distributed Systems: Concepts and Design (5th. ed.). Addison-Wesley. Publishing Company, USA.

Contenido

- Invocación Remota
- Protocolos Petición-Respuesta
- RPC/RMI



Conceptos

- **Aplicación Distribuida:** Aplicaciones que se componen de programas cooperantes corriendo en procesos distintos. Es necesario invocar operaciones en otros procesos, generalmente en computadores diferentes.
- **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”***.
- Se puede tener implementaciones en lenguajes diferentes.

Introducción

Un Middleware proporciona

- **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
- **Protocolos de comunicación:** Los protocolos del middleware no dependen de los protocolos de transporte subyacentes.
- **Hardware de los computadores:** Se emplean en el empaquetado y desempaquetado de mensajes.
- **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.

Introducción

- Los conceptos de middleware a alto nivel se centran en cómo los procesos se comunican en un sistema distribuido.
- Hay tres paradigmas de invocación remota:
 - **Protocolos de petición-respuesta**, un modo relativamente a bajo nivel para ejecutar una operación remota. Sienta las bases para RPC y RMI
 - **Remote procedure call (RPC)**, es el primer modelo (y el más conocido) para facilitar las llamadas a servidores remotos.
 - **Remote method invocation (RMI)**, extensión de RPC para la llamada a métodos de objetos en nodos remotos.

Protocolo Petición Respuesta

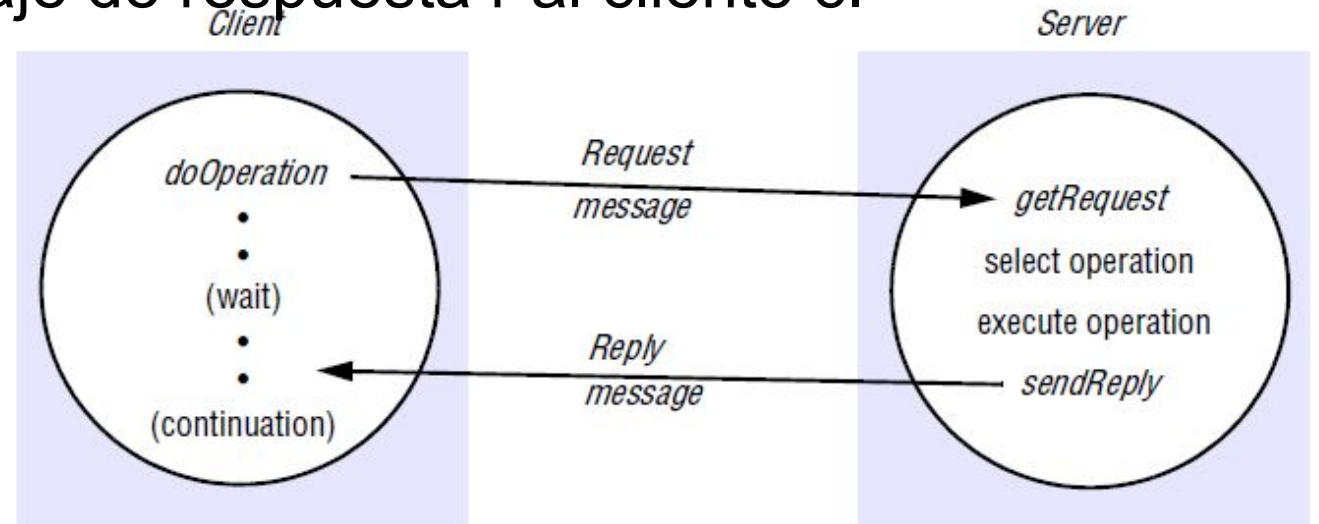
- Comunicación cliente-servidor, está orientada a soportar los roles y el intercambio de mensajes de las interacciones típicas cliente-servidor.
- En el caso normal, la comunicación petición-respuesta es síncrona, ya que el proceso cliente se bloquea hasta que llega la respuesta del servidor.
- Esta 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 más tarde.

Protocolo Petición Respuesta Primitivas



Se basa en tres primitivas de comunicación:

- ***doOperation(s,args)***: invoca una operación remota en s, con los argumentos indicados en args (parámetros y tipo de operación).
- ***getRequest***: espera/adquiere una petición en el servidor.
- ***sendReply(r,c)***: manda el mensaje de respuesta r al cliente c.



Protocolo Petición Respuesta Primitivas



public byte[] doOperation (RemoteRef s, int operationId, byte[] arguments)

Sends a request message to the remote server and returns the reply. The arguments specify the remote server, the operation to be invoked and the arguments of that operation.

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.

Protocolo Petición Respuesta Primitivas



- Información transmitida en un mensaje de solicitud o respuesta.

Figure 5.4 Request-reply message structure

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

Protocolo Petición Respuesta

Modelo de Fallos



- Si las tres primitivas ***doOperation***, ***getRequest***, ***sendReply*** se implementan con datagramas UDP, **adolecerán de los mismos fallos de comunicación** que cualquier otro ejemplo de aplicación de UDP. Esto es:
 - **Fallos por omisión:** el servidor no responde o el mensaje se pierde
 - **Bizantino:** Los mensajes pueden llegar duplicados y desordenados
- Además, el protocolo puede padecer el fallo de los procesos.

Protocolo Petición Respuesta

Modelo de Fallos



Modelo de Fallo por Omisión: Estrategia de *timeout* en *doOperation()*.

- Opción 1: la operación *doOperation* simplemente retorne un indicador de fallo.
- Opción 2: reintentar *doOperacion* de forma repetida hasta se obtenga una respuesta.

Modelo de Fallo Bizantino:

- Opción 1: Puede implementar un mecanismo para descartarlas si todavía está realizando la operación. Esto es posible con el identificador de petición:

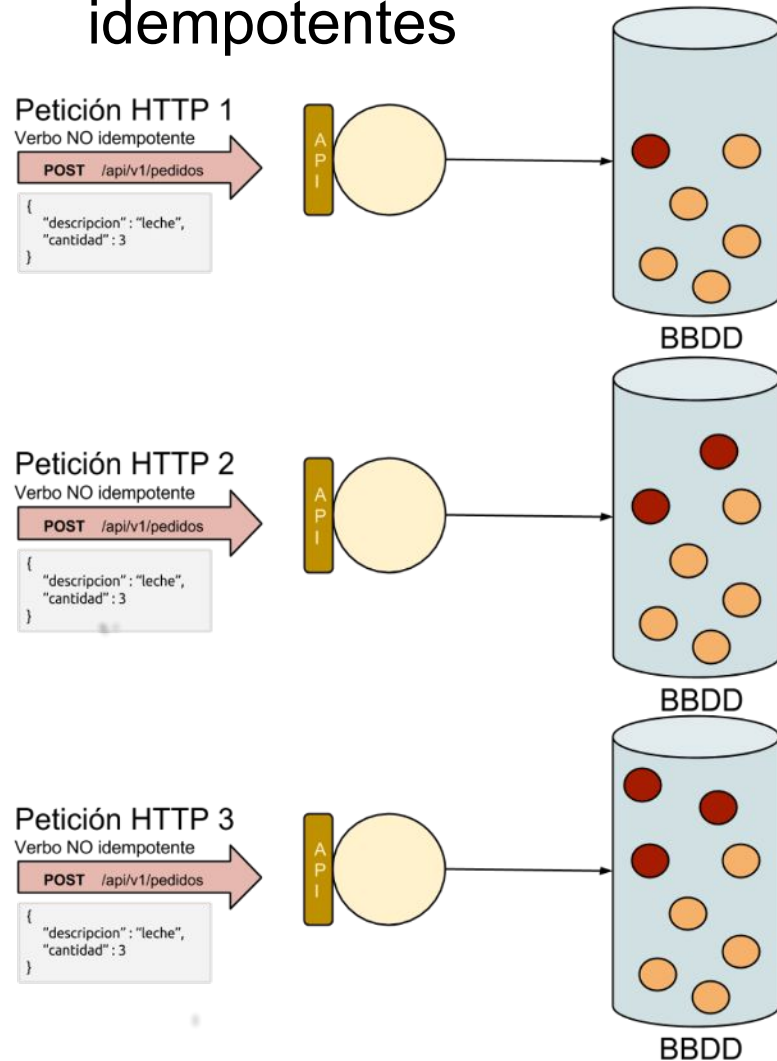
`id_cliente # id_peticion`

- Opción 2: Si ya la ha realizado y la operación es idempotente, la realiza de nuevo.
 - Si no, puede implementar un histórico con los resultados de las operaciones realizadas.

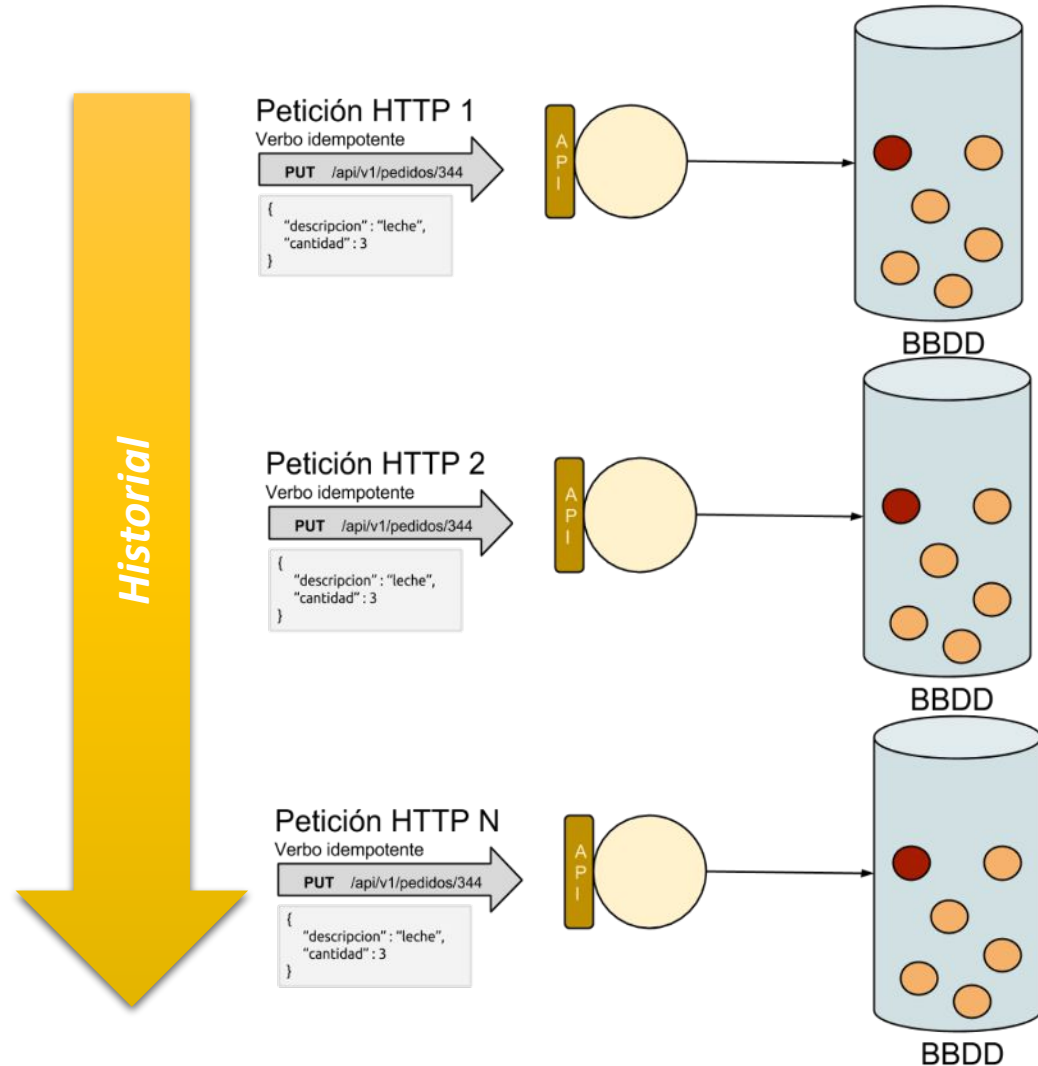
Protocolo Petición Respuesta

Idempotencia

- Operaciones no idempotentes



- Operaciones idempotentes



Protocolo Petición Respuesta

Estilos de protocolos 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 protocolo petición-respuesta (RR).** 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.

Protocolo Petición Respuesta

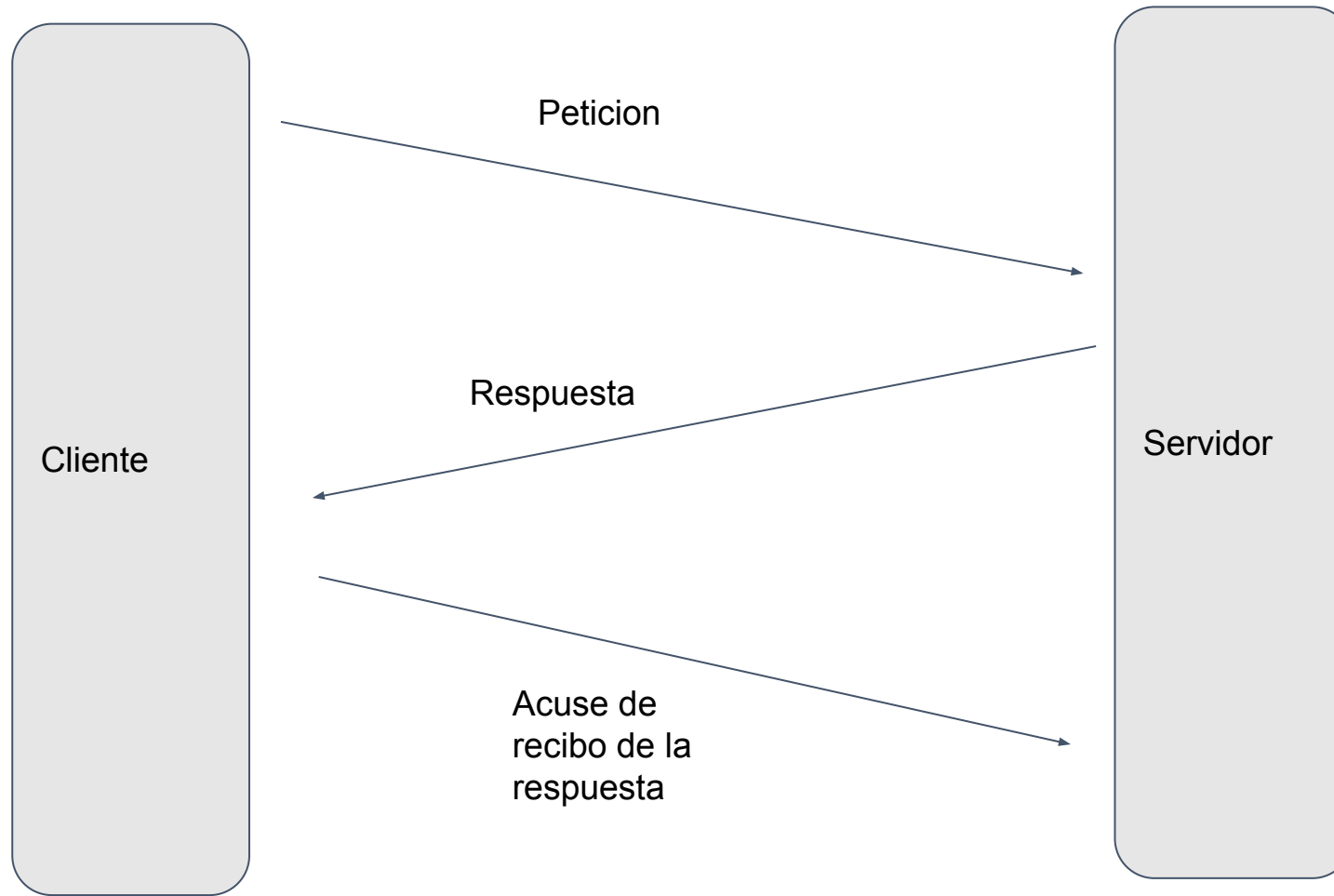
Estilos de protocolos RPC



- 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 protocolo petición-respuesta (RR).** 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.

Protocolo Petición Respuesta

Estilos de protocolos RPC



Protocolo Petición Respuesta

Estilos de protocolos RPC



<i>Name</i>	<i>Messages sent by</i>		
	<i>Client</i>	<i>Server</i>	<i>Client</i>
R	<i>Request</i>		
RR	<i>Request</i>	<i>Reply</i>	
RRA	<i>Request</i>	<i>Reply</i>	<i>Acknowledge reply</i>

Protocolo petición-respuesta

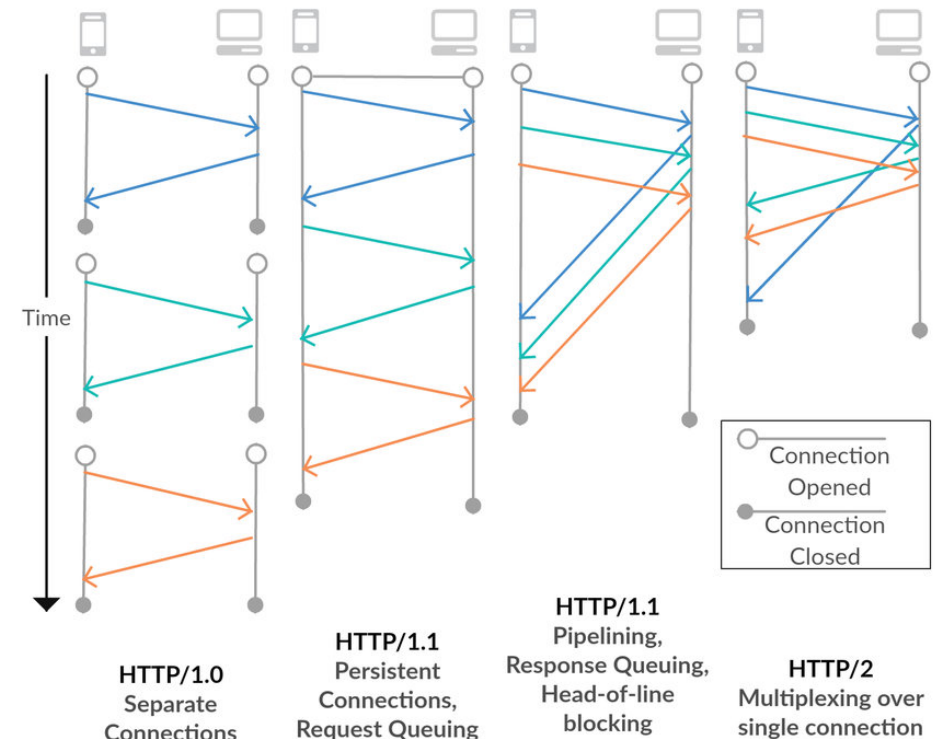
HTTP: un ejemplo de protocolo RR

- **HTTP:** basado en protocolo petición – respuesta.
 - Cliente: Navegador.
 - Servidor: WebServer.
- HTTP Especifica los mensajes que involucran métodos, los argumentos, resultados y reglas para ensamblarlos en los mensajes.
- Soporta un conjunto fijo de métodos (GET, PUT, POST, etc.) que son aplicables a todos los recursos. Al contrario de otros protocolos, cada objeto tiene sus propios métodos. Además permite:
 - **Negociación del contenido:** las peticiones de los clientes pueden incluir información (por ejemplo lenguaje o tipo de medio),
 - **Autenticación:** se utilizan credenciales y desafíos para conseguir una autenticación del estilo clave de acceso.

Protocolo petición-respuesta

HTTP: un ejemplo de protocolo RR

- 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.



Protocolo petición-respuesta

HTTP: un ejemplo de protocolo RR

- **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.

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.
 - Admite argumentos al URL

Protocolo petición-respuesta

HTTP: un ejemplo de protocolo RR

- **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 los datos proporcionados en el cuerpo de la petición. Este método está diseñado para:
 - Proporcionar un bloque de datos (por ejemplo los obtenidos en un formulario) a un proceso de 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.

Protocolo petición-respuesta

HTTP: un ejemplo de protocolo RR

- **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.

Más sobre métodos HTTP: <https://assertible.com/blog/7-http-methods-every-web-developer-should-know-and-how-to-test-them>

Protocolo petición-respuesta

Mensaje HTTP



Mensaje HTTP request

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

Mensaje HTTP reply

<i>HTTP version</i>	<i>status code</i>	<i>reason</i>	<i>headers</i>	<i>message body</i>
HTTP/1.1	200	OK		resource data

Protocolo petición-respuesta

Códigos HTTP

Códigos de estado de respuesta HTTP indican el estado (exitoso o no) de una petición HTTP. Las respuestas se agrupan en cinco clases:

1. Respuestas informativas (100–199),
2. Respuestas satisfactorias (200–299),
3. Redirecciones (300–399),
4. Errores de los clientes (400–499),
5. Errores de los servidores (500–599).

Code	Reason-Phrase	Defined in...
100	Continue	Section 6.2.1
101	Switching Protocols	Section 6.2.2
200	OK	Section 6.3.1
201	Created	Section 6.3.2
202	Accepted	Section 6.3.3
203	Non-Authoritative Information	Section 6.3.4
204	No Content	Section 6.3.5
205	Reset Content	Section 6.3.6
206	Partial Content	Section 4.1 of [RFC7233]
300	Multiple Choices	Section 6.4.1
301	Moved Permanently	Section 6.4.2
302	Found	Section 6.4.3
303	See Other	Section 6.4.4
304	Not Modified	Section 4.1 of [RFC7232]
305	Use Proxy	Section 6.4.5
307	Temporary Redirect	Section 6.4.7
400	Bad Request	Section 6.5.1
401	Unauthorized	Section 3.1 of [RFC7235]
402	Payment Required	Section 6.5.2
403	Forbidden	Section 6.5.3
404	Not Found	Section 6.5.4
405	Method Not Allowed	Section 6.5.5
406	Not Acceptable	Section 6.5.6
407	Proxy Authentication Required	Section 3.2 of [RFC7235]
408	Request Timeout	Section 6.5.7
409	Conflict	Section 6.5.8
410	Gone	Section 6.5.9
411	Length Required	Section 6.5.10
412	Precondition Failed	Section 4.2 of [RFC7232]
413	Payload Too Large	Section 6.5.11
414	URI Too Long	Section 6.5.12
415	Unsupported Media Type	Section 6.5.13
416	Range Not Satisfiable	Section 4.4 of [RFC7233]
417	Expectation Failed	Section 6.5.14
426	Upgrade Required	Section 6.5.15
500	Internal Server Error	Section 6.6.1
501	Not Implemented	Section 6.6.2
502	Bad Gateway	Section 6.6.3
503	Service Unavailable	Section 6.6.4
504	Gateway Timeout	Section 6.6.5
505	HTTP Version Not Supported	Section 6.6.6

Códigos importantes del protocolo HTTP

Referencia:

<https://datatracker.ietf.org/doc/html/rfc7231#section-6.1>

Atención especial:

200, 301, 400, 403, 404, 408, 409, 500, 504

Llamada a Procedimiento Remoto RPC



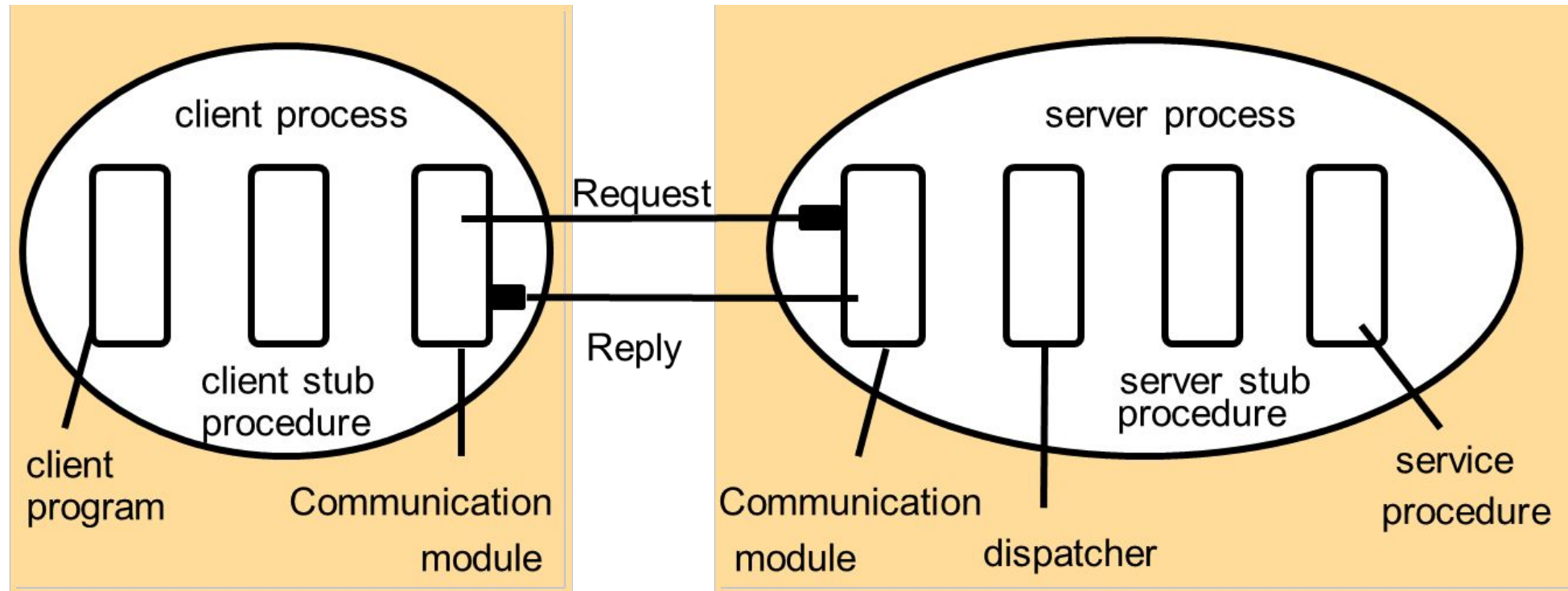
- La llamada a procedimiento remoto (RPC) representa un adelanto en computación distribuida, con el objetivo de hacer que la programación de sistemas distribuidos parezca similar, si no idéntica, a la programación convencional.
- Logra un alto nivel de transparencia de distribución. Esta unificación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

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



Interfaces

- **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.

Interfaces

Parámetros de entrada:

- Se 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.

Interfaces

- **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.

Interfaces

- 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.

Semánticas de invocación



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

Semánticas de invocación

- Medidas de tolerancia a fallo:
 - ***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.

Semánticas de invocación

- **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.

Semánticas de invocación

- **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.

Semánticas de invocación

- **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**.

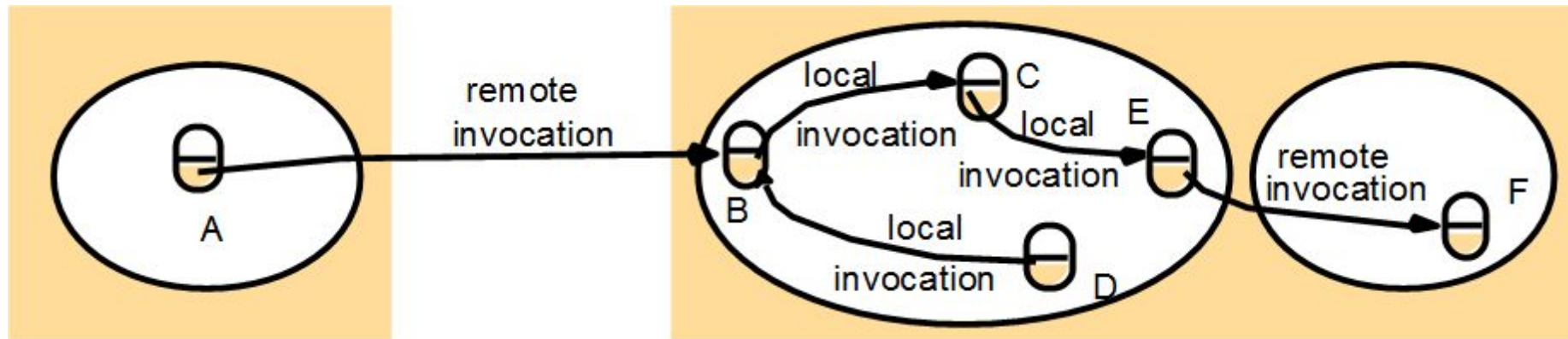
- Invocación a Método Remoto
- Remote method invocation

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.

Invocaciones de métodos locales y remotas

- Ejemplo: AppMovil solicita sonidos guardados por otro proceso.



El modelo de objetos distribuido

- **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 siguientes 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.

El modelo de objetos distribuido

- 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.

El modelo de objetos distribuido

- **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***.

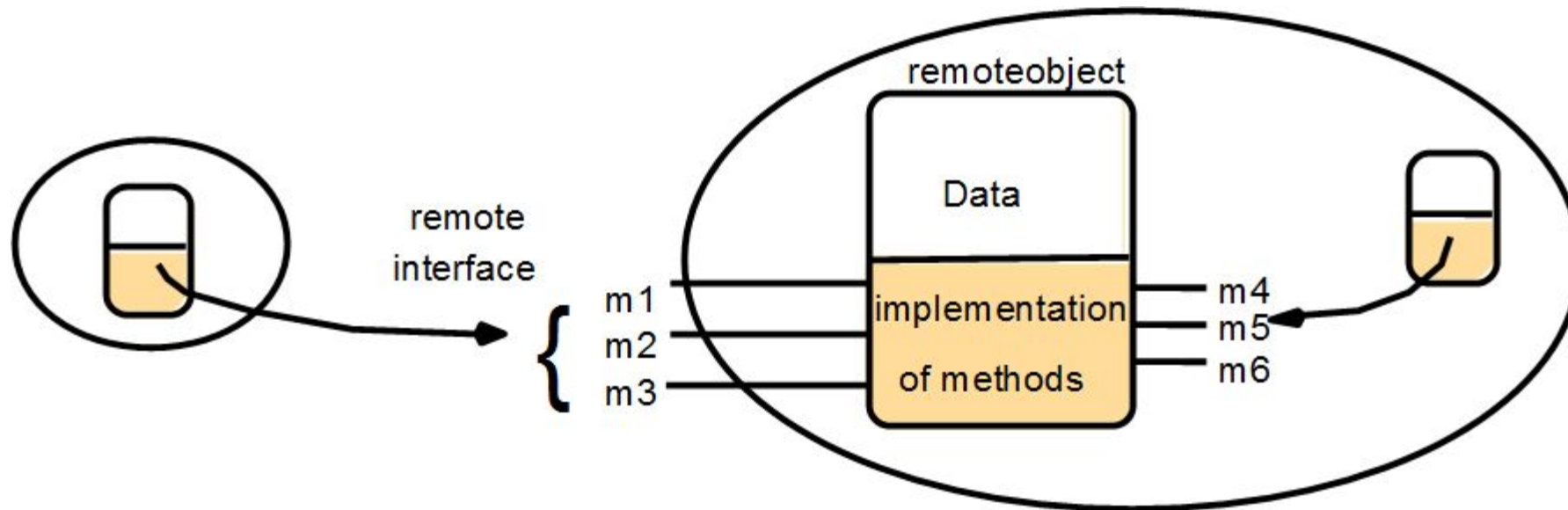
El modelo de objetos distribuido

- **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.

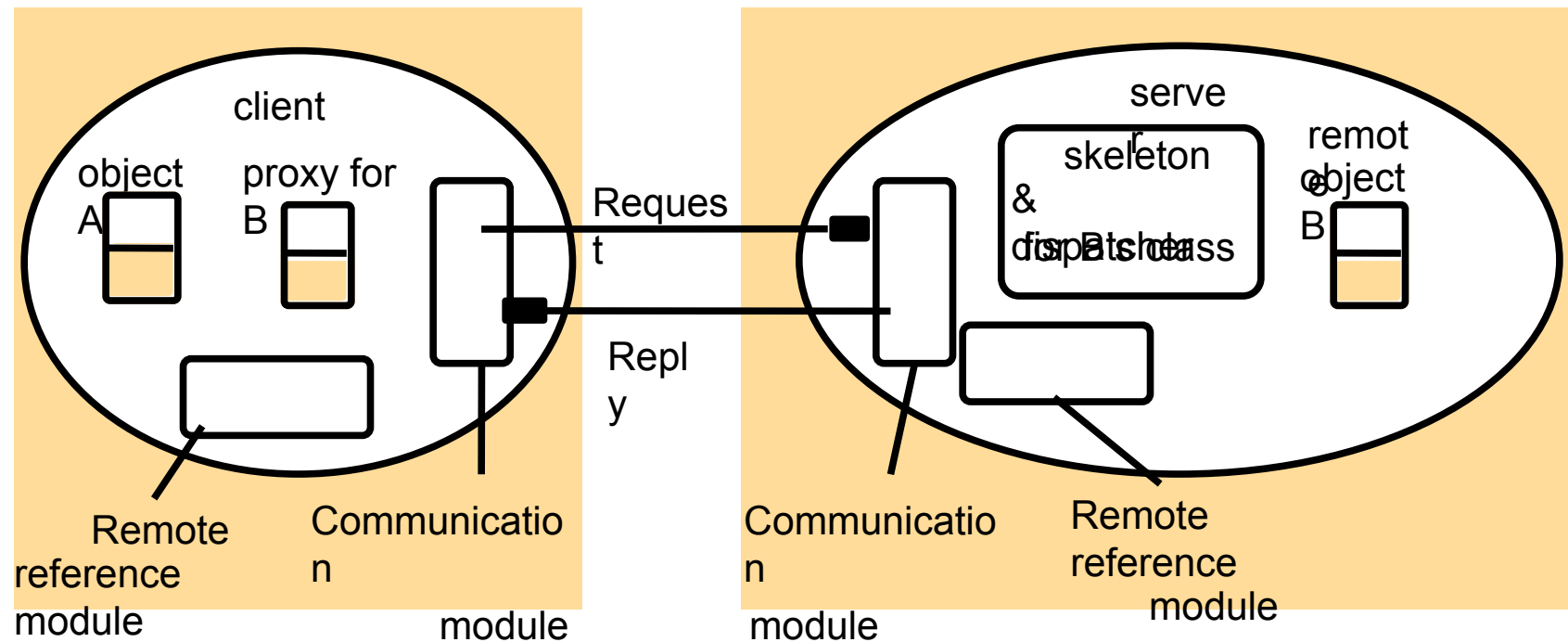
El modelo de objetos distribuido

- **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



Implementación de RMI

- **Módulo de comunicación.** Los dos módulos cooperantes realizan el protocolo de peticiónrespuesta, 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 ***idPetición*** 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.

Implementación de RMI

- **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.

Implementación de RMI

- 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.

Implementación de RMI

- **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.

Implementación de RMI

- *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.

Apache Kafka <https://kafka.apache.org/>

Apache Kafka es una plataforma de mensajería de suscripción/publicación, transmisión de eventos distribuida de código abierto utilizado ampliamente para encaminar datos de alto rendimiento, análisis de transmisión, integración de datos y aplicaciones de misión crítica.

- Intermediario de mensajería (broker de mensajes de alta carga)
- Captura de los cambios de estado de una aplicación
- Seguimiento de actividad de sitios web
- Procesamiento de flujo de datos

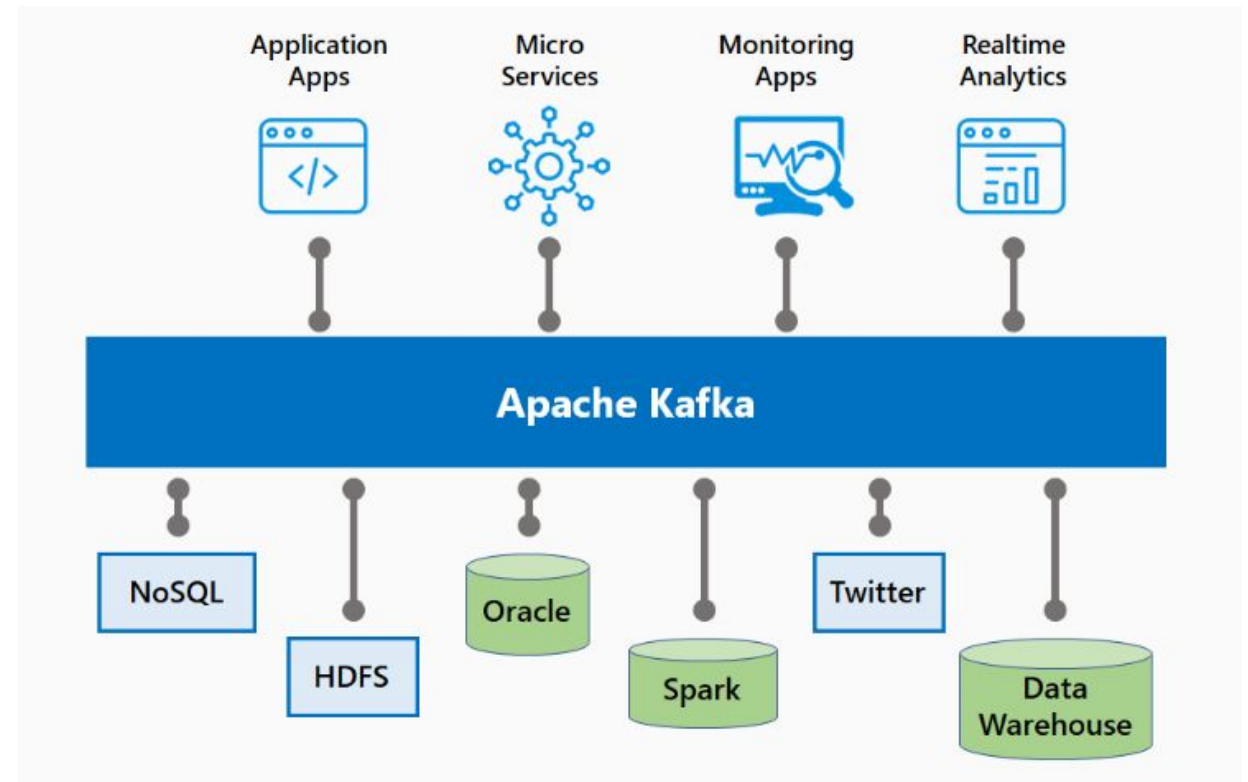


Imagen obtenida de: <https://www.gavstech.com/all-about-apache-kafka-an-evolved-log/>

Práctica de Laboratorio

Servidor de aplicaciones JavaEE

- Guía de clase práctica:
 - <https://www.educa.una.py/politecnica/mod/page/view.php?id=129954>

Objetos distribuidos utilizando JavaRMI

- Guía de clase práctica:
 - <https://www.educa.una.py/politecnica/mod/page/view.php?id=129953>

Código fuente:

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