Arquitectura de software

Profesor:
Máster Carlos Carranza Blanco
Programación III
Ingeniería del Software

11/8/2022

Definición

- *El diseño de la Arquitectura de un Software es el proceso por el cual se define una solución para los requisitos técnicos y operacionales del mismo.*
- *Este proceso define qué componentes forman el software, cómo se relacionan entre ellos, y cómo mediante su interacción llevan a cabo la funcionalidad especificada, cumpliendo con los criterios previamente establecidos; como seguridad, disponibilidad, eficiencia o usabilidad.*

Fundamentos

- *Durante el diseño de la arquitectura se tratan tópicos que puedan provocar un impacto importante en el éxito o fracaso de nuestro software. Son esenciales realizar las siguientes interrogantes para cubrir este punto:*
  - *¿En qué entorno va a ser desplegado nuestro sistema?*
  - *¿Cómo va a ser nuestro sistema puesto en producción?*
  - *¿Cómo van a utilizar los usuarios nuestro sistema?*
  - *¿Qué otros requisitos debe cumplir el sistema? (seguridad, rendimiento, concurrencia, configuración…)*
  - *¿Qué cambios en la arquitectura pueden impactar al sistema ahora o una vez desplegado?*

Fundamentos…

- *Para diseñar la arquitectura de un sistema es importante tener en cuenta:*

- *Los intereses de los distintos agentes que participan. Estos agentes son los usuarios del sistema, el propio sistema y los objetivos del negocio.*

- *Cada uno de ellos establece requisitos y restricciones que deben tomarse en cuenta para el diseño de la arquitectura, los que en algún momento podrían entrar en conflicto.*

Fundamentos…

- *Para los usuarios es importante que el software responda a la interacción de una forma fluida, mientras que para los objetivos del negocio es importante que el software cueste poco.*

- *Los usuarios pueden querer que se implemente primero una funcionalidad útil para su trabajo del día a día, mientras que el software puede tener prioridad en que se implemente la funcionalidad que permita definir su estructura.*

Fundamentos…

- *He aquí, que el trabajo del arquitecto es delinear los escenarios y requisitos de calidad importantes para cada agente así como los puntos clave que debe cumplir y las acciones o circunstancias que no deben ocurrir.*
- *El objetivo final de la arquitectura es identificar los requisitos que producen un impacto en la estructura del software y reducir los riesgos asociados con la construcción del mismo.*
- *La arquitectura debe soportar los cambios futuros del software, del hardware y de funcionalidad demandada por los clientes (que ocurren muy a menudo). Del mismo modo, es responsabilidad del arquitecto analizar el impacto de sus decisiones de diseño y establecer un compromiso entre los diferentes requisitos de calidad así como entre los compromisos necesarios para satisfacer a los usuarios, al software y los objetivos del negocio.*

Fundamentos…

- *Finalmente, resumamos que la Arquitectura de Software debería poseer las siguientes capacidades:*
- *Mostrar la estructura del software, pero ocultando los detalles.*
- *Concebir y diseñar todos los casos de uso.*
- *Satisfacer en la medida de lo posible los intereses de los agentes.*
- *Ocuparse de los requisitos funcionales y de calidad.*
- *Determinar el tipo de software a desarrollar.*
- *Determinar los estilos arquitecturales que se usarán.*
- *Tratar las principales cuestiones transversales.*

Fundamentos…

- *Consideramos el dibujo arquitectónico en un proyecto de software la propia estructura de módulos y carpetas o paquetes en el caso de Java o cualquier añadido que ayude a expresar la intención de nuestro sistema sin expresar el cómo está hecha.*
- *Robert C. Martin(Uncle Bob) define una serie de "arquitecturas limpias" que tienen una serie de objetivos en común:*
- ***Independiente de los frameworks.** La existencia de esta forma de construir cosas en el sistema no depende de un framework.*

Fundamentos…

- ***Testable.*** *Tu arquitectura hace que tu código pueda ser testado.*
- ***Independiente de la UI.*** *Tus reglas de negocio no se ven alteradas por un requerimiento de UI, cuando desarrollas una funcionalidad nueva es la UI la que se adapta a tus reglas de negocio y nunca al revés.*
- ***Independiente de la base de datos.*** *Puedes cambiar el motor de persistencia ya que tus reglas de negocio no son dependientes de la implementación concreta de la base de datos sino que es la base de datos la que se adapta a estas reglas.*
- ***Independiente de cualquier componente externo.*** *Se aplica la misma regla descrita en la base de datos pero relacionada a componentes externos así como integraciones con otros sistemas, librerías, etc…*

Si una arquitectura cumple estos objetivos podría entrar en el grupo de arquitecturas limpias.

Casos de uso

- ***Un caso de uso es una acción que un usuario o agente externo realiza en nuestro sistema. Se nombran siempre con un verbo + nombre***. *Si estuviéramos desarrollando una aplicación en la que se venden productos, ListarProductos sería un nombre válido así como ComprarProducto.*
- *Una manera común de identificar la intención de nuestro sistema es modelarlo en base a sus casos de uso. Son la forma de acceder a las reglas de negocio de nuestra aplicación, por lo tanto podríamos decir que representan la parte pública del dominio de nuestra aplicación. Al ser la parte pública hacen de entrada/salida de datos al dominio.*

Casos de uso…

- ***Los casos de uso nos ayudan a expresar con un lenguaje más natural las acciones posibles que nuestro sistema puede realizar*** *y de esa forma listan las funcionalidades del mismo.*

- *Un listado de los casos de uso ordenados por funcionalidad nos ayudará a saber de qué trata la aplicación con la que estamos trabajando.*

Reglas de negocio

- *Una regla de negocio es un requerimiento del encargado en definir cómo funciona una aplicación. Definen el comportamiento de nuestro sistema y cómo reacciona a las acciones por parte de un usuario o un agente externo si tuviera que interactuar con otros sistemas. Las reglas de negocio aportan valor al sistema que estamos construyendo.*
- *Como hemos descrito hasta ahora nuestras reglas de negocio son el núcleo de nuestra arquitectura, todo depende de estas reglas, ya que al fin y al cabo creamos software para ofrecer una solución a una necesidad.*

¿Por qué es tan importante separar nuestras reglas de negocio?

- *El software evoluciona, los frameworks, base de datos, librerías que usamos son solo herramientas con las que construimos un sistema, pero tal como van evolucionando, la forma de usarlas cambia o incluso son reemplazadas por unas mejores, será mucho más fácil reemplazar estas nuevas herramientas en nuestro sistema si están lo más aisladas posible.*
- *Existe una única regla para usar una arquitectura limpia, la regla de dependencias. Esta regla expresa que la dependencia entre componentes de nuestro sistema debe de ser desde los detalles de implementación a nuestro dominio y nunca dejar que nuestro dominio conozca estos detalles. Esta es la herramienta que tenemos para aislar lo que realmente aporta valor a nuestra aplicación de la implementación o tecnologías que usamos.*

Reglas de negocio…

- *Un cliente difícilmente te va a explicar exactamente las reglas de su negocio, seguramente exprese lo que espera encontrar o cómo va a reaccionar la aplicación en determinado caso, somos nosotros los desarrolladores los encargados de traducir su lenguaje y extraer las reglas de negocio y modelar la aplicación en base a lo que llamamos un modelo de datos de dominio, siendo el dominio el negocio del que estamos hablando.*
- ***Si cambiamos partes de nuestro sistema, deberíamos tener la seguridad de que estamos haciendo cambios sin dañar otras, para eso necesitamos que nuestra arquitectura sea testable.***

Testing

- *Toda esta separación entre reglas de negocio y el resto de partes de nuestro sistema hacen que nuestros componentes sean testables ya que comunicamos las piezas que componen este puzzle con abstracciones.*

- *Estas características nos permite cambiar el detalle de implementación de una clase lo que hace que nuestra arquitectura sea independiente de las herramientas que utilicemos, ya que hay que distinguir entre el qué hacer (abstracción) y el cómo hacerlo (implementación).*

Independiente de agentes externos

- *Nuestra aplicación, como hemos comentado, debe ser independiente de agentes externos tales como frameworks, bases de datos, UI, APIs externas u otros sistemas que no tenemos control.*
- *Para cumplir con esta regla debemos tener claro que cualquier agente externo puede ser dañino, ya que en cualquier momento puede ser reemplazado por otro que pueda cumplir con la misma misión en nuestro sistema.*
- *Todo framework o librería que nos haga desarrollar código utilizando formas que no son estandars en el lenguaje que estamos desarrollando deberían ser cubiertas con una abstracción, de forma que si en un futuro queremos cambiar esa librería por otra o incluso implementarlo nosotros mismos lo podamos hacer solo haciendo un cambio en cómo se crea la implementación para esa abstracción.*

Arquitectura de clases

- *El objetivo es crear una arquitectura de objetos que sirva como base para el diseño del sistema.*
- *Dependiendo del tipo de aplicación existen varias arquitecturas. Ellas se distinguen según la organización de los objetos de acuerdo a su funcionalidad. Esto es llamado dimensión de arquitectura.*
- *Dimensión de la arquitectura:*
  - ***Unidimensional:*** *un solo grupo de objetos para manejar la funcionalidad y la interacción externa.*
  - ***Bidimensional:*** *un grupo de objetos para manejar la funcionalidad y otros para las interacciones externas.*
  - ***Tridimensional:*** *La más usada en los sistemas de información que siendo el Modelo-Vista-Control.*

Arquitectura de clases

- *Arquitectura Modelo-Vista-Control:*
  - *Es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz del usuario y la lógica de negocio en tres componentes distintos.*
  - *El modelo es el sistema de gestión de base de datos y la lógica de negocio y el controlador es el responsable de recibir los eventos de entrada desde la vista.*

Identificación de clases según Estereotipos

- *La arquitectura de objetos debe considerar los tres tipos de estereotipos de objetos. Para ello se deben identificar primero las clases borde, las clases entidad y finalmente las de Control.*
- *En general, los cambios más comunes a un sistema son los de funcionalidad y bordes. Los cambios de las interfaces deben afectar solo a los objetos borde.*
- *Los cambios a la funcionalidad son más dificiles de administrar, ya que ésta puede abarcar todos los tipos de objetos.*

Identificación de clases según Estereotipos…

- ***Clases Bordes(interfaz o boundaries):***
  - *Toda la funcionalidad especificada en las descripciones de los casos de uso que depende directamente de los aspectos externos del sistema, se ubica en los objetos borde, pues a través de ellos se comunican los actores con el sistema.*
  - *La tarea de una clase borde es traducir los eventos generados por un actor en eventos comprendidos por el sistema, y traducir los eventos del sistema en una presentación comprensible para el actor.*
  - *Las clases borde en otras palabras, describen la comunicación bidireccional entre el sistema y los actores.*

Identificación de clases según Estereotipos…

- ***Clases Entidad(entities):***
  - *Se utilizan objetos entidad para modelar la información que el sistema debe manejar a corto y largo plazo. La información a corto plazo existe durante la ejecución de un caso de uso, mientras que la información a largo plazo trasciende los caso de uso, por lo que es necesario guardarla en alguna base de datos o archivos.*
  - *Durante la identificación de objeto entidad, se encontrara que objetos que objetos similares aparecen en varios casos de uso*

Identificación de clases según Estereotipos…

- ***Clases Control:***
  - *En la mayoría de los casos de uso, existe un comportamiento que no se puede asignar de forma natural a ninguno de los otros dos tipos de objetos ya vistos. Una posibilidad es repartir el comportamiento entre los dos tipos de objetos, pero la solución no es buena si se considera el aspecto de extensibilidad.*
  - *Un cambio en el comportamiento podría afectar varios objetos, dificultando su modificación. Para evitar estos problemas, tal comportamiento se asigna a objetos control.*
  - *Es difícil lograr un buen balance en la distribución del comportamiento del caso de uso entre los objetos entidad, borde y control. Los objetos de control normalmente proveen a administración de los demás tipos de objetos.*

Estilo Arquitectural N-Capas (N-Layer)

- Se basa en una distribución jerárquica de los roles y las responsabilidades para proporcionar una división efectiva de los problemas a resolver.
- Las capas de una aplicación pueden residir en la misma máquina o pueden estar distribuidos entre varios equipos.
- Separa de forma clara la funcionalidad de cada capa.

Beneficios…

- **Abstracción** ya que los cambios se realizan a alto nivel y se puede incrementar o reducir el nivel de abstracción que se usa en cada capa del modelo.
- **Aislamiento** ya que se pueden realizar actualizaciones en el interior de las capas sin que esto afecte al resto del sistema.
- **Rendimiento** ya que distribuyendo las capas en distintos niveles físicos se puede mejorar la escalabilidad, la tolerancia a fallos y el rendimiento.
- **Testeabilidad** ya que cada capa tiene una interfaz bien definida sobre la que realizar las pruebas y la habilidad de cambiar entre diferentes implementaciones de una capa.
- **Independencia** ya que elimina la necesidad de considerar el hardware y el despliegue así como las dependencias con interfaces externas.

Cuando usarlo…

- Ya tienes construidas capas de una aplicación anterior, que pueden reutilizarse o integrarse.
- Ya tienes aplicaciones que exponen su lógica de negocio a través de interfaces de servicios.
- La aplicación es compleja y el alto nivel de diseño requiere la separación para que los distintos equipos puedan concentrarse en distintas áreas de funcionalidad.
- La aplicación debe soportar distintos tipos de clientes y distintos dispositivos.
- Quieres implementar reglas y procesos de negocio complejos o configurables.

Estilo Arquitectural N-Niveles(N-Tier)

- Define la separación de la funcionalidad en segmentos/niveles físicos separados, similar que el estilo en N-Capas pero sitúa cada segmento en una máquina distinta. En este caso hablamos de Niveles físicos (Tiers).
- Separación de niveles físicos (Servidores normalmente) por razones de escalabilidad, seguridad, o simplemente necesidad,

Beneficios…

- **Mantenibilidad** ya que cada nivel es independiente de los otros las actualizaciones y los cambios pueden ser llevados a cabo sin afectar a la aplicación como un todo.
- **Escalabilidad** porque los niveles están basados en el despliegue de capas realizar el escalado de la aplicación es bastante directo.
- **Disponibilidad** ya que las aplicaciones pueden redundar cualquiera de los niveles y ofrecer así tolerancia a fallos.

Cuando usarlo…

- Los requisitos de procesamiento de las capas de la aplicación difieren.
- Los requisitos de seguridad de las capas de la aplicación difieren.
- Quieres compartir la lógica de negocio entre varias aplicaciones.
- Tienes el suficiente hardware para desplegar el número necesario de servidores en cada nivel.

Manejo de Capas y Patrones de Diseño

- *Cuando se construye software como producto empresarial o comercial, se llevan a cabo varias técnicas de manera que el desarrollo se haga en forma ordenada y así poder asegurar un avance continuo del proyecto, un producto final de calidad, y además realizar posteriores mejoras sea una tarea más fácil.*
- *Una de las más utilizadas se llama la programación por capas, que consiste en dividir el código fuente según su funcionalidad principal.*

Manejo de Capas y Patrones de Diseño …

- *La programación para lograr sacarle el mayor provecho a la programación por capas se necesita seguir una serie de pasos complejos los cuales primeramente deben ser definidos para cada proyecto en específico.*
- *Luego deben ser revisados para asegurarse de que el modelo adoptado cumpla con las normas necesarias para que la aplicación sea del agrado del usuario, y por último debe ser implementado por el grupo de desarrollo encargado para tal fin, los cuales siguiendo el modelo propuesto obtienen una herramienta útil para facilitar la labor de programación dividiendo la aplicación en módulos y capas fáciles de pulir.*

Características de la Programación en Capas

- *La programación por capas es una técnica de ingeniería de software propia de la programación por objetos, éstos se organizan principalmente en 3 capas: la capa de presentación o frontera, la capa de lógica de negocio o control, y la capa de datos.*
- *Siguiendo el modelo, el desarrollador se asegura avanzar en la programación del proyecto de una forma ordenada, lo cual beneficia en cuanto a reducción de costos por tiempo, al ser dividida la aplicación general en varios módulos y capas que pueden ser tratados de manera independiente y hasta en forma paralela.*

Capa de Presentación

- *La presentación del programa ante el usuario, debe manejar interfaces que cumplan con el objetivo principal de este componente, el cual es facilitar al usuario la interacción con la aplicación.*
- *Dentro de la parte técnica, la capa de presentación contiene los objetos encargados de comunicar al usuario con el sistema mediante el intercambio de información, capturando y desplegando los datos necesarios para realizar alguna tarea. En esta capa los datos se procesan de manera superficial por ejemplo, para determinar la validez de su formato o para darles algún orden específico.*
- *Esta capa se comunica únicamente con la capa de Reglas de Negocio o Control.*

Capa de Lógica de Negocio

- *Es aquí donde se encuentra toda la lógica del programa, así como las estructuras de datos y objetos encargados para la manipulación de los datos existentes, así como el procesamiento de la información ingresada o solicitada por el usuario en la capa de presentación.*

- *Representa el corazón de la aplicación ya que se comunica con todas las demás capas para poder llevar a cabo las tareas. Por ejemplo, mediante la capa de presentación obtiene la información ingresada por el usuario, y despliega los resultados. Si la aplicación se comunica con otros sistemas que actúan en conjunto, lo hace mediante esta capa. También se comunica con la capa de datos para obtener información existente o ingresar nuevos datos.*

Capa de Datos

- *Es la encargada de realizar transacciones con bases de datos y con otros sistemas para obtener o ingresar información al sistema.*
- *Es en esta capa donde se definen las consultas a realizar en la base de datos, tanto las consultas simples como las consultas complejas para la generación de reportes más específicos.*
- *Esta capa envía la información directamente a la capa de reglas de negocio para que sea procesada e ingresada en objetos según se necesite.*

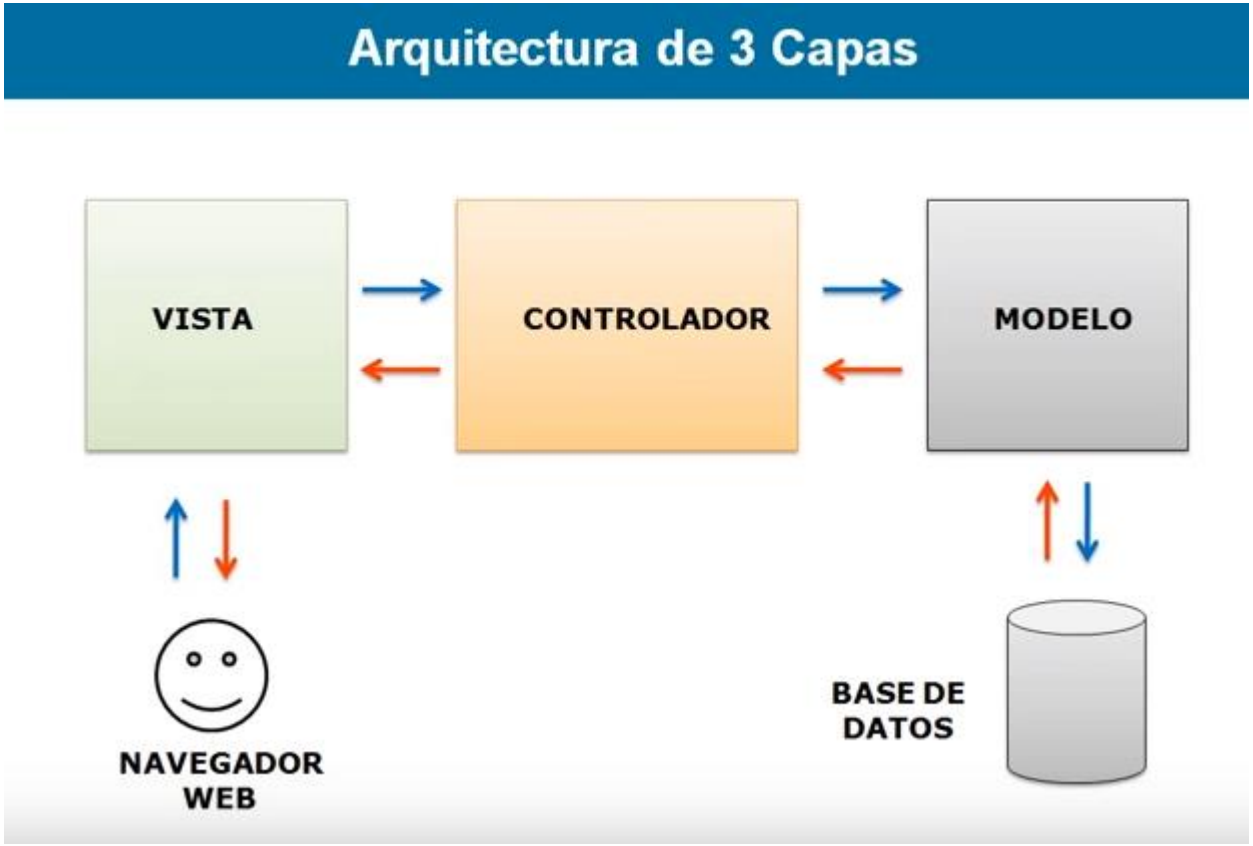
Ventajas

- *Al implementar este modelo de programación, se asegura un trabajo de forma ordenada y separada, debido a que sigue el principio de "divide y vencerás".*
- *Cada capa está dividida según su funcionalidad cuando se quiere modificar el sistema basta con cambiar un objeto o conjunto de objetos de una capa. Esto se llama modularidad.*

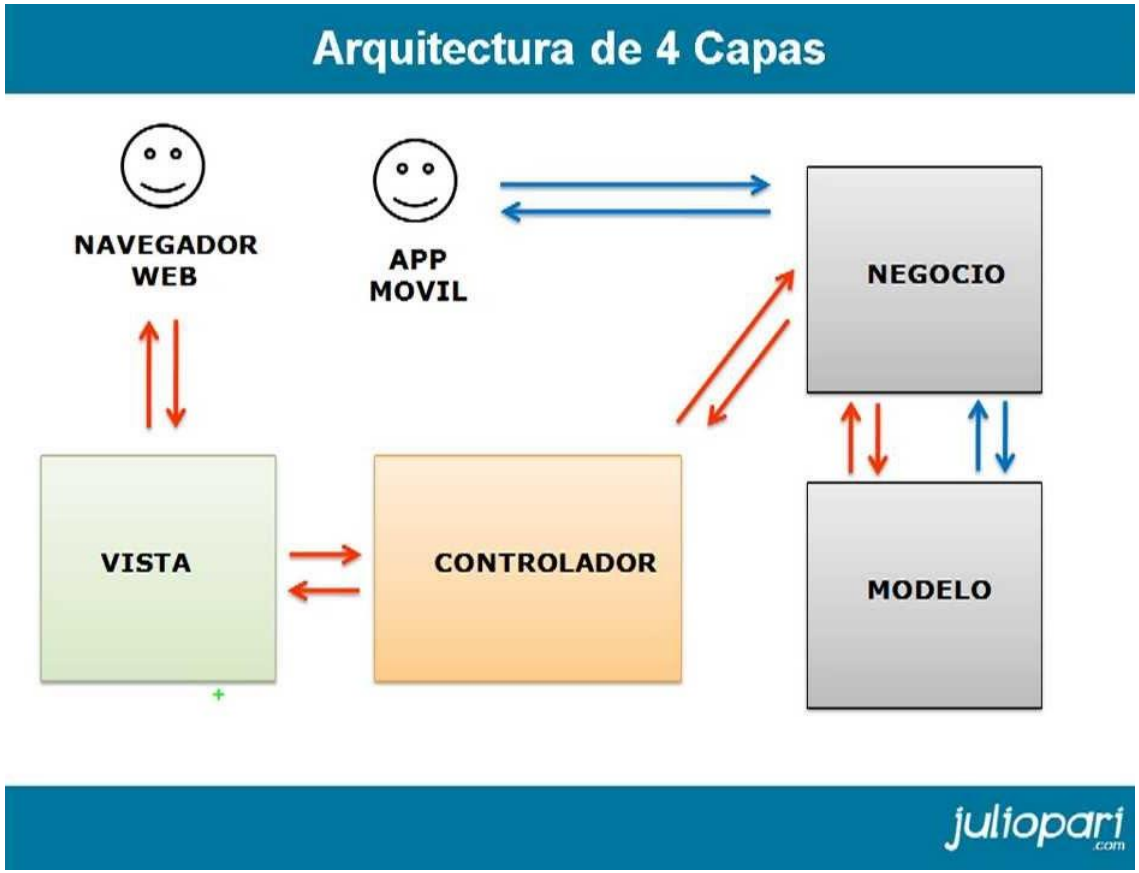
Desventajas

- *Cuando se implementa un modelo de programación en capas, se debe llegar a un balance entre el número de capas y subcapas que componen el programa. Este debe ser el necesario y suficiente para realizar un trabajo específico con eficiencia y ser lo más modular posible.*
- *De lo contrario se tiene una serie de desventajas como: pérdida de eficiencia, realización de trabajo innecesario o redundante entre capas, gasto de espacio de la aplicación debido a la expansión de las capas, o bien una alta dependencia entre los objetos y capas que contradice el objetivo principal del modelo.*

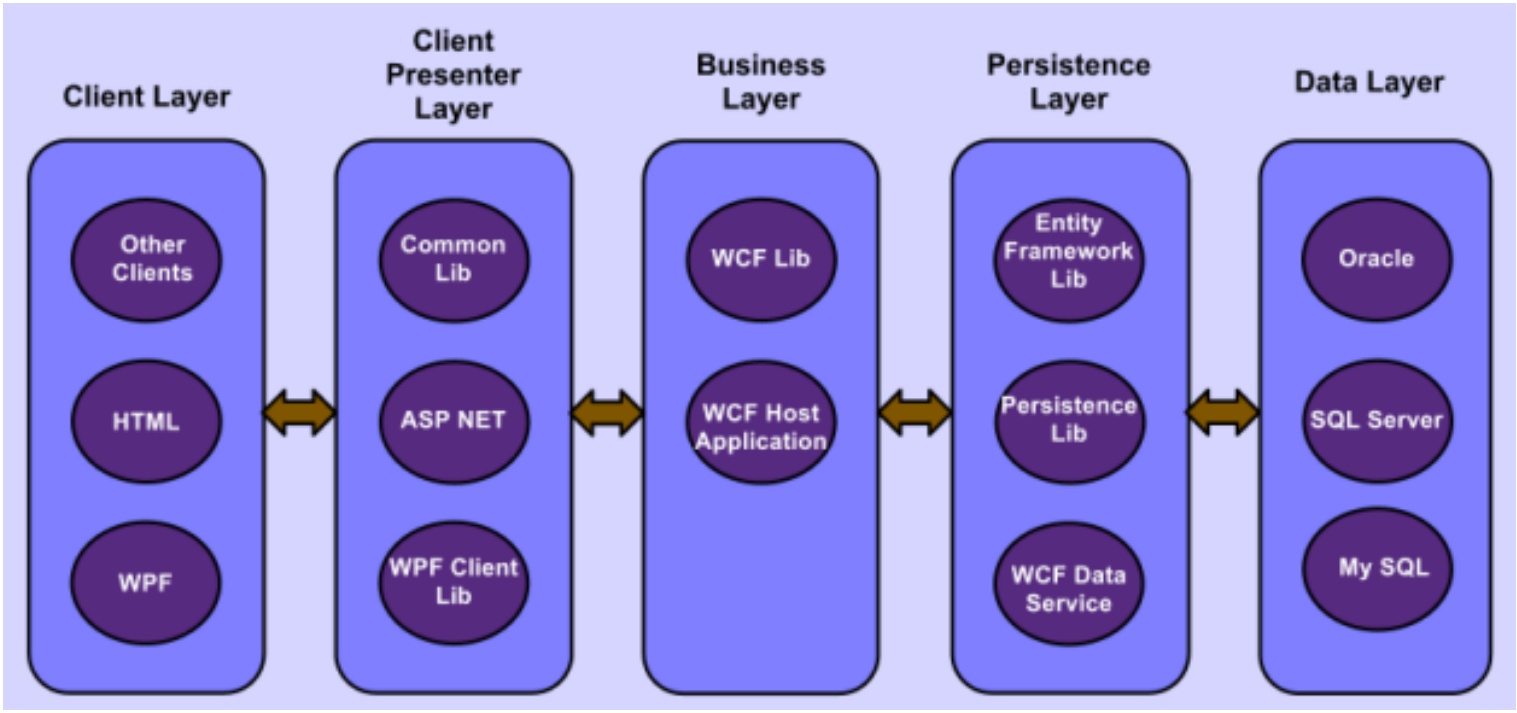
N-Capas (N-Layer)



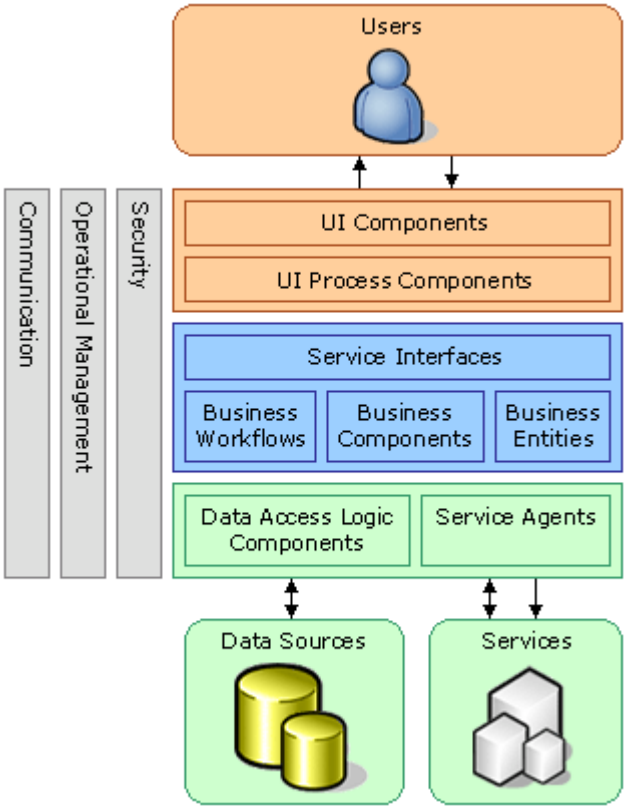
N-Capas (N-Layer)

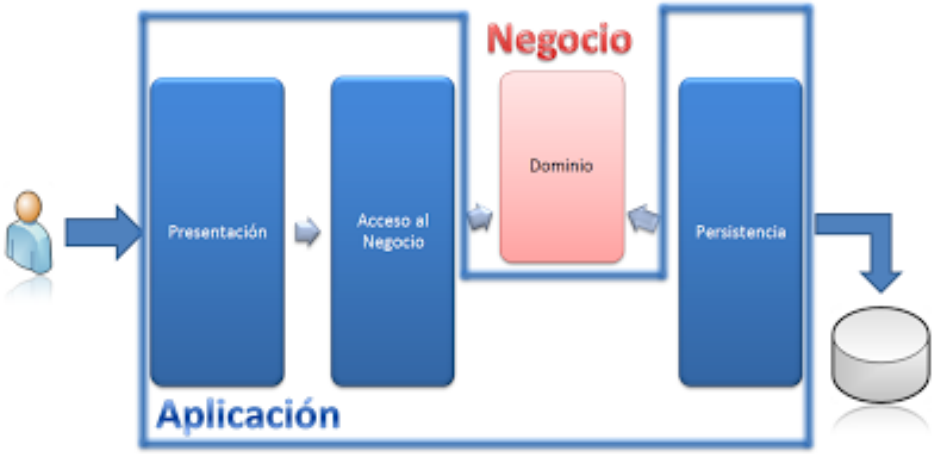


N-Capas (N-Layer)



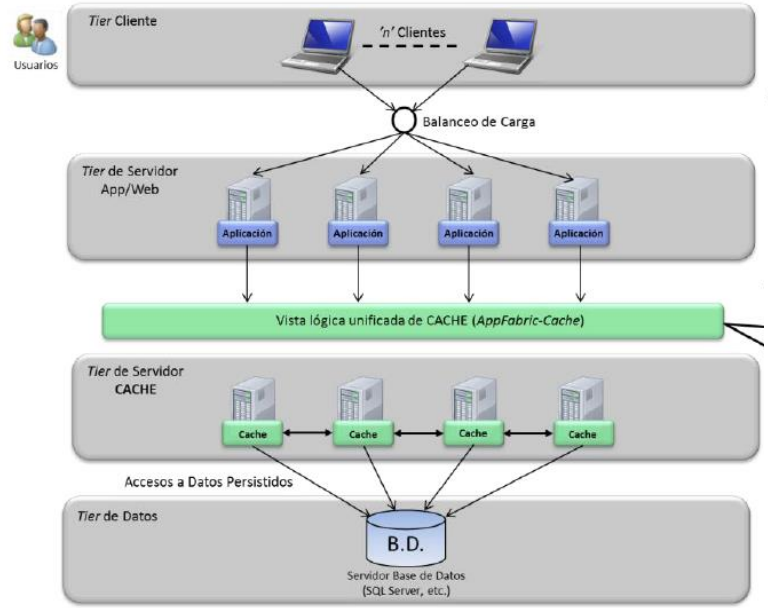
N-Capas (N-Layer)



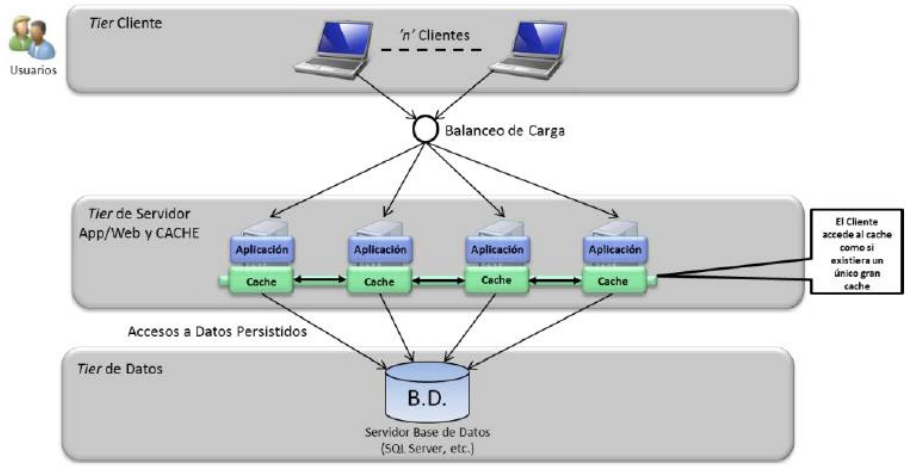


N-Niveles(N-Tier)

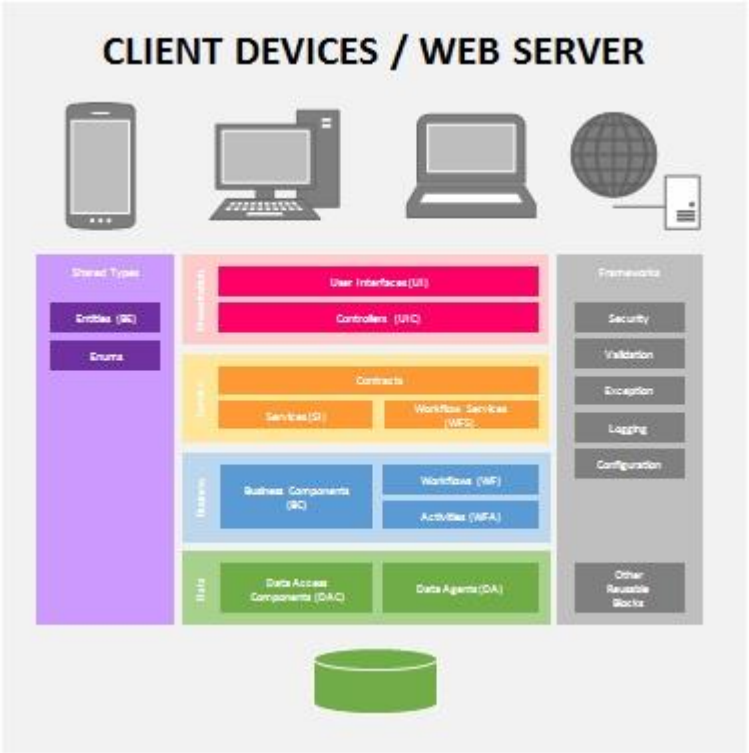
Arquitectura '*N-Tier*' con '*Web-Farm*' balanceado y Cache distribuido

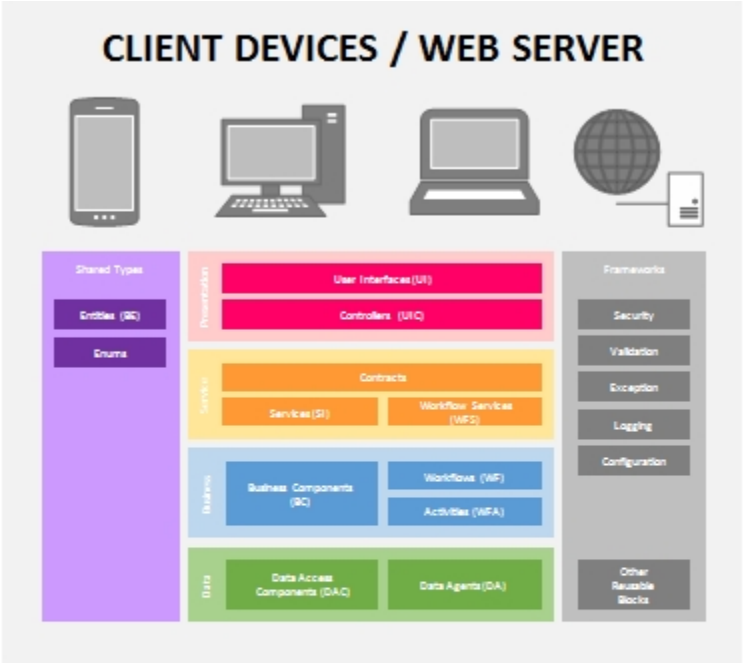


Arquitectura '*N-Tier*' con '*Web-Farm*' balanceado y Cache distribuido



N-Niveles(N-Tier)





N-Niveles(N-Tier)