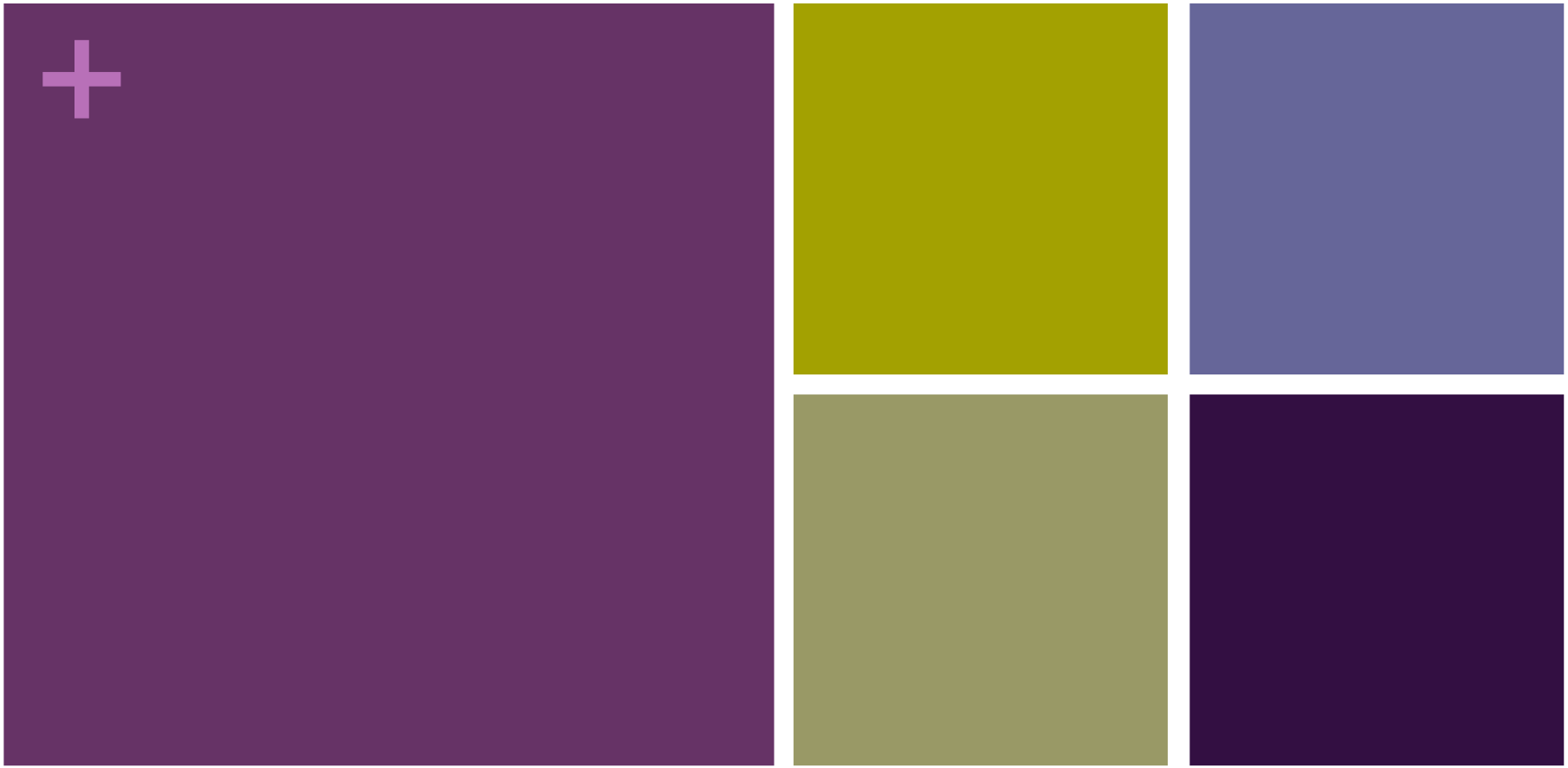


Sistemas Distribuidos
Middleware

Rodrigo Santamaría

Middleware

- Introducción
- Middleware de bajo nivel
- Invocación remota
- Servicios web

Introducción

- **Middleware**: es una capa software que:
  - Enmascara la heterogeneidad de la red subyacente, el hardware, el sistema operativo y los lenguajes de programación
  - Ofrece un modelo computacional para los programadores de aplicaciones distribuidas

Introducción

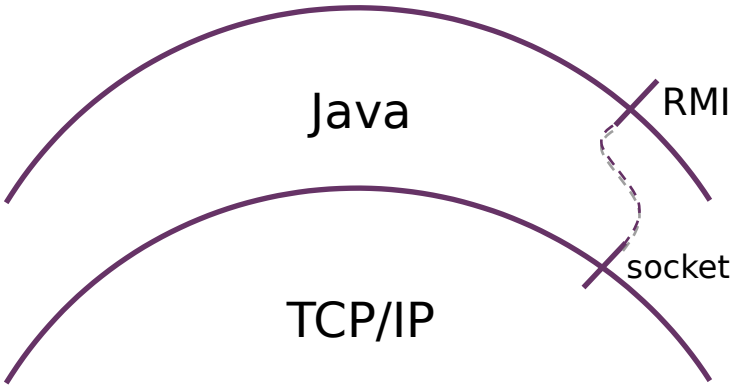
UDP y TCP

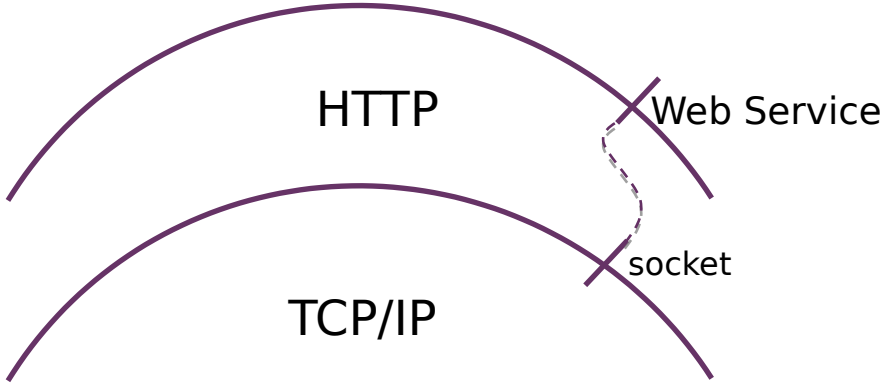
- Protocolos de nivel de transporte de Internet
  - Ambos permiten un paso de mensajes básico
  - UDP: con fallos por omisión
  - TCP: garantiza la entrega en condiciones normales, pero al coste de una bajada en el rendimiento
    - Generalmente, usamos TCP sobre IP, o TCP/IP
- Aunque TCP/UDP abstraen del nivel de red, no abstraen perfectamente de los niveles de hardware y ssoo
  - Distinto almacenamiento de números (little endian/big endian)
  - Distinta codificación de caracteres (ASCII/Unicode)
  - Pérdida de mensajes, seguridad, enrutado, etc.

Introducción

Middleware vs TCP/IP

- **TCP/IP**: transmisión de bytes *mediante* de TCP/UDP+IP
- **Middleware**: transmisión de mensajes *sobre* TCP/UDP+IP





https://www.quora.com/What-is-the-difference-between-a-socket-and-a-web-service
http://blog.arungupta.me/rest-vs-websocket-comparison-benchmarks/

Introducción

Tipos de middleware

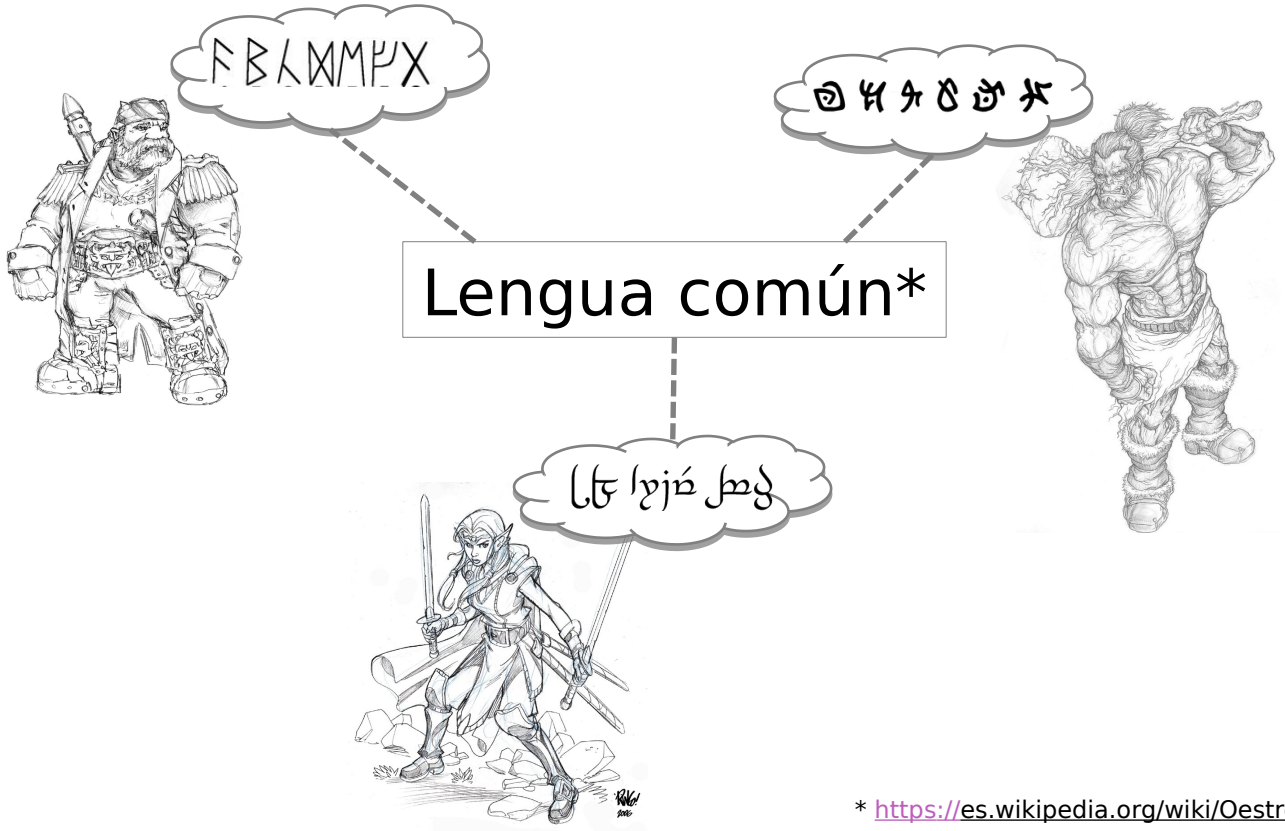
- Bajo nivel
  - Ofrece funcionalidades esenciales, generalmente relacionadas con cambios sobre el soporte básico TCP/UDP + IP
    - Representación independiente de SSOO (marshalling)
    - Multicast sobre IP
    - Red virtual sobre la red IP subyacente
- Alto nivel (invocación remota)
  - Centrada en el envío/recepción de datos
    - Remote Procedure Call (RPC)
    - Remote Method Invocation (RMI)

Middleware

- Introducción
- Middleware de bajo nivel
  - Marshalling
  - Redes superpuestas
- Invocación remota
- Servicios web

Marshalling

- Proceso de ensamblado de datos de forma que puedan ser convenientemente transmitidos en un mensaje
  - Traducción de los valores locales a una representación externa



* https://es.wikipedia.org/wiki/Oestron

Marshalling

CORBA CDR

- CORBA CDR se define en CORBA 2.0 (2004)
- Permite representar los tipos de datos manejados por CORBA
- El tipo de dato no está incluido en la representación del dato
  - Requiere conocimiento de los tipos por ambas partes

index in sequence of bytes	← 4 bytes →	*notes on representation*
0–3	5	*length of string*
4–7	"Smit"	*'Smith'*
8–11	"h___"	
12–15	6	*length of string*
16–19	"Lond"	*'London'*
20–23	"on__"	
24–27	1984	*unsigned long*

Colouris et al. 2011

Representación en CDR de una estructura `Person{string, string, long}` para el valor `{'Smith', 'London', 1984}`

Marshalling

Serialización Java

- Permite representar objetos Java de un modo adecuado para almacenarlos en disco o transmitirlos
- La representación incluye
  - Estado del objeto
  - Definición de su clase (nombre, versión)
  - Definición de otras clases (heredadas, implementadas) a través de referencias o *handles*

Marshalling

Serialización Java

```
public class Person implements Serializable {
  private String name;
  private String place;
  private int year;
  public Person(String aName, String aPlace, int aYear) {
    name = aName;
    place = aPlace;
    year = aYear;
   }
 // followed by methods for accessing the instance variables
 }
```

```
Person p = new Person("Smith", "London", 1984);
```

Colouris et al. 2011

Serialized values				*Explanation*
Person	8-byte version number		h0	*class name, version number*
3	int year	java.lang.String name	java.lang.String place	*number, type and name of instance variables*
1984	5 Smith	6 London	h1	*values of instance variables*

The true serialized form contains additional type markers; h0 and h1 are handles.

Marshalling

Extensible Markup Language (XML)

- Definido por el W3C para su uso general en Internet[1]
- Modo de codificación textual que representa tanto el contenido como los detalles de su estructura o apariencia
- Cada dato en un fichero XML va etiquetado
  - Las etiquetas pueden servir para definir la estructura
  - O para asociar pares atributo-valor

```
<person id="123456789">
    <name>Smith</name>
    <place>London</place>
    <year>1984</year>
    <!-- a comment -->
</person >
```

id es un atributo
person, *name*, *place* o *year* son elementos
name, *place* y *year* son elementos contenidos en *person*

[1]Proviene, junto con HMTL, de SGML, un lenguaje de marcas muy complicado. HTML se diseñó para definir la apariencia de las páginas web, mientras que XML se diseñó para escribir documentos estructurados para la web

Marshalling

JavaScript Object Notation (JSON)

- Lenguaje de formato de datos de estándar abierto
- Creado por la necesidad de compartir datos entre cliente y servidor sin recurrir a lenguajes de programación[1]
- Sintaxis 'aligerada' respecto a XML

```
{"id":"123456789",
 "name":"Smith",
 "place":"London",
 "year":1984
}
```

Requiere 66 caracteres para representar la misma información que el XML (113), un 40% de reducción

¿la misma información?

- Uso de corchetes [] para representar listas y llaves {} para representar diccionarios/objetos

[1]JavaScript no tiene un tipado de datos fuerte, lo que requería un formato externo que representara estructuras de datos complejas

Marshalling

Comparativa

Marshalling	Conocimiento previo	Tipos de datos
CORBA	Sí	Cualquiera del lenguaje
Java	No*	Cualquiera del lenguaje
XML	No	Texto
JSON	No	Texto (ligero)

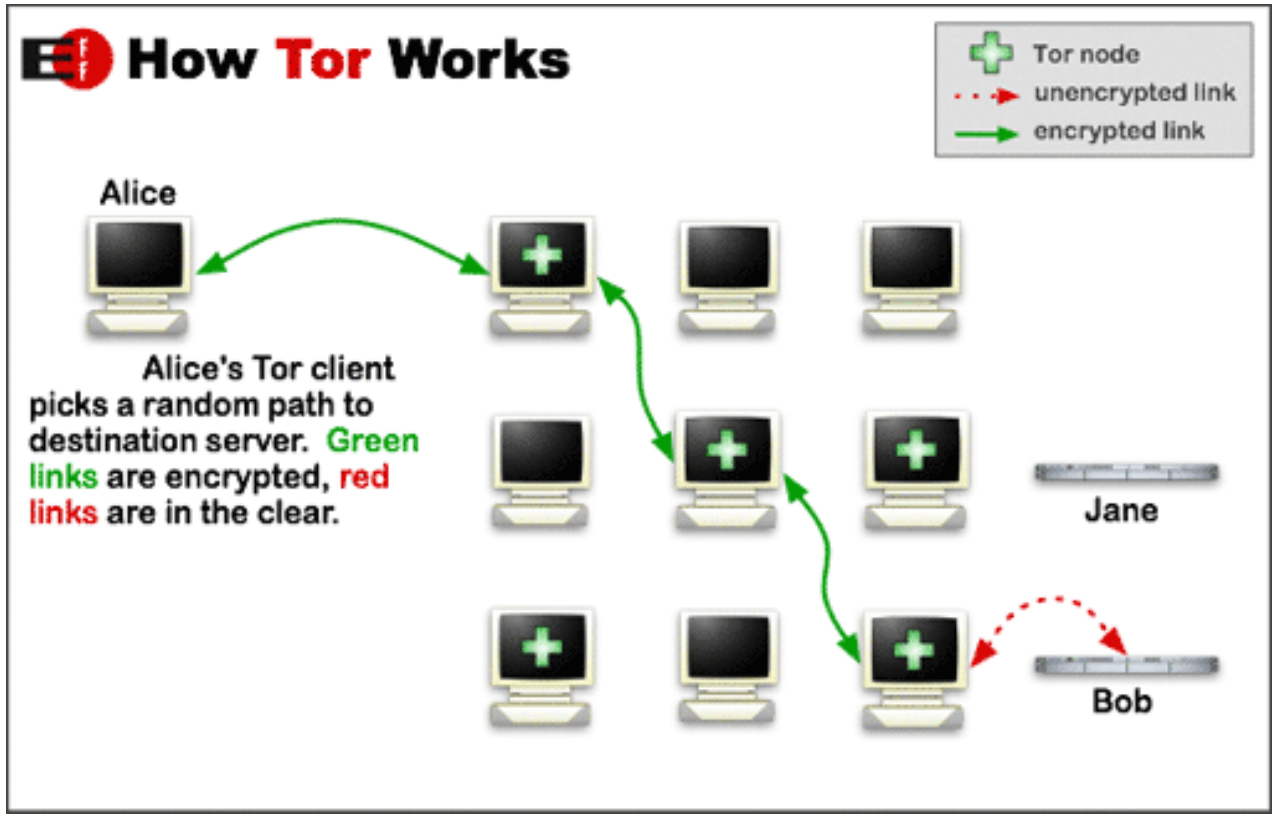
Redes superpuestas

- Red superpuesta (overlay network)
  - Red virtual consistente en nodos reales y enlaces virtuales, construida sobre una red subyacente (típicamente IP)
  - Ofrece funcionalidades adicionales
    - Servicio optimizado a las necesidades de una aplicación
    - Operación más eficiente en un determinado entorno de red
    - Características adicionales, como multicast o seguridad
- Algunos ejemplos:
  - Redes P2P (ahorro de espacio y ancho de banda en servidores)
  - Redes VPN, Tor (anonimato, ubicación)

Redes superpuestas

Red Tor

- Red superpuesta a IP para separar identificación de enrutado
  - 36 millones de usuarios en 2011

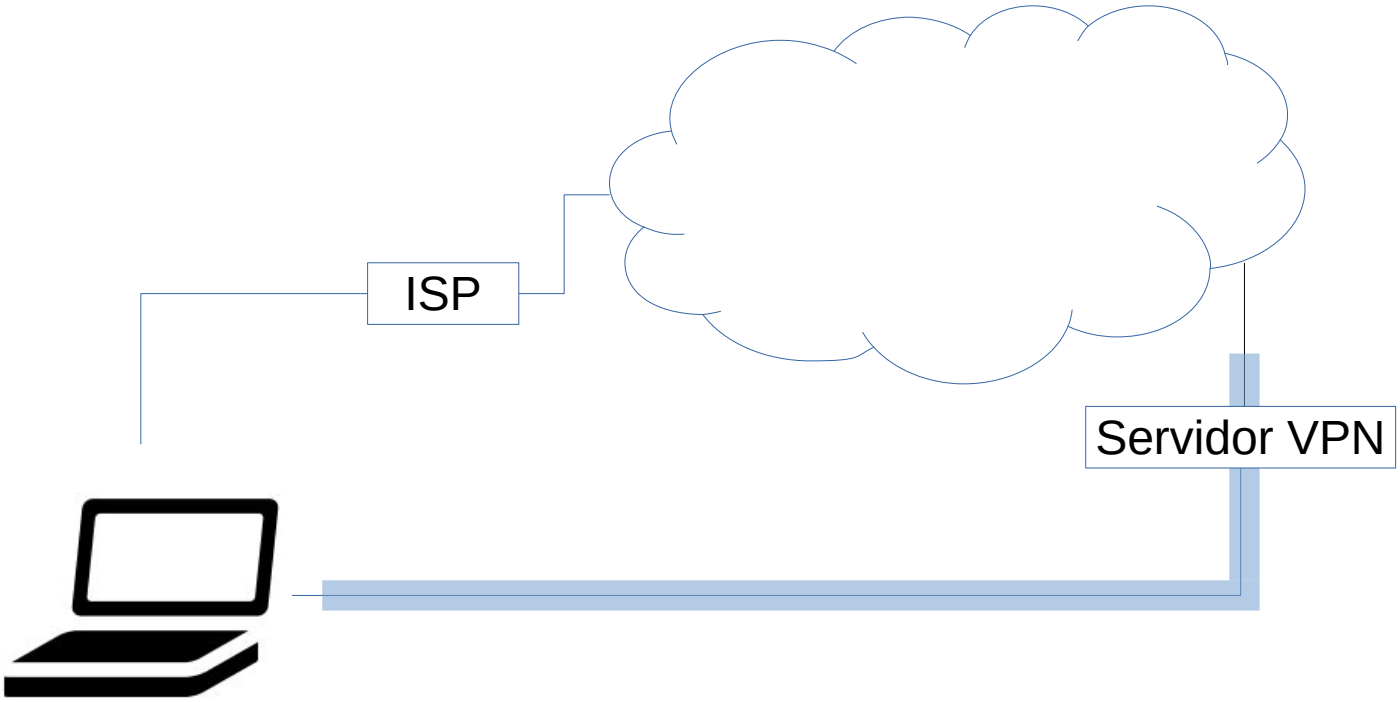


https://en.wikipedia.org/wiki/Tor_%28anonymity_network%29#Operation

Redes superpuestas

Red VPN

- Una versión simplificada de Tor
  - Un tercio de los usuarios de internet la han usado alguna vez



Middleware

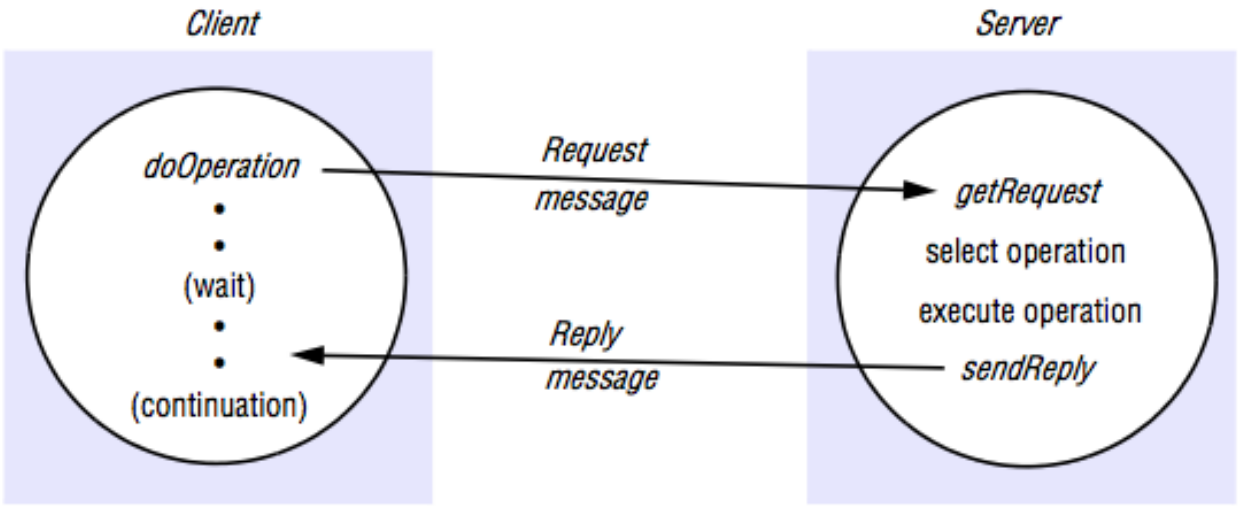
- Introducción
- Middleware de bajo nivel
- Invocación remota
  - Petición-respuesta
  - Remote Procedure Call
  - Remote Method Invocation
- Servicios web

Invocación remota

- 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 RP para la llamada a métodos de objetos en nodos remotos

Protocolo de petición-respuesta

- 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*



Colouris et al. 2011

Protocolo de petición-respuesta

Modelo de fallo

- Por omisión: el servidor no responde o el mensaje se pierde
  - *doOperation* puede implementar un timeout en su espera por el mensaje de respuesta.
    - Repite la petición o determina que el servidor está caído (tras un determinado número de intentos)
- Bizantino: el servidor recibe peticiones duplicadas
  - Puede implementar un mecanismo para descartarlas si todavía está realizando la operación
  - 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 de petición-respuesta

Variaciones

- Variaciones sobre el protocolo petición-respuesta
  - **Request (R)**: cuando no hay valor de retorno y no se requiere confirmación de que la operación se ha realizado
  - **Request-Reply (RR)**: el protocolo básico
  - **Request-reply-acknowledge (RRA)**: permite borrar del histórico las respuestas de las que obtiene acuse de recibo (A).

Protocolo	Mensajes enviados por		
	Cliente	Servidor	Cliente
R	Request		
RR	Request	Reply	
RRA	Request	Reply	Acknowledge reply

Protocolo de petición-respuesta

Caso de estudio: HTTP

- HTTP es un buen ejemplo de protocolo de petición-respuesta
  - Protocolo para hacer peticiones a servidores web
    - Páginas HTML
    - Datos en XML, JSON, etc.
  - Las peticiones especifican una URL que incluye
    - El DNS del host servidor
    - Número de puerto (opcional)
    - Identificador de un recurso
- Fija una serie de tipos de petición: PUT, GET, POST, UPDATE.

Protocolo de petición-respuesta

HTTP: GET y POST

- **GET**: pide el recurso cuya URL se da como argumento.
  - Si es un dato, el servidor lo retorna
  - Si es un programa, el servidor lo ejecuta y devuelve su salida
    - Se pueden añadir argumentos a la URL para el programa

Colouris et al. 2011

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

petición

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

respuesta

- **POST**: especifica la URL de un recurso que pueda tratar con los datos suministrados en el cuerpo del mensaje
  - Postear en una lista de correo/modificar un perfil web
  - Añadir entradas a una base de datos
  - Proveer datos a un programa (p.ej. un formulario)

Remote Procedure Call

- **Objetivo**: hacer la programación distribuida similar, si no igual, a la programación convencional
- **Procedimiento**: extender las llamadas a procedimientos para que funcionen en entornos distribuidos
  - Los procedimientos ubicados en máquinas remotas se pueden llamar como si fueran locales [Birrell & Nelson, 1984]

Remote Procedure Call

Problemas de diseño: interfaces

- Programación mediante interfaces
  - Casi todos los lenguajes modernos son modulares
    - La interfaz de cada módulo especifica la manera de usarlo
      - El resto del módulo permanece ‘oculto’
  - En un SD, los módulos pueden ejecutarse en procesos separados
    - El paso de argumentos por referencia no está permitido
    - Los parámetros de salida van en un mensaje de respuesta
- Lenguajes de definición de interfaz (IDL)
  - En un SD, los métodos a menudo están en diferentes lenguajes
  - Los IDLs permiten llamar procedimientos de otros lenguajes
  - Sun XDR, CORBA IDL o WSDL son algunos ejemplos de IDL

Remote Procedure Call

Problemas de diseño: semántica de la llamada

- Diversos modos de garantizar la entrega de mensajes
  - ***Reintentos***: controla la retransmisión de la petición hasta que el servidor la reciba o la asunción de que ha fallado
  - ***Filtro de duplicados***: controla el uso de retransmisiones y si el servidor descarta peticiones duplicadas
  - ***Retransmisión de resultados***: controla si se debe mantener un histórico de resultados para poder retransmitirlos si se pierden

Fault tolerance measures			*Call 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*

Colouris et al. 2011

Remote Procedure Call

Problemas de diseño: semántica de la llamada

- Semántica "**quizás**":
  - El procedimiento remoto se ejecuta 0 ó 1 veces
    - Fallos por omisión (mensaje de respuesta perdido)
    - Fallos de parada (falla el servidor o su operación remota)
- Semántica "**al menos uno**"
  - El procedimiento remoto se ejecuta 1+ veces
    - Fallos de parada (falla el servidor)
    - Fallos arbitrarios (re-ejecución de la operación remota)
- Semántica "**como mucho uno**"
  - El procedimiento remoto se ejecuta 0 ó 1 veces, pero si se ejecuta 0 veces tenemos conocimiento de ello

Remote Procedure Call

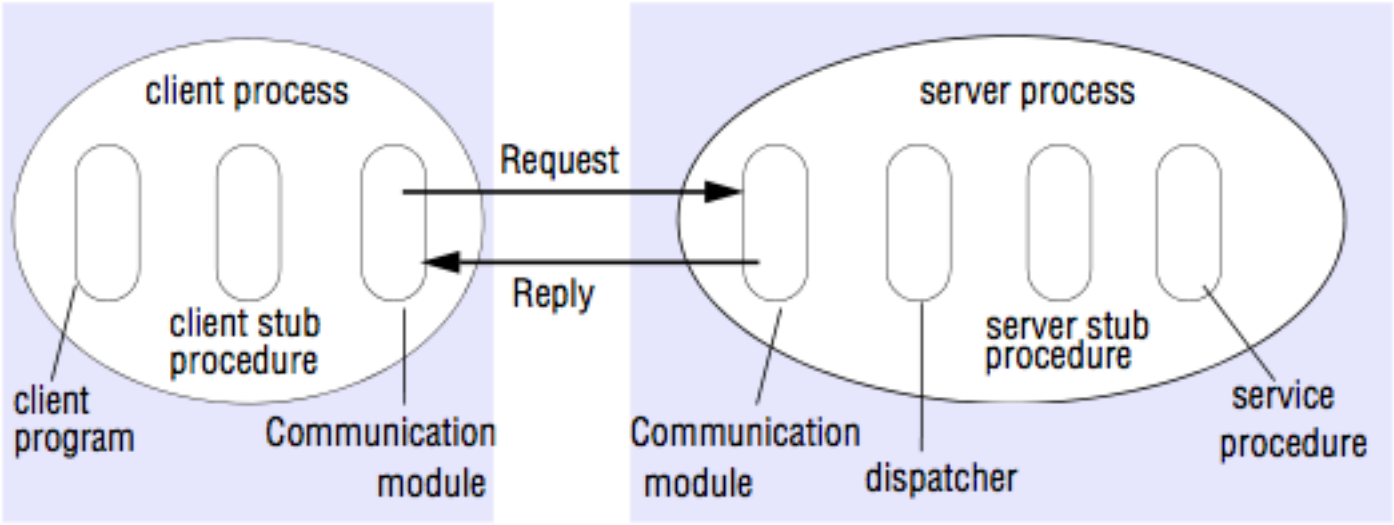
Problemas de diseño: transparencia

- Birrell y Nelson [1984] intentaron hacer las RPC tan parecidas a las llamadas locales como fuera posible
  - Todos los procedimientos de middleware a bajo nivel (marshalling, etc.) se ocultan al programador.
- Sin embargo, las llamadas remotas
  - Tienen más posibilidades de fallar que las locales y no se puede determinar si fue un fallo de red o de proceso
    - Las aplicaciones deben poder recuperarse de estos fallos
  - Tienen una latencia mayor
    - Es conveniente minimizar las llamadas remotas

Remote Procedure Call

Implementación

- El cliente incluye un procedimiento **stub** (cabo) que realiza el marshalling y (en vez de ejecutar el procedimiento) envía la petición y espera por la respuesta
- El servidor tiene otro **stub** por cada procedimiento de servicio. Cuenta también con un único **despachador** que selecciona el stub que hay que invocar dependiendo de la petición recibida
- Los **módulos de comunicación** (uno por proceso) direccionan el mensaje y se encarga de posibles semánticas (retransmisiones, timeouts, duplicados, multicast, etc.)



Coulouris et al. 2011

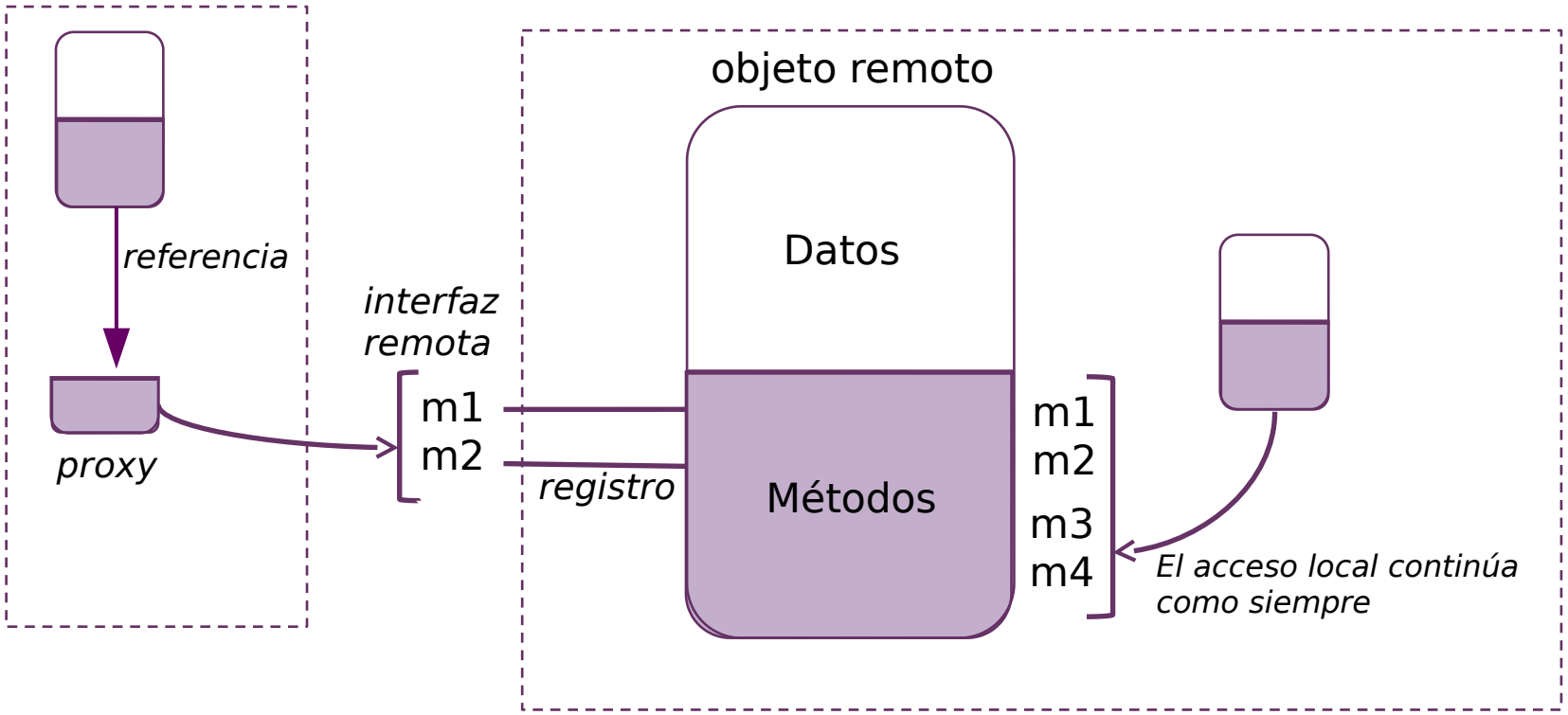
Remote Method Invocation

- Extensión de RPC para **objetos distribuidos**: un objeto puede invocar un método de un objeto remoto.

- Aspectos a tener en cuenta
  - ***Atributos***: en un modelo orientado a objetos, se puede acceder a atributos del objeto directamente.
  - ***Referencias a objetos***: cuando accedemos a un objeto, lo hacemos a través de referencias.
  - ***Interfaces***: definición de un método sin especificar su implementación.
  - ***Excepciones***: si un método falla de manera controlada, puede lanzar una excepción, pasando el control de la ejecución al objeto que lo invocó.

Remote Method Invocation

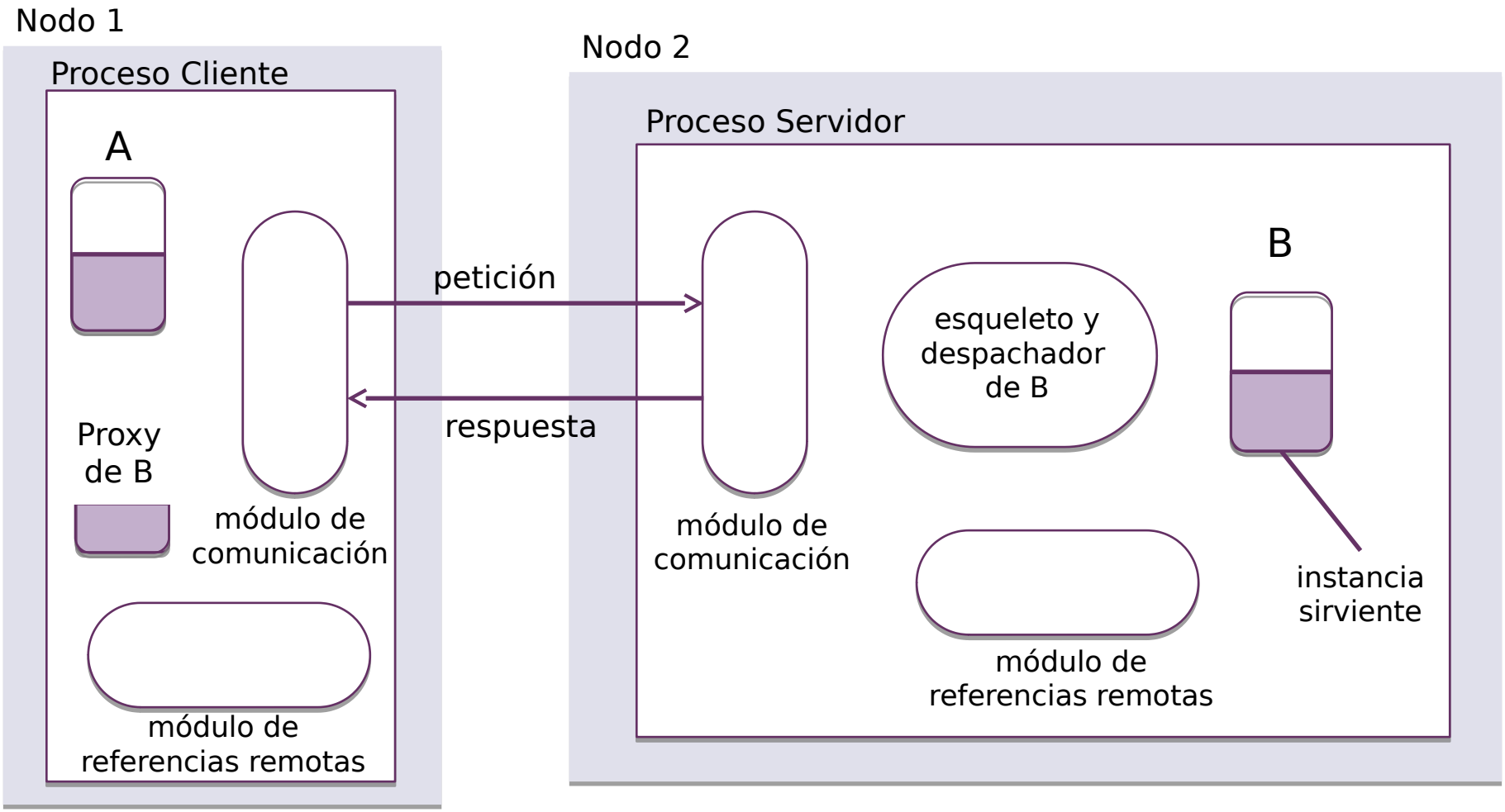
Interfaz remota



- Cada objeto distribuido tendrá una **interfaz** remota indicando los métodos disponibles.
- Esa interfaz se anunciará mediante un registro (*binding*) y se instanciará a modo de proxy en los clientes
- Evita el acceso a **atributos** (referencia a memoria remota) y trata las **referencias** a objetos remotos como *proxies*

Remote Method Invocation

Implementación



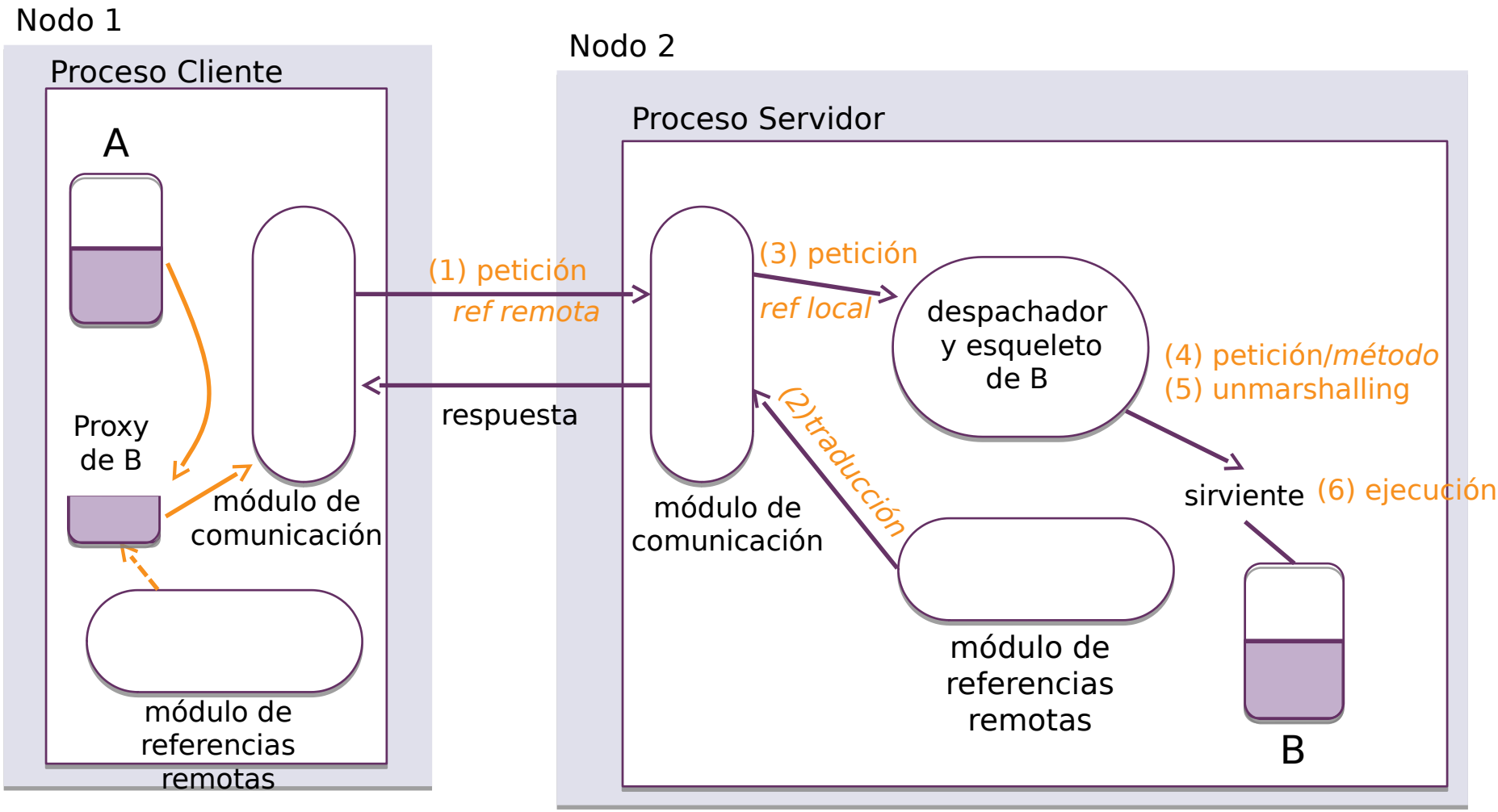
Remote Method Invocation

Implementación

- **Proxy**: realiza la invocación remota transparente al cliente, comportándose como un objeto local, pero en vez de ejecutar el método invocado, lo manda en un mensaje al servidor.
- **Despachador**: recibe mensajes de petición del módulo de comunicación, selecciona el método adecuado y pasa la petición al esqueleto
- **Esqueleto**: implementa los métodos de la interfaz remota, recibiendo una petición del despachador, realizando el unmarshalling e invocando al sirviente
- **Sirviente**: instancia de una clase que provee el cuerpo de un objeto remoto. Es el que realmente ejecuta las peticiones remotas
- **Módulo de referencias remotas**: traduce las referencias locales a remotas (o viceversa)
  - Crea referencias remotas
  - Contiene una entrada por cada objeto remoto y por cada proxy
- **Módulo de comunicación**: lleva a cabo el protocolo de petición-respuesta
  - Cuando llega una petición, el módulo de comunicación del servidor traduce la referencia remota en una referencia local mediante el módulo de referencias remotas
  - Manda la referencia local junto con el mensaje al despachador

Remote Method Invocation

Implementación



Remote Method Invocation

Caso de estudio: Java RMI

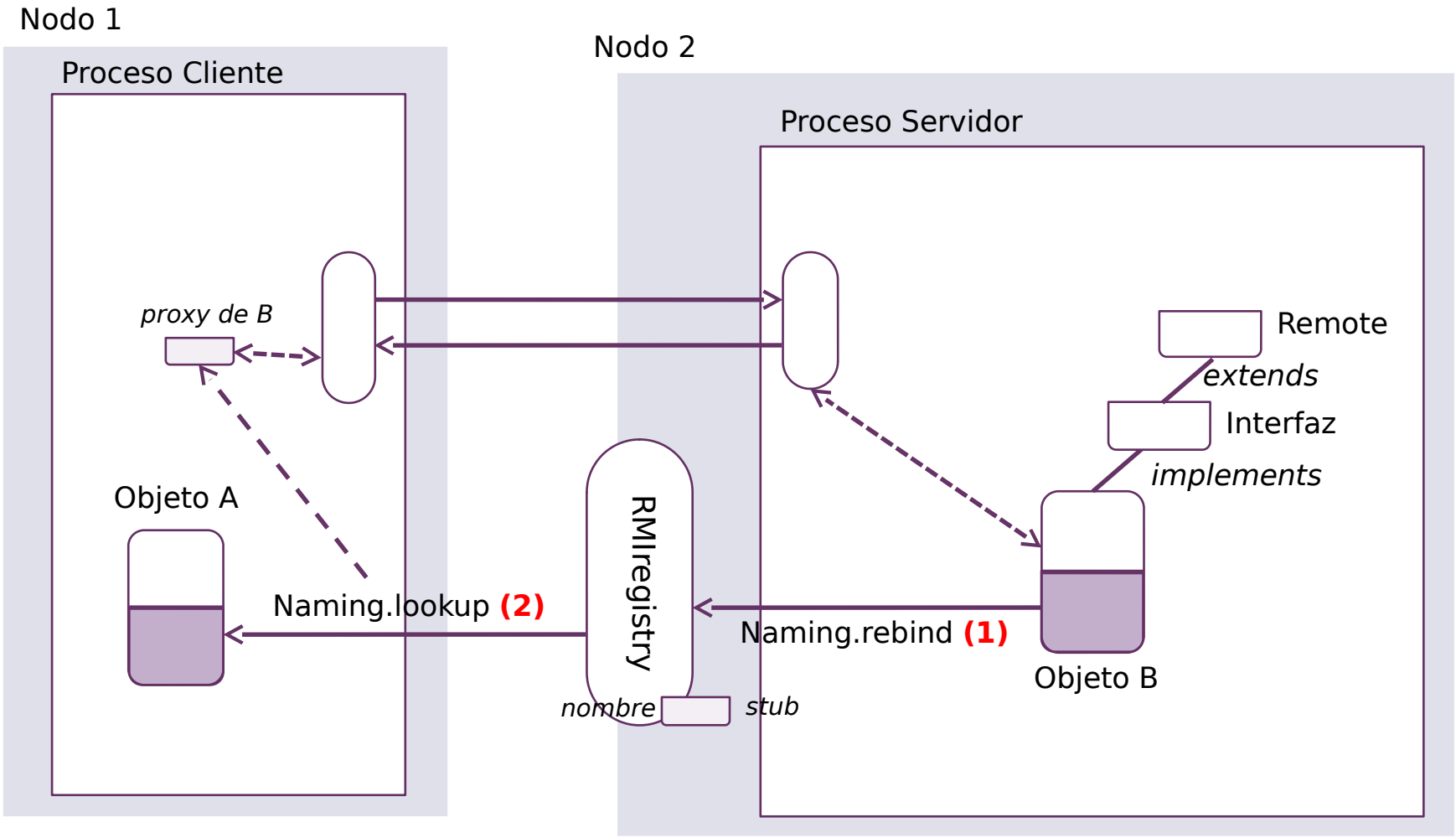
- Afortunadamente, la generación toda esta infraestructura es automática y relativamente trasparente

- Por ejemplo, en Java los objetos remotos deben implementar una interfaz que herede de la clase ***Remote***
  - Los métodos de esta interfaz serán accesibles vía remota
  - Estos métodos deben lanzar la excepción *RemoteException*
  - Utilizan un *registroRMI* para hacerse públicos

```
/**
 * Interfaz para un servicio RMI de corredores de bolsa
 * Debe heredar de java.rmi.Remote
 * Debe manejar RemoteException en sus métodos
 */
public interface CorredorDeBolsa extends Remote
 {
  String listarTitulos() throws RemoteException;
  void comprar(String nombre, int cantidad) throws RemoteException;
  void vender(String nombre, int cantidad) throws RemoteException;
  }
```

Remote Method Invocation

Caso de estudio: Java RMI



Middleware

- Introducción
- Middleware de bajo nivel
- Invocación remota
- Servicios web
  - Introducción
  - SOAP
  - REST
  - XML/JSON-RPC

Introducción

Tipos de middleware

Middleware	Enfoque	Aparición	Comentarios
Petición-respuesta	Paso de mensajes	70s	Inicios, patrón todavía válido
RPC	Uso de métodos	1984	Prog. estructurada Muy utilizado
RMI	Uso de objetos	1995	Muy utilizado
Servicios Web	Métodos como servicios	1998 **SOAP** 2000 **REST**	Simple, muy utilizado
P2P	Arquitecturas horizontales	1999 (Napster)	Horizontal, muy utilizado

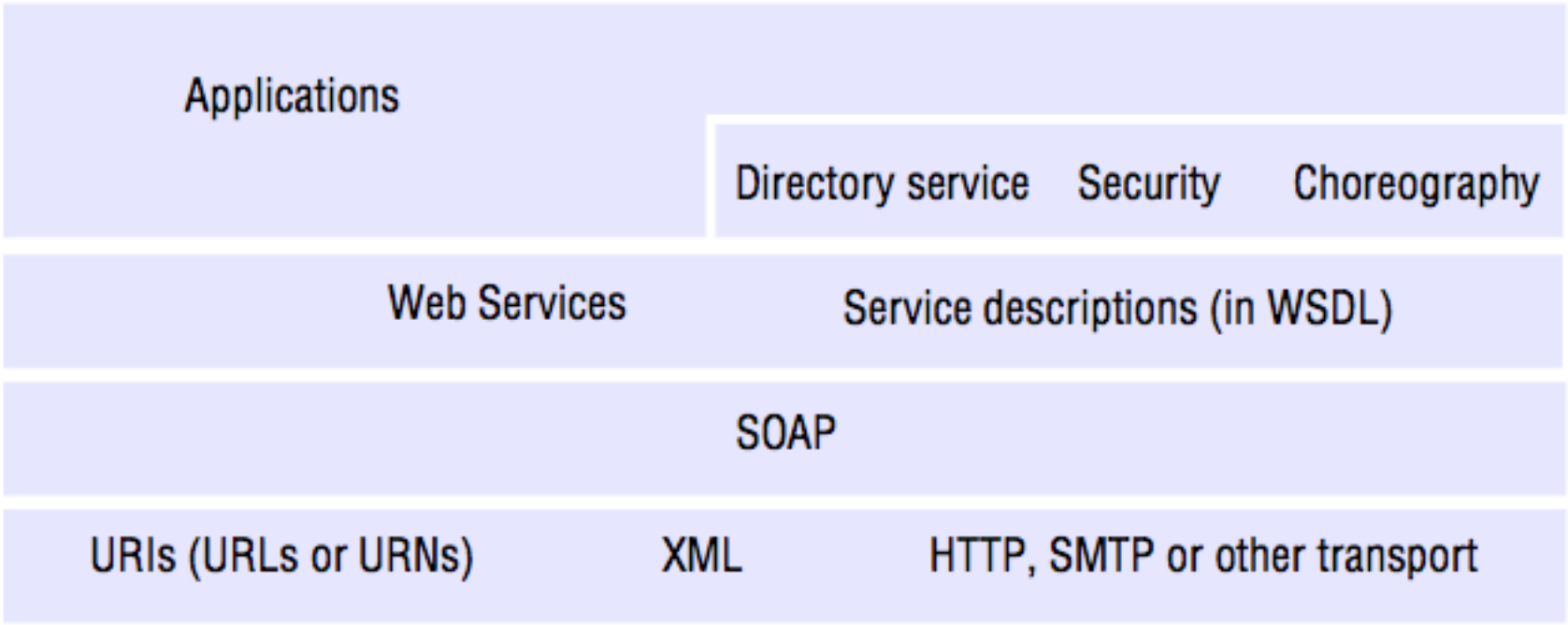
- Los middleware se adaptan a las necesidades y tecnologías del momento
- Algunos tipos de middleware no son necesariamente excluyentes, p. ej.:
  - El protocolo de petición-respuesta se aplica en servicios web o RMI
  - Un middleware P2P puede dar soporte horizontal a un servicio web

SOAP

- **Simple Object Access Protocol** (Dave Winer et al. 1998)
  - Especificación para el intercambio de información estructurada en servicios web, a través de redes de ordenadores
- Se basa en tres componentes principales
  - **WSDL**: lenguaje de descripción del servicio
  - **HTTP/SMTP**: protocolo de comunicación
  - **XML**: lenguaje de especificación de peticiones y respuestas
- **Independiente**: puede usarse sobre servicios escritos en cualquier lenguaje
- **Neutral**: puede usarse sobre cualquier protocolo de transporte

SOAP

Arquitectura



Colouris et al. 2011

- Las aplicaciones pueden ser otros servicios web
- Los servicios llevan asociadas descripciones en un lenguaje (IDL) neutral
- El servicio usa un protocolo para aceptar peticiones y ofrecer resultados en XML, basado en un direccionamiento mediante URIs y protocolos tipo HTTP

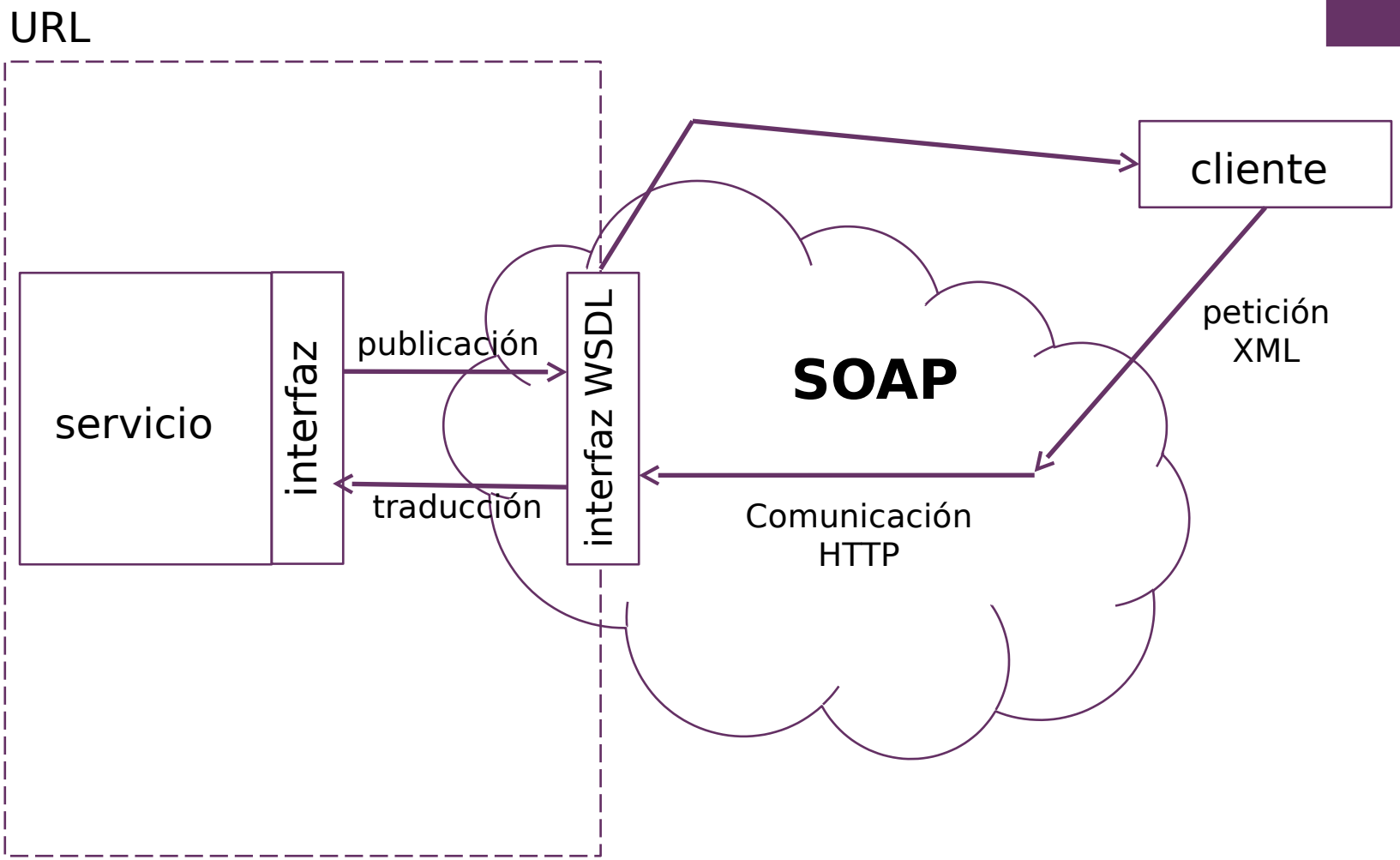
WSDL

Web Service Definition Language

- Lenguaje para especificar la interfaz de un servicio web
- **Neutral** respecto al lenguaje de programación
- WSDL es un tipo de IDL (Interface Definition Language)
  - CORBA IDL
  - Servicios Web (**WSDL**)
  - Facebook (Thrift)

SOAP

Modo de funcionamiento



REST

- REpresentational State Transfer (Roy Fielding, 2000)
- Arquitectura similar a la definida por SOAP
  - Basado en HTTP y URLs
  - Mensajes en XML (generalmente)
- **Utiliza la potencia de la URI** como WSDL
- **Enfocado a los datos** en vez de a las interfaces
  - Un servicio REST es *RESTful* si cumple que:
    - La URI del servicio es http://foo.com/resources
    - Los datos están en un formato estándar (XML, JSON)
    - Sólo hay operaciones HTTP: GET, PUT, DELETE, POST
    - API accesible a través de un navegador (hipertexto)

REST

Recursos y representaciones

- Un **recurso** es una fuente de información específica
  - Referenciada por un identificador global (URI)
- Una **representación** de un recurso es la forma en la que se intercambia dicho recurso
- Recurso:
  - Un círculo
- Representación:
  - Centro y radio en un .svg
  - Tres puntos de su circunferencia en un .csv

REST

Operaciones

	Colección (http://foo.com/resources)	**Elemento** (http://foo.com/resources/item17)
GET	**Lista** las URIs de los elementos de la colección	**Recupera** la representación del elemento
PUT	**Reemplaza** la colección entera por otra	**Reemplaza** el elemento (si no existe, falla)
POST	**Crea** un nuevo elemento en la colección, le asigna una URI automáticamente y devuelve esa URI	Trata el elemento como una colección, y **crea** un nuevo elemento en ella
DELETE	**Elimina** la colección entera	**Elimina** el elemento

REST

Modelo centrado en servicios (SOAP)

- Método: getUserDetails
  - argumento: ID del usuario
- getUserDetails(1235):

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-
encoding">
 <soap:body pb="http://www.acme.com/phonebook">
  <pb:GetUserDetails>
   <pb:UserID>12345</pb:UserID>
  </pb:GetUserDetails>
 </soap:Body>
</soap:Envelope>
```

Modelo centrado en datos (RESTful)

- Dato: detalles del usuario 1235

http://www.acme.com/phonebook/**UserDetails/12345**

- Dato: detalles del usuario de nombre John y apellido Doe

http://www.acme.com/phonebook/UserDetails**?firstName=John&lastName=Doe**

- Dato: detalles del usuario de nombre John y apellido Doe en formato JSON

http://www.acme.com/phonebook/UserDetails.**json**?firstName=John&lastName=Doe

REST

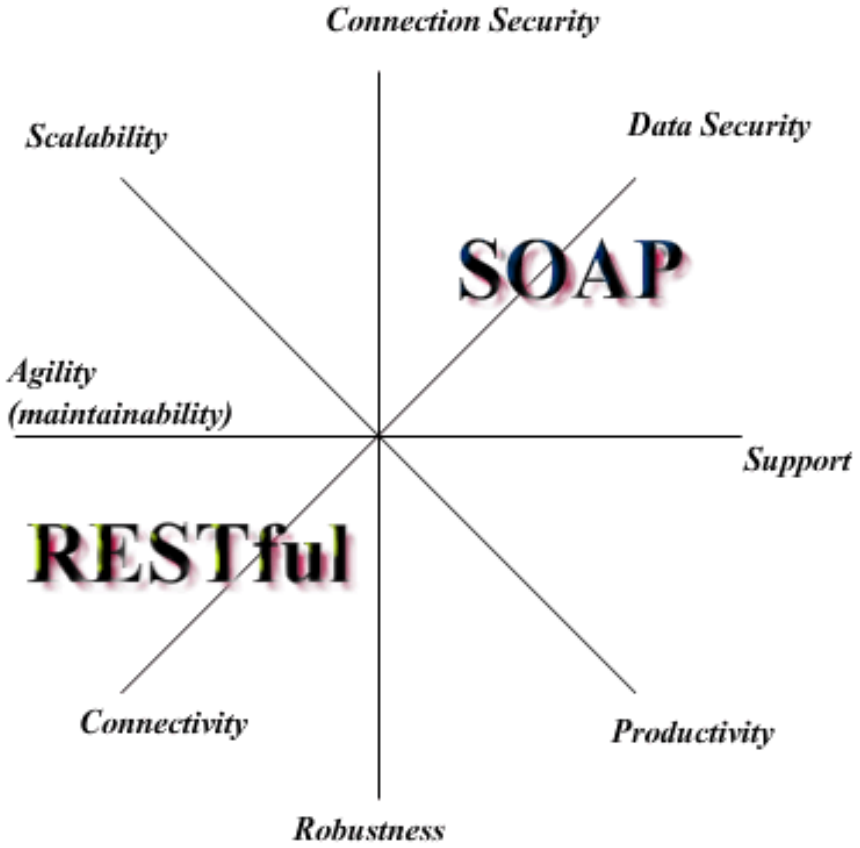
REST vs SOAP

	REST	SOAP
Alcance	Arquitectura	Protocolo
Estándar?	No	Sí
Mensajes	HTML o XML	XML
Tipado	No (HTML) o fuerte (XML)	Fuerte (XML)
Ancho de banda	Menor en HTML	Tags adicionales de XML
Testeo	Navegador o herr. tipo Postman	Herramientas tipo soapUI
Bibliotecas	No (URIs)	Sí
Petición usual	GET	POST
Cliente	Sencillo	Algo más complejo

http://geeknizer.com/rest-vs-soap-using-http-choosing-the-right-webservice-protocol/

REST

REST vs SOAP



http://nicolas-zozol.developpez.com

REST

REST no es perfecto

- REST no siempre es mejor decisión de diseño que SOAP
  - Las URIs tienen un tamaño máximo
    - No soportan argumentos muy largos
    - Para adjuntar datos (texto o binarios) necesitamos usar POST
      - SOAP es mejor al estar estandarizado
  - Las URIs son públicas
    - Problema si los argumentos son delicados
  - Tipado fuerte
    - REST lo soporta (vía XML) pero no tiene un estándar de tipos
- Generalmente, REST es una solución más sencilla y limpia
  - Suficiente para servicios web poco complejos

REST

Ejemplos*

- Twitter
  - REST es la API original de Twitter, y todavía hoy la más popular entre desarrolladores
  - https://developer.twitter.com/en/docs/api-reference-index
- Google
  - Tuvo una REST API hasta 2010
    - https://developers.google.com/web-search/docs
  - Ahora utiliza JSON/Atom Custom Search API
    - http://code.google.com/apis/customsearch/v1/overview.html

* Una buena cantidad de APIs, la mayoría REST, se pueden consultar http://programmableweb.com

REST

Filosofía

- La filosofía subyacente a REST es que una orden (verbo) sobre un recurso web (objeto) debería devolver una versión u otra dependiendo del usuario (sujeto)
- Por ejemplo, en el caso de GET
  - Si el usuario es una persona, devuelve un fichero legible
    - Típicamente un documento HTML como hasta ahora
  - Si el usuario es una máquina, devuelve un fichero en un lenguaje
    - Por ejemplo XML, JSON, etc.
- La web actual, dado un concepto (información), usa una sola representación (página web). Con varias representaciones, la web se está convirtiendo en una herramienta más completa.

XML-RPC

- Creado en 1998 por Dave Winer
- RPC que usa
  - XML como sistema de codificación de las peticiones
  - HTTP como mecanismo de envío
- Evolucionó hasta convertirse en SOAP
  - Más sencillo, no requiere un WSDL
- Como estándar, no es muy evolucionado
  - Simplemente define tipos de datos básicos y peticiones en XML

XML-RPC

Datos, invocación y respuesta

marshalling

```
<int>42</int>
<double>-12.53</double>
<boolean>1</boolean>
<string>Hello world!</string>
<nil/>


<base64>eW91IGNhbid0IHJlYWyE=</base64>


<array>
  <data>
    <value><i4>1404</i4></value>
    <value><string>Cat</string></value>
    <value><i4>1</i4></value>
  </data>
</array>
```

invocación remota

```
<?xml version="1.0"?>
<methodCall>

<methodName>examples.getStateName</methodName>
  <params>
    <param>
        <value><int>40</int></value>
    </param>
  </params>
</methodCall>
```

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
        <value><string>Dakota</string></value>
    </param>
  </params>
</methodResponse>
```

JSON-RPC

- Similar a XML-RPC, pero datos y mensajes se especifican en JSON, mucho menos verboso
  - JSON para definición de datos y mensajes
  - HTTP o sockets TCP/IP para envío
- Ejemplos

```
--> {"jsonrpc": "2.0", "method": "subtract", "params": [42, 23], "id": 1}
<-- {"jsonrpc": "2.0", "result": 19, "id": 1}
```

```
--> {"jsonrpc": "2.0", "method": "foobar", "id": 10}
<-- {"jsonrpc": "2.0", "error": {"code": -32601, "message": "Procedure not found."}, "id": 10}
```

JSON-RPC

- De alguna manera, JSON-RPC cierra el círculo:

Método	Año	WSDL	Recursos	Datos
XML-RPC	1998	No	No	XML
SOAP	1998	Sí	No	XML/WSDL
REST	2000	No	Sí	Cualquiera
JSON-RPC	2005	No	No	JSON

Resumen

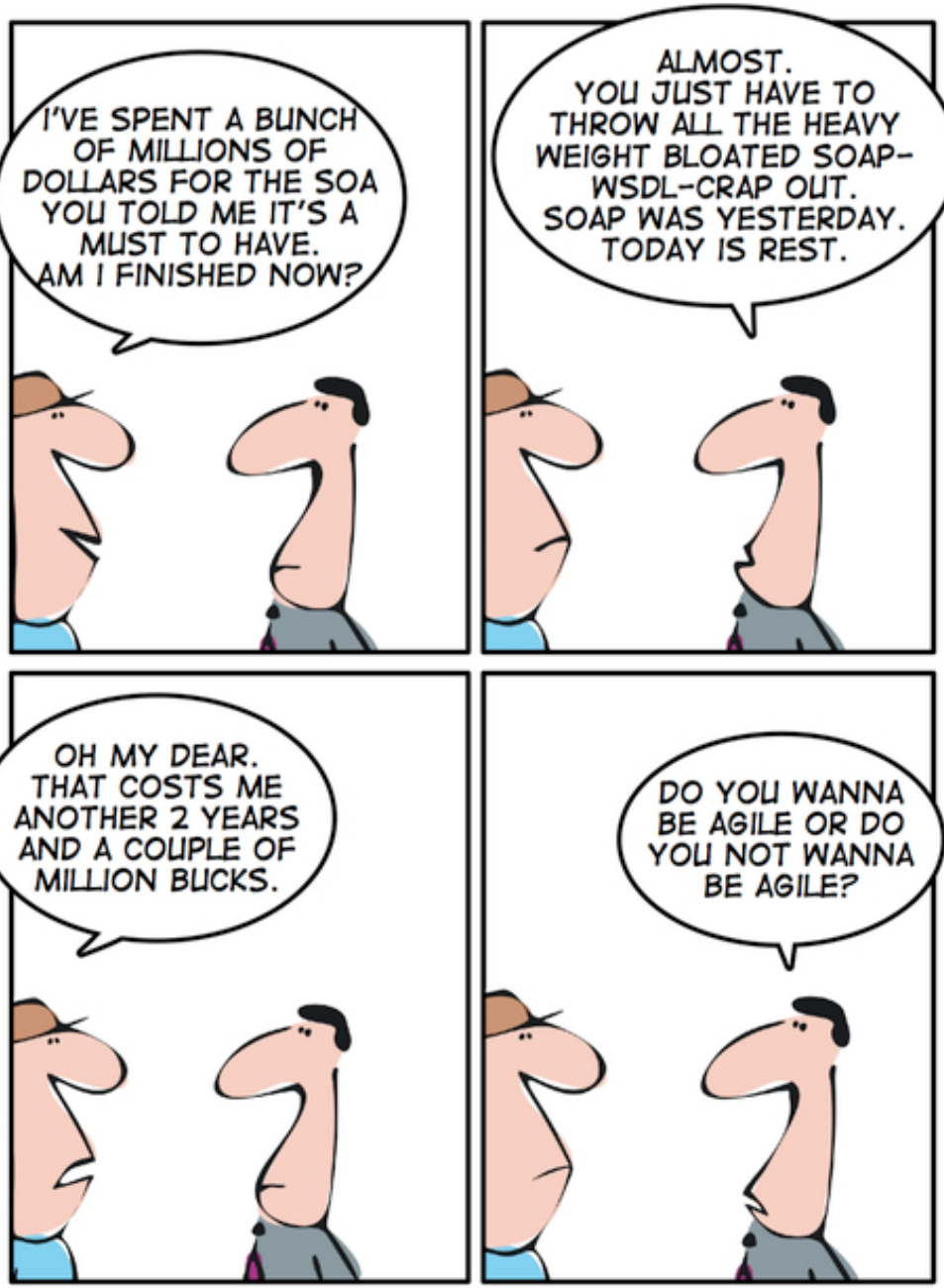
- El **middleware** es una capa software que abstrae de los problemas de un sistema distribuido
- El middleware de **bajo nivel** se encarga de resolver problemas de **heterogeneidad**
  - El **marshalling** trata los datos comunicados para crear representaciones **homogéneas**
  - Las **redes superpuestas** mejoran problemas de **direccionamiento**, **movilidad**, **seguridad** y **escala**, apoyándose en las redes de comunicación existentes
- El middleware de **alto nivel** resuelve problemas de **comunicación,** mediante esquemas de **petición-respuesta (RR)**
- El esquema RR debe tratar con fallos arbitrarios o de omisión, utilizando distintas **estrategias** (**R, RR, RRA**) que pueden necesitar de componentes adicionales como timeouts o históricos de peticiones. Estas estrategias darán lugar a distintas **semánticas** de comunicación (**0+, 1+, 0-1**)
- La implementación de estos esquemas en programación estructurada es **RPC**, que trata llamadas a métodos remotos como locales gracias a una arquitectura de **stubs** y **módulos de comunicación**
- La implementación en orientación a objetos es **RMI**, que permite invocar métodos de objetos remotos gracias a una combinación de **publicación de interfaces** y objetos **proxy**

Resumen (s. web)

- Los servicios web son un tipo de middleware que busca la **simplicidad** en base a las características de **Internet**
- Se centran en un paso de mensajes mediante **HTTP** y uso de datos en texto plano con estructura (**XML, JSON**)
- **SOAP** se centra en el uso de un XML propio para cada servicio, escrito bajo un estándar **WSDL**
- **REST** se centra en el uso de la propia estructura de **URIs** en HTTP para las peticiones, y en un formato abierto a definir en la respuesta (típicamente JSON)
- Un servicio es **RESTful** si cumple con un intercambio basado en **recursos**, entendidos como objetos de datos, que se modifican sólo mediante las cuatro acciones de HTTP (**GET, POST, PUT, DELETE**)
- Todo servicio web necesita en el fondo definir sus **mensajes** y **datos** compartidos, es responsabilidad del desarrollador elegir aquél que mejor se adapte a sus sistema

Referencias

- G. Colouris, J. Dollimore, T. Kindberg and G. Blair. *Distributed Systems: Concepts and Design (5th Ed)*. Addison-Wesley, 2011
  - Capítulo 4 para middleware de bajo nivel
  - Capítulo 5 para invocación remota
  - Capítulo 9 para servicios web
- Seminario sobre Java RMI
- Seminarios sobre REST y REST avanzado
- Servicios web:
  - Learn REST: A tutorial - http://rest.elkstein.org
  - How I explained REST to my wife
    - http://www.looah.com/source/view/2284
  - REST vs SOAP: the right web service
    - http://geeknizer.com/rest-vs-soap-using-http-choosing-the-right-webservice



THE CONSULTANTS HANDBOOK PART 2:
MAKE SURE YOUR COSTUMER IS ALWAYS AGILE

http://geekandpoke.typepad.com/