Patrones de arquitectura de software

La **programación por capas** es una arquitectura [cliente-servidor](https://es.wikipedia.org/wiki/Cliente-servidor) en el que el objetivo primordial es la separación de la lógica de negocios de la lógica de diseño; un ejemplo básico de esto consiste en separar la capa de datos de la capa de presentación al usuario.

[](https://commons.wikimedia.org/wiki/File:Tres_capas.PNG)

La ventaja principal de este estilo es que el desarrollo se puede llevar a cabo en varios niveles y, en caso de que sobrevenga algún cambio, solo se ataca al nivel requerido sin tener que revisar entre código mezclado. Un buen ejemplo de este método de programación sería el [modelo de interconexión de sistemas abiertos](https://es.wikipedia.org/wiki/Modelo_de_interconexi%C3%B3n_de_sistemas_abiertos).

Además, permite distribuir el trabajo de creación de una aplicación por niveles; de este modo, cada grupo de trabajo está totalmente[abstraído](https://es.wikipedia.org/wiki/Abstracci%C3%B3n_(programaci%C3%B3n_orientada_a_objetos)) del resto de niveles, de forma que basta con conocer la [API](https://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones) que existe entre niveles.

En el [diseño](https://es.wikipedia.org/wiki/Dise%C3%B1o) de [sistemas informáticos](https://es.wikipedia.org/wiki/Sistema_inform%C3%A1tico) actual se suelen usar las [arquitecturas multinivel](https://es.wikipedia.org/wiki/Arquitectura_software) o Programación por capas. En dichas arquitecturas a cada nivel se le confía una misión simple, lo que permite el diseño de arquitecturas escalables (que pueden ampliarse con facilidad en caso de que las necesidades aumenten).

El más utilizado actualmente es el diseño en tres niveles (o en tres capas).

Capas y niveles

1. **Capa de presentación:** la que ve el usuario (también se la denomina "capa de usuario"), presenta el sistema al usuario, le comunica la información y captura la información del usuario en un mínimo de proceso (realiza un filtrado previo para comprobar que no hay errores de formato). También es conocida como interfaz gráfica y debe tener la característica de ser "amigable" (entendible y fácil de usar) para el usuario. Esta capa se comunica únicamente con la capa de negocio.
2. **Capa de negocio:** es donde residen los [programas](https://es.wikipedia.org/wiki/Programa_(computaci%C3%B3n)) que se ejecutan, se reciben las peticiones del usuario y se envían las respuestas tras el proceso. Se denomina capa de negocio (e incluso de lógica del negocio) porque es aquí donde se establecen todas las reglas que deben cumplirse. Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de datos, para solicitar al gestor de [base de datos](https://es.wikipedia.org/wiki/Base_de_datos) almacenar o recuperar datos de él. También se consideran aquí los programas de aplicación.
3. **Capa de datos:** es donde residen los datos y es la encargada de acceder a los mismos. Está formada por uno o más gestores de bases de datos que realizan todo el almacenamiento de datos, reciben solicitudes de almacenamiento o recuperación de información desde la capa de negocio.

Todas estas capas pueden residir en un único ordenador, si bien lo más usual es que haya una multitud de ordenadores en donde reside la capa de presentación (son los clientes de la arquitectura cliente/servidor). Las capas de negocio y de datos pueden residir en el mismo ordenador, y si el crecimiento de las necesidades lo aconseja se pueden separar en dos o más [ordenadores](https://es.wikipedia.org/wiki/Ordenador). Así, si el tamaño o complejidad de la base de datos aumenta, se puede separar en varios ordenadores los cuales recibirán las peticiones del ordenador en que resida la capa de negocio.

Si, por el contrario, fuese la complejidad en la capa de negocio lo que obligase a la separación, esta capa de negocio podría residir en uno o más ordenadores que realizarían solicitudes a una única base de datos. En sistemas muy complejos se llega a tener una serie de ordenadores sobre los cuales corre la capa de negocio, y otra serie de ordenadores sobre los cuales corre la base de datos.

En una arquitectura de tres niveles, los términos "capas" y "niveles" no significan lo mismo ni son similares.

El término "capa" hace referencia a la forma como una solución es segmentada desde el punto de vista lógico:

* **Presentación**. (Conocida como capa Web en aplicaciones Web o como capa de usuario en Aplicaciones Nativas)
* **Lógica de Negocio**. (Conocida como capa Aplicativa)
* **Datos**. (Conocida como capa de Base de Datos)

En cambio, el término "nivel" corresponde a la forma en que las capas lógicas se encuentran distribuidas de forma física. Por ejemplo:

* Una solución de tres capas (presentación, lógica del negocio, datos) que residen en un solo ordenador (Presentación+lógica+datos). Se dice que la arquitectura de la solución es de tres capas y *un nivel*.
* Una solución de tres capas (presentación, lógica del negocio, datos) que residen en dos ordenadores (Presentación+lógica por un lado; lógica+datos por el otro lado). Se dice que la arquitectura de la solución es de tres capas y *dos niveles*.

La **Arquitectura dirigida por eventos**, *Event-drivenarchitecture* o EDA, es un patrón de [arquitectura software](https://es.wikipedia.org/wiki/Arquitectura_software) que promueve la producción, detección, consumo de, y reacción a [eventos](https://es.wikipedia.org/wiki/Evento_estad%C3%ADstico).

Un evento puede ser definido como "un cambio significativo en un estado". Por ejemplo, cuando un consumidor compra un coche, el estado del coche pasa de "se vende" a "vendido". La arquitectura del sistema del vendedor de coches debe tratar este cambio de estado como un evento, cuyo suceso puede ser conocido en otras aplicaciones en la arquitectura. Desde una perspectiva formal, lo que es producido, publicado, propagado, detectado o consumido es un mensaje (típicamente asíncrono) llamado notificación del evento, y no el evento en si mismo, el cuál es el cambio de estado que disparó la emisión del evento. Los eventos no viajan, solamente ocurren. Por otro lado, el término evento es frecuentemente usado para denotar el mensaje de notificación en sí mismo, lo cual puede llevar a algún tipo de confusión.

Este patrón arquitectónico puede ser aplicado por el diseño e implementación de aplicaciones y sistemas que transmitan eventos entre componentes software que estén emparejados libremente y servicios. Un sistema dirigido por eventos está compuesto típicamente de emisores de eventos (o agentes) y consumidores de eventos (o "sink" en inglés). Los consumidores tienen la responsabilidad de llevar a cabo una reacción tan pronto como el evento esté presente. La reacción puede o no puede ser completamente proporcionada por el consumidor en sí mismo. Por ejemplo, el consumidor debe tener solamente la responsabilidad de filtrar, transformar y reenviar el evento a otro componente o debe proporcionar una reacción propia a algún evento.

Construir aplicaciones y sistemas alrededor de una arquitectura dirigida por eventos permite a estas aplicaciones y sistemas ser construidos de una manera que facilita un mayor grado de reacción, debido a que los sistemas dirigidos por eventos están, por el diseño, más normalizados para entornos no predecibles y asíncronos.

La arquitectura dirigida por eventos puede complementar la arquitectura orientada a servicios (SOA) porque los servicios pueden ser activados por disparadores que se encuentran en eventos entrantes. Este paradigma es particularmente útil cuando el consumidor no proporciona algún contenedor ejecutivo propio.

SOA 2.0 engloba las implicaciones de las arquitecturas SOA y EDA proporcionando a un más rico y más robusto nivel, creando un nuevo patrón de eventos. Este nuevo concepto de disparadores de patrones de inteligencia promueve a humanos autónomos o procesamiento automático que añade valor exponencial al negocio. Esto se debe a que se inyecta información de valor añadido en patrón reconocido que no podía haber sido obtenido previamente.

La maquinaría computacional y los sensores (como sensores de cualquier tipo, actuadores, controladores,...) pueden detectar cambios de estado de objetos o condiciones y crear eventos que pueden ser procesados por un servicio o un sistema. Los disparadores de eventos son condiciones que tienen como resultado la creación de un evento.

La **Arquitectura Orientada a Servicios** (**SOA**, siglas del inglés *ServiceOrientedArchitecture*) es un paradigma de arquitectura para diseñar y desarrollar [sistemas distribuidos](https://es.wikipedia.org/wiki/Sistemas_distribuidos). Las soluciones  SOA  han sido creadas para satisfacer los objetivos de negocio las cuales incluyen facilidad y flexibilidad de integración con sistemas legados, alineación directa a los [procesos de negocio](https://es.wikipedia.org/wiki/Procesos_de_negocio) reduciendo costos de implementación, innovación de servicios a clientes y una adaptación ágil ante cambios incluyendo reacción temprana ante la competitividad.

Permite la creación de [sistemas de información](https://es.wikipedia.org/wiki/Sistemas_de_informaci%C3%B3n) altamente escalables que reflejan el negocio de la organización, a su vez brinda una forma bien definida de exposición e invocación de servicios (comúnmente pero no exclusivamente [servicios web](https://es.wikipedia.org/wiki/Servicio_Web)), lo cual facilita la interacción entre diferentes sistemas propios o de terceros.

SOA proporciona una metodología y un marco de trabajo para documentar las capacidades de negocio y puede dar soporte a las actividades de integración y consolidación.

En [computación](https://es.wikipedia.org/wiki/Computaci%C3%B3n), un **micronúcleo** (en [inglés](https://es.wikipedia.org/wiki/Idioma_ingl%C3%A9s), **microkernel** o **μkernel**) es un tipo de [núcleo](https://es.wikipedia.org/wiki/N%C3%BAcleo_(inform%C3%A1tica)) de un [sistema operativo](https://es.wikipedia.org/wiki/Sistema_operativo) que provee un conjunto de primitivas o llamadas mínimas al sistema para implementar servicios básicos como espacios de direcciones, [comunicación entre procesos](https://es.wikipedia.org/wiki/Comunicaci%C3%B3n_entre_procesos) y [planificación](https://es.wikipedia.org/wiki/Planificador) básica. Todos los otros servicios ([gestión de memoria](https://es.wikipedia.org/wiki/Gesti%C3%B3n_de_memoria), [sistema de archivos](https://es.wikipedia.org/wiki/Sistema_de_archivos), operaciones de E/S, etc.), que en general son provistos por el núcleo, se ejecutan como procesos servidores en espacio de usuario.

El paradigma del micronúcleo, tuvo una gran relevancia académica durante los años ochenta y principios de los noventa, dentro de lo que se denominó [selfhealingcomputing](https://es.wikipedia.org/w/index.php?title=Self_healing_computing&action=edit&redlink=1" \o "Self healing computing (aún no redactado)), esto es, sistemas independientes que fuesen capaces de superar por si mismos errores de [software](https://es.wikipedia.org/wiki/Software) o [hardware](https://es.wikipedia.org/wiki/Hardware). En un principio pretendía ser una solución a la creciente complejidad de los sistemas operativos.

Las principales ventajas de su utilización son la reducción de la complejidad, la descentralización de los fallos (un fallo en una parte del sistema no se propagaría al sistema entero) y la facilidad para crear y depurar [controladores de dispositivos](https://es.wikipedia.org/wiki/Controlador_de_dispositivo). Según los defensores de esta tendencia, esto mejora la tolerancia a fallos y eleva la portabilidad entre plataformas de hardware.

Por otro lado, sus principales dificultades son la complejidad en la sincronización de todos los módulos que componen el micronúcleo y su acceso a la memoria, la anulación de las ventajas de [Zero Copy](https://es.wikipedia.org/wiki/Zero_Copy" \o "Zero Copy) y la integración con las aplicaciones. Además, los procesadores y arquitecturas modernas de hardware están optimizadas para sistemas de núcleo que pueden mapear toda la memoria. Sus detractores le achacan también y fundamentalmente, mayor complejidad en el código, menor rendimiento y limitaciones en diversas funciones.