Formatos de archivos de datos para intercambio

Se entiende por formato a "la estructura interna y codificación de un objeto digital, que permite que éste sea procesado o presentado en una forma accesible".

Existen diferentes formatos según el tipo de datos que se trate. Así, por ejemplo, hay formatos comunes para texto, imagen o sonido, y otros específicos dentro de ciertas disciplinas y áreas.

Los formatos de archivo determinan cómo se pueden usar los datos. Es importante decidir qué formatos de archivo utilizar para la recopilación de datos, el procesamiento de datos, el almacenamiento de datos y la conservación a largo plazo.

Los formatos se pueden clasificar en dos tipos: abiertos y cerrados.

Como señala el Open Data Handbook[1]:

"Un formato abierto es aquel donde las especificaciones del software están disponibles para cualquier persona, de forma gratuita, así cualquiera puede usar dichas especificaciones en su propio software sin ninguna limitación en su reutilización que fuere impuesta por derechos de propiedad intelectual."

Si el formato del archivo es 'cerrado', esto puede ser debido a que el formato es propietario y sus especificaciones no están disponibles públicamente, o porque el formato es propietario y aunque las especificaciones se han hecho públicas, su reutilización es limitada.

Los formatos siempre estarán asociados a software y hardware específicos, lo que significa que su reutilización en el futuro dependerá de la disponibilidad de estos en el mediano y largo plazo.

Teniendo presente la necesidad de interoperabilidad (intercambiar y/o relacionar datos entre aplicaciones/plataformas/sistemas heterogéneas), es habitual la elección de formatos de archivos abiertos y, particularmente, estándares.

Dos factores adicionales a lo anterior son, por un lado la amigabilidad o accesibilidad que ofrecen para su interpretación por parte de los usuarios; y por el otro, las capacidades de conversión de los datos a otros formatos, como una una estrategia que permitirá su preservación en el largo plazo, una vez que un formato dado se vuelvan obsoleto.

Dentro de los formatos de archivos[2] de datos, que satisfacen los principios FAIR[3], se encuentran:

- Contenedores: TAR, GZIP, ZIP
- Bases de datos: XML, CSV, JSON
- Geoespacial: SHP, DBF, GeoTIFF, NetCDF
- Video: MPEG, AVI, MXF, MKV
- Sonido: WAVE, AIFF, MP3, MXF FLAC
- Estadísticas: DTA, POR, SAS, SAV
- Imágenes: TIFF, JPEG 2000, PDF, DNG, GIF, BMP, SVG

1 https://opendatahandbook.org/
2 https://howtofair.dk/how-to-fair/file-formats/
3 https://howtofair.dk/what-is-fair/

- Datos tabulares: CSV, TXT
- Texto: XML, PDF / A, HTML, JSON, TXT, RTF
- Archivo web: WARC

RPC

En computación distribuida, una **llamada a procedimiento remoto** (**RPC**) es un mecanismo usado por un programa de computadora que hace que un procedimiento (subrutina) se ejecute en un espacio de direcciones diferente (comúnmente en otra computadora en una red compartida), que se escribe como si fuera una llamada de procedimiento normal (local), sin que el programador escriba explícitamente los detalles de la interacción remota.

Es consecuencia, el programador escribe esencialmente el mismo código ya sea que la subrutina sea local para el programa en ejecución o remota.

Esta es una forma de interacción cliente-servidor (la persona que llama es el cliente, el ejecutor es el servidor), normalmente implementada a través de un sistema de paso de mensajes de solicitud-respuesta.

En el paradigma de la programación orientada a objetos, las RPC se representan mediante la invocación de métodos remotos (RMI).

El modelo RPC implica un nivel de transparencia de ubicación, es decir, que los procedimientos de llamada son en gran medida los mismos, ya sean locales o remotos, pero por lo general no son idénticos. Esto es, las llamadas locales se pueden distinguir de las llamadas remotas. Las llamadas remotas suelen ser mucho más lentas y menos fiables que las llamadas locales, por lo que es importante distinguirlas.

Las RPC son una forma de comunicación entre procesos (IPC), en la que diferentes procesos tienen diferentes espacios de direcciones: si están en la misma máquina host, tienen distintos espacios de direcciones virtuales, aunque el espacio de direcciones físico sea el mismo; mientras que si están en diferentes hosts, el espacio de direcciones físicas es diferente.

Se han utilizado muchas tecnologías diferentes (a menudo incompatibles) para implementar el concepto.

Las RPC pueden considerarse como un caso especial del modelo general de paso de mensajes.

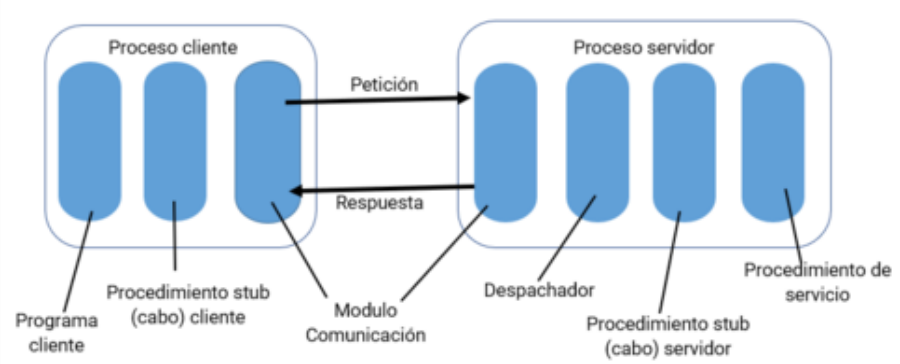
En un programa distribuido, con arquitectura cliente-servidor, el servidor proporciona a los clientes un listado de los módulos que están disponibles para su uso, así como los argumentos necesarios para realizar una correcta llamada a los mismos.

De esta forma el cliente únicamente debe preocuparse de la interfaz, no necesita saber detalles de la implementación del módulo, como por ejemplo el lenguaje de programación, o la plataforma en la que el servicio fue implementado, de esta manera se soluciona el problema de la heterogeneidad de los sistemas distribuidos actuales.

Por otro lado, si un módulo que se ejecuta en un proceso necesita acceder a variables de otro módulo que se ejecuta en otro proceso, no puede acceder a ellas, ya que el paso de argumentos por referencia no está permitido. Como consecuencia de esto último, los parámetros de salida van

en un mensaje de respuesta. Una excepción es CORBA que puede acceder a las variables de un módulo situado en otro proceso mediante los métodos getter y setter.

La implementación de una llamada a procedimiento remoto requiere una serie de componentes de software, los cuales se pueden ver en la siguiente figura traducida de la que se encuentra en el libro "Distributed Systems: Concepts and Design"[4].



Dentro del proceso cliente se encuentra el programa principal (programa cliente). Este se comunica con un procedimiento stub, encargado de realizar la función de marshaling, para transformar los datos al formato adecuado y enviarlos como un mensaje de solicitud a través del módulo de comunicación. Una vez que la respuesta llega, el stub deshace la transformación para obtener los resultados.

El módulo de comunicación del proceso cliente se comunica con el del proceso servidor, desde el cual transmite los datos a un distribuidor (dispatcher) que hace llegar los datos al procedimiento stub correspondiente, que realizará la función de marshaling con el procedimiento servicio. Al existir varios stub, uno por cada procedimiento de servicio, el distribuidor se encarga de elegir el que corresponda en cada petición.

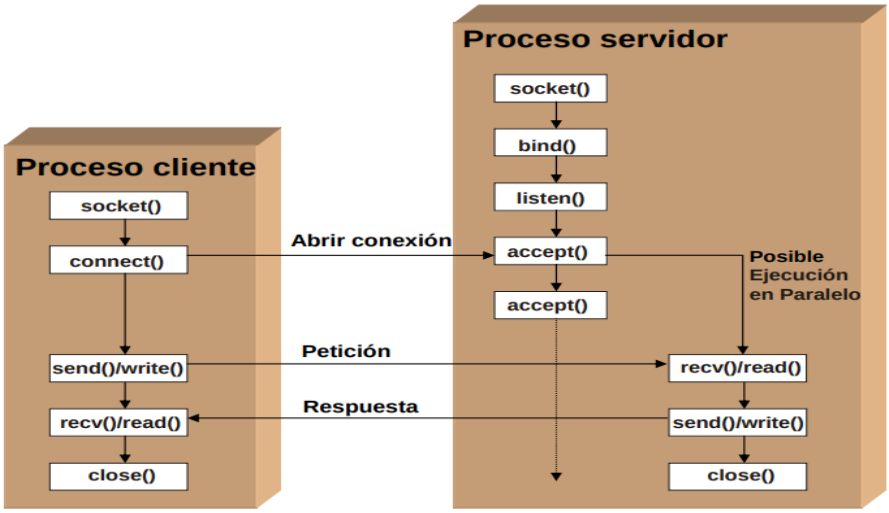
Los módulos de comunicación no sólo se encargan de direccionar los mensajes sino también de otros aspectos relativos a la comunicación como mensajes duplicados, timeouts o retransmisiones.

Hay varios tipos de protocolos, muchos de ellos estandarizados como pueden ser el RPC de Sun denominado ONC RPC(RFC 1057), el RPC deOpen Software Foundation(OSF) denominado DCE/RPCy el "Modelo de Objetos de Componentes Distribuidos de Microsoft" (Distributed Component Object Model, DCOM), aunque ninguno de estos es compatible entre sí. La mayoría de ellos utilizan un lenguaje de descripción de interfaz (interface description language o IDL) que define los métodos exportados por el servidor.

Generalmente, las llamadas a procedimiento remoto (RPC) están implementadas mediante protocolos de petición-respuesta.

Un esquema habitual de la secuencia funcional seguido por las aplicaciones servidor y cliente, usando streams sockets, responde a:

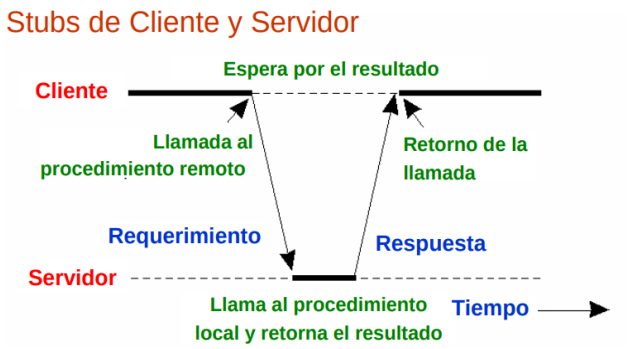
4 G. Colouris, J. Dollimore, T. Kindberg and G. Blair (2011). «Chapter 5.3 - Remote procedure call». Distributed Systems: Concepts and Design. Addison-Wesley. p. 195-201.

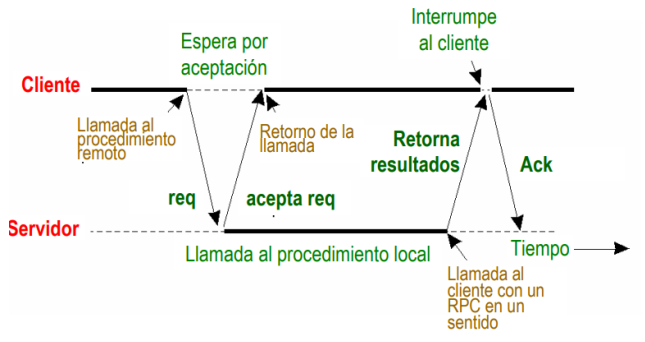


Donde se observan algunas de las primitivas clásicas:

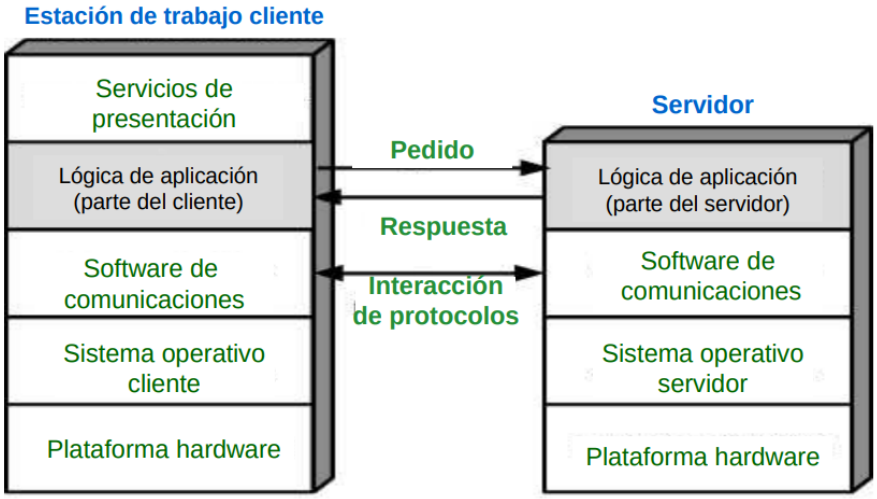
Primitiva	Significado
Socket	Crea un nuevo punto final de comunicación
Bind	Adjunta una dirección local a un socket
Listen	Anuncia el deseo de aceptar conexiones
Accept	Se bloquea el llamador hasta que llegue un requerimiento de conexión
Connect	Intenta activamente establecer una conexión
Send	Envía datos sobre la conexión
Receive	Recibe algunos datos sobre la conexión
Close	Libera la conexión

En las imagenes siguientes, se muestran esquemas de comportamiento bajo perspectiva temporal de los stubs en modo síncrono y asíncrono, respectivamente:





Es conveniente recordar que la interacción entre las aplicaciones servidor y cliente, requieren de un conjunto de capas de software, como puede verse en la siguiente figura:



Actualmente, se está utilizando XML o JSON, como lenguajes para definir el IDL y HTTP como protocolo de aplicación, dando lugar a servicios web como XML-RPC, su derivado SOAP o JSON-RPC.

XML-RPC

Es un protocolo muy simple ya que define sólo unos pocos tipos de datos y comandos útiles, además de una descripción completa de corta extensión.

La simplicidad del XML-RPC contrasta con la mayoría de protocolos RPC que tiene una documentación extensa y requiere considerable soporte de software para su uso.

Según la especificación de*XML-RPC*[5], los principales tipos de datos son:

Nombre	Tags de ejemplo	Descripción

array	`<array>`
	`<data>`
	`<value><i4>1404</i4></value>`
	`<value><string>Algo aquí</string></value>`
	`<value><i4>1</i4></value>`
	`</data>`
	`</array>`
		Array de valores, sin almacenar claves

base64	`<base64>eW91IGNhbid0IHJlYWQgdGhpcyE=</base64>`	Datos binarios codificados en base 64
boolean	`<boolean>1</boolean>`	Valor lógico (0 o 1)
date/time	`<dateTime.iso8601>19980717T14:08:55</dateTime.iso8601>`	Día y hora

5 http://www.xmlrpc.com/spec

double	<double>-12.53</double>	Número de punto flotante de doble precisión
integer	<i4>42</i4> o <int>42</int>	Número entero
string	<string>Hola mundo</string>	String (cadena) de caracteres. Debe seguir la codificaciónXML.
struct	<struct> <member> <name>foo</name> <value><i4>1</i4></value> </member> <member> <name>bar</name> <value><i4>2</i4></value> </member> </struct>	Array de valores, almacenando claves
nil	<nil/>	Valor nulo; una extensión XML-RPC

Ejemplos

- Una invocación **XML-RPC** podría ser:

```
<?xml version="1.0"?>
<methodCall>
  <methodName>org.wikipedia.intercambioDatos</methodName>
  <params>
    <param>
      <value><i4>360</i4></value>
    </param>
    <param>
      <value><i4>221</i4></value>
    </param>
  </params>
</methodCall>
```

- Una respuesta a la invocación:

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value><string>Intercambio datos nro. 360 por 221</string></value>
    </param>
  </params>
</methodResponse>
```

JSON-RPC

Funciona enviando una solicitud a un servidor que implementa este protocolo.

En ese caso, el cliente suele ser un software que intenta llamar a un único método de un sistema remoto. Se pueden pasar múltiples parámetros de entrada al método remoto como una matriz u objeto, mientras que el método en sí también puede devolver múltiples datos de salida (esto depende de la versión implementada).

Todos los tipos de transferencia son objetos únicos, serializados mediante JSON[6].

Una solicitud es una llamada a un método específico proporcionado por un sistema remoto. Puede contener tres miembros:

- method - Un string con el nombre del método a invocar. Los nombres de métodos que comienzan con "rpc" están reservados para métodos internos de rpc.
- params - Un objeto o matriz de valores que se pasarán como parámetros al método definido. Este miembro puede ser omitido.
- id - Una cadena o número no fraccionario que se utiliza para hacer coincidir la respuesta con la solicitud a la que responde. Este miembro puede omitirse si no se debe devolver ninguna respuesta.

El receptor de la solicitud debe responder con una respuesta válida a todas las solicitudes recibidas.

Una respuesta puede contener los miembros mencionados a continuación.

- result - Los datos devueltos por el método invocado. Si se produjo un error al invocar el método, este miembro no debe existir.
- error - Un objeto de error si hubo un error al invocar el método; de lo contrario, este miembro no debe existir. El objeto debe contener *código* de miembros (entero) y *mensaje* (cadena). *Un miembro de datos* opcional puede contener más datos específicos del servidor. Hay códigos de error predefinidos que siguen a los definidos para XML-RPC.
- id - El id de la solicitud a la que responde.

Dado que hay situaciones en las que no se necesita ni se desea una respuesta, se introdujeron las notificaciones. Una notificación es similar a una solicitud excepto por la identificación, que no es necesaria porque no se devolverá ninguna respuesta. En este caso, el miembro id debe omitirse (Versión 2.0) o ser `null` (Versión 1.0).

Ejemplos

Versión 2.0[7]

- Solicitud:

```
{ "jsonrpc" : "2.0" , "method" : "restar" , "params" : { "minuendo" : 42 , "sustraendo" : 23 }, "id" : 3 }
```

- Respuesta:

```
{ "jsonrpc " : "2.0" , "result" : 19 , "id" : 3 }
```

- Notificación (sin respuesta):

6 https://web.archive.org/web/20080517011921/http://json-rpc.org/wiki/specification
7 https://www.jsonrpc.org/specification

```
{ "jsonrpc" : "2.0" , "method" : "actualizar" , "params" : [ 1 , 2 , 3 , 4 , 5 ]}
```

Versión 1.0

- Solicitud:

```
{ "method" : "eco" , "params" : [ "Hola JSON-RPC" ], "id" : 1 }
```

- Respuesta:

```
{ "result" : "Hola JSON-RPC" , "error" : nulo , "id" : 1 }
```

Algunas implementaciones en C++ de XML-RPC y JSON-RPC

https://xmlrpcpp.sourceforge.net/

https://xmlrpc-c.sourceforge.io/

https://github.com/ifm/xmlrpc-c No es mirror del anterior, sino un fork para usar cmake

https://github.com/Tencent/rapidjson

https://github.com/uskr/jsonrpc-lean

https://github.com/cinemast/libjson-rpc-cpp

https://jsonrpc-cpp.sourceforge.net/

https://codeberg.org/jfinkhaeuser/json-rpc

https://www.genivia.com/doc/xml-rpc-json/html/index.html