

PROGRAMACIÓN DE SERVICIOS Y PROCESOS
TÉCNICO EN DESARROLLO DE APLICACIONES
MULTIPLATAFORMA

Modelos de comunicaciones

ÍNDICE

/ 1. Introducción y contextualización práctica	3
/ 2. Arquitectura cliente/servidor	4
/ 3. Características básicas del modelo cliente/servidor	5
/ 4. Caso práctico 1: “Servidores y clientes suficientes para todos”	5
/ 5. Ventajas e inconvenientes del modelo cliente/servidor	6
/ 6. Modelos cliente/servidor	6
/ 7. Comunicación en el modelo cliente/servidor	7
/ 8. Caso práctico 2: “¿Cómo implementar una aplicación cliente/servidor?”	8
/ 9. Diagramas de estados	8
/ 10. Resumen y resolución del caso práctico de la unidad	9
/ 11. Bibliografía	10

OBJETIVOS



Conocer la arquitectura cliente/servidor.

Comprender el rol del servidor.

Conocer el rol del cliente.

Profundizar sobre el proceso de comunicación en un sistema con arquitectura cliente/servidor.



/ 1. Introducción y contextualización práctica

En esta unidad, seguiremos aprendiendo sobre el modelo de arquitectura del paradigma cliente/servidor.

Veremos en qué consiste este paradigma y cuáles son sus características básicas, así como sus ventajas e inconvenientes a la hora de utilizarlo.

También estudiaremos en detalle cuáles son las funciones específicas, tanto del servidor como del cliente.

Seguidamente, practicaremos con algunos ejemplos de modelos que usan la arquitectura cliente/servidor, como pueden el famoso P2P.

Por último, vamos a aprender cómo se realiza una comunicación básica entre un cliente y un servidor, mediante el concepto de transacción.

Escucha el siguiente audio, en el que planteamos el caso práctico que iremos resolviendo a lo largo de esta unidad.

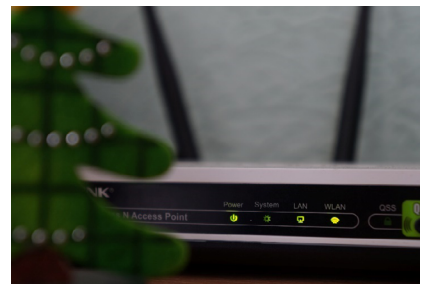


Fig. 1. Comunicación en red.



Audio Intro. "Aplicando conceptos"

<https://bit.ly/3afdJwI>



/ 2. Arquitectura cliente/servidor

A nivel general, cuando se trata de aplicaciones de comunicaciones entre equipos, el modelo más utilizado es el modelo Cliente/Servidor, debido a que ofrece una enorme flexibilidad, interoperabilidad y estabilidad para acceder a recursos que están centralizados en un ordenador. Este modelo apareció por primera vez en los años 80, resolviendo uno de los grandes problemas de la época: un ordenador necesita un servicio que provee otro.

El modelo Cliente/Servidor lo usan aplicaciones tan famosas como WhatsApp, Telegram, navegadores web, etc.

Este modelo está compuesto por los siguientes elementos:

- **Cliente:** Es quien va a interactuar con el usuario, realizando las peticiones que sean necesarias al servidor y mostrando a posteriori los resultados a la persona con la que está interactuando. Las principales funciones que va a cumplir el cliente son:
 - Interactuar con el usuario.
 - Procesar todas las peticiones para comprobar que sean válidas, evitando enviar peticiones malintencionadas al servidor.
 - Recibir los resultados que le envía el servidor.
 - Dar un formato correcto a los datos recibidos para mostrárselos al usuario.
- **Servidor:** Este va a ser quien provea de uno o varios servicios a los clientes. Las funciones que deberá desempeñar son:
 - Aceptar las peticiones de los clientes.
 - Procesar en orden las peticiones de los clientes.
 - Dar un formato correcto a los datos enviados a los clientes.
 - Realizar validaciones de datos.
 - Debe asegurar que la información sea consistente.
 - Mantener seguro el sistema.

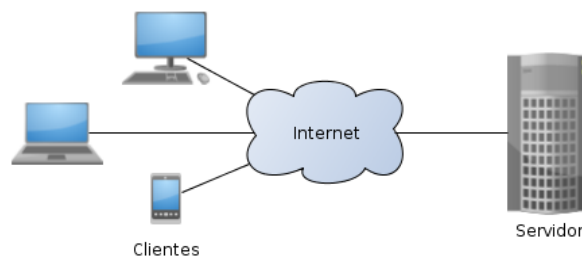


Fig. 2. Modelo Cliente/Servidor.



Audio 1. "Variantes de modelo Cliente/Servidor"

<https://bit.ly/2DXpATQ>





/ 3. Características básicas del modelo cliente/servidor

Dentro de las **aplicaciones que utilizan modelo Cliente/Servidor**, vamos a destacar una serie de características básicas:

- Existe una combinación del cliente con los usuarios, ya que estos son los que interactúan entre sí. También existe una combinación entre el servidor y los recursos que se van a ofrecer, es decir: es el cliente el que deberá proporcionar una interfaz gráfica al usuario y será el servidor el que permita el acceso remoto a los recursos que se compartirán.
- Las operaciones que podrán realizar los clientes y los servidores van a necesitar, claramente, unos requisitos totalmente diferentes en lo que respecta al tratamiento de información, por razones obvias, y será el servidor quien realice todo el trabajo de procesamiento.
- Existe una clara relación entre el cliente y el servidor, pudiéndose ejecutar en un mismo ordenador o en varios ordenadores distribuidos en red.
- Un servidor podrá dar servicio a uno o muchos clientes, realizando siempre las operaciones necesarias para que las peticiones se ejecuten en orden y que ningún cliente se quede sin atender.
- Los servidores van a tener un rol pasivo en este modelo, ya que estarán esperando a que los clientes les envíen sus peticiones, mientras que los clientes tendrán un rol activo en el modelo, ya que serán los que envíen las peticiones cuando las necesiten.
- La comunicación entre cliente y servidor únicamente se puede realizar a través del envío de mensajes, tantos como hagan falta.
- Los clientes podrán utilizar cualquier plataforma para conectarse a los servidores.



Fig. 3. Cliente realizando una petición.

/ 4. Caso práctico 1: “Servidores y clientes suficientes para todos”

Planteamiento: Pilar y José están estudiando detenidamente el modelo que siguen las aplicaciones con la arquitectura de cliente/servidor.

«Es todo muy lógico, cada una de las partes encaja perfectamente en el trabajo que tiene que realizar», le comenta José a Pilar, a lo que ella le responde que lleva razón, pero que tiene una duda: «¿cómo se supone que funcionan las aplicaciones si hay más de un cliente?».

Nudo: ¿Crees que las aplicaciones pueden funcionar con un servidor dando servicio a muchos clientes? ¿Cómo?

Desenlace: Las aplicaciones que usan el modelo cliente/servidor deben ser capaces de dar servicio a todos los clientes que lo necesiten, da igual que sea uno o que sean millones, ¿qué ocurriría si Google solo pudiera dar servicio a mil clientes a la vez?

Esto es algo bastante importante, ya que debemos dar soporte a todos los clientes que lo necesiten y, para ello, probablemente deberemos tener más de un servidor. El uso de varios servidores en la arquitectura cliente/servidor es la norma que predomina, ya que es inconcebible no poder dar soporte a los clientes.

Tenemos, incluso, casos en los que los propios servidores van a actuar como clientes que piden soporte a otros servidores para obtener un servicio del que no disponen y poder dar soporte a sus clientes.

/ 5. Ventajas e inconvenientes del modelo cliente/servidor

Las principales **ventajas del modelo Cliente/Servidor** son las siguientes:

- Utiliza clientes que son ligeros, lo cual significa que no van a necesitar un *hardware* muy potente para funcionar, sino todo lo contrario, ya que quien realizará todos los cálculos necesarios será el servidor.
- Muestra una gran facilidad en la integración entre sistemas que comparten la información, lo cual va a permitir que dispongamos de unas interfaces gráficas muy sencillas de usar para el usuario.
- Predominan las interfaces gráficas que son interactivas, por lo que los usuarios pueden interactuar con el servidor. Haciendo uso de interfaces gráficas conseguiremos transmitir únicamente los datos, consiguiendo una menor congestión en la red.
- El desarrollo y mantenimiento de este tipo de aplicaciones es relativamente sencillo.
- Este modelo es modular, lo que implica que va a permitir la integración de nuevos componentes y tecnologías de una forma muy sencilla, permitiendo un crecimiento escalable.
- Se tiene un acceso centralizado de los recursos en el servidor.
- Los clientes pueden acceder de forma simultánea al servidor, compartiendo los datos entre ellos de una forma rápida y sencilla.

El modelo Cliente/Servidor también presenta algunos **inconvenientes**, como, por ejemplo:

- El mantenimiento de los sistemas es un poco más complejo, ya que interactúan diferentes *hardware* y *software*, lo cual puede ocasionar múltiples errores.
- Se tienen que controlar absolutamente todos los errores posibles del sistema.
- Se tiene que garantizar una buena seguridad en el sistema. Si cae el servidor, cae el servicio para todos sus clientes.
- Se debe garantizar la consistencia de la información transmitida.



Fig. 4. Intruso en el sistema.

/ 6. Modelos cliente/servidor

El modelo Cliente/Servidor se puede clasificar según un parámetro llamado capas, o *triers* en inglés. Según el número de capas que tenga, lo vamos a poder clasificar de las siguientes formas:

- **Sistema de 1 capa:** Vamos a tener este tipo de modelo cuando tanto el cliente como el servidor se encuentren en el mismo ordenador. Este sistema puede no llegar a considerarse como un modelo Cliente/Servidor, ya que, en realidad, no hay una conexión de red para la comunicación del cliente y del servidor.
- **Sistema de 2 capas:** Este es el modelo tradicional en lo que se refiere a Cliente/Servidor y es el que vamos a utilizar en nuestras aplicaciones. Aquí existen un cliente y un servidor totalmente identificables. Este modelo tiene un gran inconveniente: tiene una bajísima escalabilidad, pudiendo llegar a sobrecargarse si hay muchos clientes o si los clientes son pocos, pero mandan una gran cantidad de peticiones al servidor.



- **Sistema de 3 capas:** Este modelo fue diseñado para mejorar las limitaciones del modelo de 2 capas, añadiendo, para ello, una nueva capa con un nuevo servidor, que puede ser o bien un servidor de aplicaciones, el cual se encargará de interactuar con los clientes y de enviar las peticiones al servidor de datos; o bien un servidor de datos, que será el encargado de recibir las peticiones del servidor de aplicación con lo requerido por los clientes para resolverlas. Sería posible agregar tantos servidores de datos como necesitemos sin problema alguno para mejorar el rendimiento del sistema.
- **Sistema de n capas:** Este modelo fue diseñado para mejorar las limitaciones del modelo de 3 capas y, a partir de aquí, podremos añadir tantas capas de servidores como sean necesarias, permitiendo separar funcionalidades y mejorar el rendimiento del sistema de una forma considerable. Un ejemplo de este tipo de sistemas son los conocidos programas o aplicaciones [P2P](#), como BitTorrent, Skype o Bitcoin.

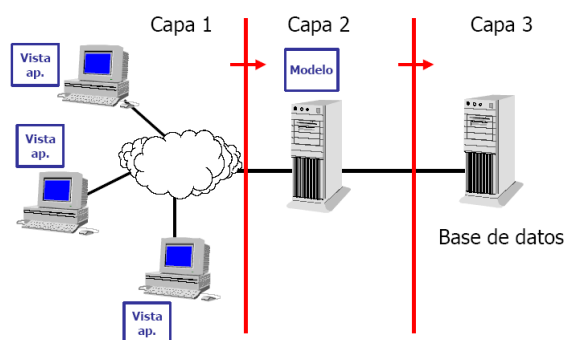


Fig. 5. Arquitectura de 3 capas o 3-triers

/ 7. Comunicación en el modelo cliente/servidor

El modelo Cliente/Servidor tiene un **sistema de comunicación íntegro a través de intercambio de mensajes**. Estos mensajes contendrán toda la información necesaria para poder ser identificados y enviados a donde se les solicite.

En este sistema, la pérdida de mensajes es un factor importantísimo a tener en cuenta, ya que podríamos estar enviando un fichero que ocupa 3 MB en varios mensajes. Si se pierden unos cuantos por el camino, porque no sean bien redireccionados, por ejemplo, el resultado de la reconstrucción del fichero de 3 MB estaría corrupto, por lo que quedaría inutilizable. Para evitar que se pierdan mensajes en este tipo de comunicaciones, se enviará por cada mensaje recibido un mensaje de confirmación al emisor. Estos son los mensajes conocidos como **ACK o acknowledgement** en inglés. En el caso de que un mensaje no llegue correctamente al receptor o se demore mucho tiempo su llegada (se considerará que se ha producido algún error), no se enviará el mensaje de ACK al emisor; en consecuencia, este entenderá que el mensaje no ha llegado, por lo que lo volverá a enviar y esperará de nuevo la llegada de su ACK.

El método de los envíos de ACK es muy **efectivo y útil**, pero la verdad es que es bastante lento, ya que deberemos esperar absolutamente todos los ACK de todos los mensajes que enviemos, además de que se producirá más tráfico en la red, con la posible congestión que puede implicar. Esto se puede mejorar de una forma muy simple, ya que, si permitimos que el receptor envíe varios ACK a la vez, en lugar de enviarlos de uno en uno, conseguiremos una mejora considerable en la transmisión de la información. Con este método, el emisor podrá enviar paquetes de X mensajes y esperará la llegada de sus correspondientes ACK en grupo. Debemos tener en cuenta que con este método de enviar varios paquetes y recibir varios ACK, puede suceder que tanto los paquetes como los ACK lleguen desordenados, o que, incluso, alguno de ellos se pierda por el camino, siendo necesario llevar un control de errores en esta técnica. Para gestionar estos errores, podremos utilizar un vector de ACK, compuesto por pares que indican el número del mensaje enviado y si se ha recibido o no su correspondiente ACK.



Vídeo 1. "Procedimiento para programar una aplicación con el modelo Cliente/Servidor"
<https://bit.ly/3gQoE2p>



/ 8. Caso práctico 2: “¿Cómo implementar una aplicación cliente/servidor?”

Planteamiento: Nuestros amigos Pilar y José han recibido su primera tarea sobre aplicaciones que usan el modelo cliente/servidor. Esta consiste en una aplicación muy simple. Deberá ser capaz de dar un servicio de hora del sistema, es decir, habrá un servidor que se encargará de proporcionar la hora del sistema a los clientes que se conecten a él. Por ahora, solo tienen que realizar el diseño de la aplicación, sin tener que implementar nada de código.

Pilar no tiene claro cómo empezar y le pregunta a José qué tal va con la práctica, a lo que él le responde que tampoco va muy bien.

Nudo: ¿Qué piensas al respecto? ¿Crees que Pilar y José deberían diseñar juntos la aplicación en vez de pensarla cada uno por su cuenta?

Desenlace: Es más que conocido que el modelo cliente/servidor tiene dos partes: la del cliente y la del servidor propiamente dicha.

Cuando se diseña una aplicación de este tipo, lo más normal es que sea bastante compleja, por lo que habrá dos equipos de desarrollo, uno para desarrollar la parte del servidor y otro que se encargue de desarrollar la parte del cliente. Con este tipo de desarrollo, se gana mucho tiempo, ya que lo único en lo que tendrán que estar de acuerdo será en la forma en la que se comunican los clientes con el servidor, si hay algún mensaje de inicio, de confirmación, de parada... y de situaciones sobre comunicación por el estilo.

En el caso de nuestros amigos, aunque la aplicación sea muy simple, sería recomendable seguir este tipo de diseño, definiendo cada uno una parte y poniéndolas en común.



Fig. 6. Desarrollo en equipo.

/ 9. Diagramas de estados

Cuando utilizamos el **modelo Cliente/Servidor** para programar aplicaciones, uno de los **fallos principales** con respecto a la seguridad que nos vamos a encontrar son:

- Que un cliente pueda realizar operaciones no autorizadas. En este caso, el servidor no va a poder procesar una petición enviada por un cliente que no tiene privilegios para ello, por ejemplo: un cliente que se encarga de realizar peticiones de precios de productos, de repente empieza a realizar peticiones de *login*.
- Que los mensajes no estén bien formados. Puede ocurrir, por ciertos eventos, que un cliente envíe al servidor un mensaje y que éste no esté formado, bien porque le falte información necesaria o porque la información que contenga sea errónea.

Estos errores deben evitarse, por razones obvias de **seguridad del sistema** y, para ello, es de vital importancia que se utilice un diagrama de estados o autómatas, para crear el comportamiento que desarrollará el servidor.

Mediante los autómatas o diagramas de estados, podremos definir todos y cada uno de los estados o comportamientos por los que va a pasar nuestro servidor, definiendo qué información debe llegar a cada uno de ellos, hacia qué estado se dirigirá cuando termine su ejecución y qué comportamientos deberá tener cuando llegue información errónea a cada uno de los estados posibles en su ejecución.



Para ello, deberemos definir en nuestro servidor dos variables, para almacenar tanto el estado en el que nos encontramos, como el comando que queremos ejecutar de una lista de posibles casos.

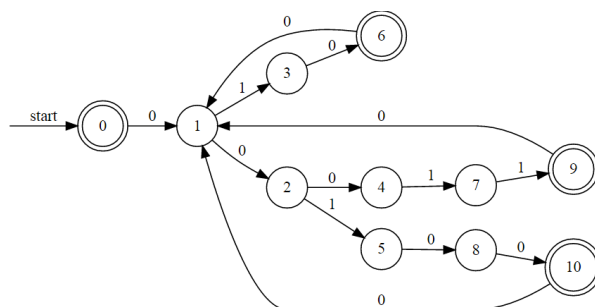


Fig. 7. Autómata simple.



Vídeo 2. "Conexión TCP simple paso a paso"

<https://bit.ly/2FegrqI>



/ 10. Resumen y resolución del caso práctico de la unidad

A lo largo de esta unidad, hemos visto en profundidad el modelo de arquitectura del paradigma **cliente/servidor**.

Hemos aprendido en qué consiste este paradigma y cuáles son sus características básicas más importantes, así como qué **ventajas e inconvenientes** tiene a la hora de utilizarlo.

También hemos estudiado más en detalle cuáles son las **funciones específicas** que debe ofrecer, tanto del servidor como del cliente.

Seguidamente, hemos conocido algunos de los **modelos** que utilizan la arquitectura del modelo cliente/servidor como, por ejemplo, el famoso P2P, uno que probablemente hayamos utilizado todos.

Finalmente, hemos visto cómo se realiza una **comunicación básica** entre un cliente y un servidor, desde la sincronización de ambos, al envío de un paquete, a través del diagrama de estados.

Resolución del caso práctico de la unidad

Para realizar el ejercicio que tienen pendiente nuestros amigos, hace falta conocer bien el modelo cliente/servidor. Una vez sabemos qué es un cliente y qué es un servidor y qué roles juega cada uno de ellos, este problema se vuelve bastante sencillo de resolver.

Lo óptimo para resolverlo sería crear, en primer lugar, un servidor que pueda ofrecer los dos servicios necesarios, el de la fecha correcta y el de la diferencia de días entre dos fechas. Una vez tengamos el servidor, tendremos que crear un cliente que se comunique con el servidor, pidiendo el servicio que necesite.

Si queremos optimizar un poco el diseño, podríamos crear un servidor de 2 capas, una para cada uno de los servicios que necesitamos ofrecer, haciendo que el diseño sea más eficiente.



/ 11. Bibliografía

Arquitectura 3 capas. (23 de febrero de 2014). *Programación web*. [Entrada de Blog]. Disponible en: <https://edgarbc.wordpress.com/arquitectura/>

Arquitectura cliente-servidor - RedesOrdenadores_GrupoC. (s.f.). Recuperado el 17 de julio de 2020 de: <https://sites.google.com/site/redesordenadoresgrupoc/home/arquitectura-cliente-servidor>

Cliente-servidor, en Wikipedia, la enciclopedia libre. Disponible en: <https://es.wikipedia.org/wiki/Cliente-servidor>

Gómez, O. (2016). *Programación de Servicios y Procesos* Versión 2.1. https://github.com/OscarMaestre/ServiciosProcesos/blob/master/_build/latex/ServiciosProcesos.pdf

Sánchez, J. M. y Campos, A. S. (2014). *Programación de servicios y procesos*. Madrid: Alianza Editorial.

WUOLAH