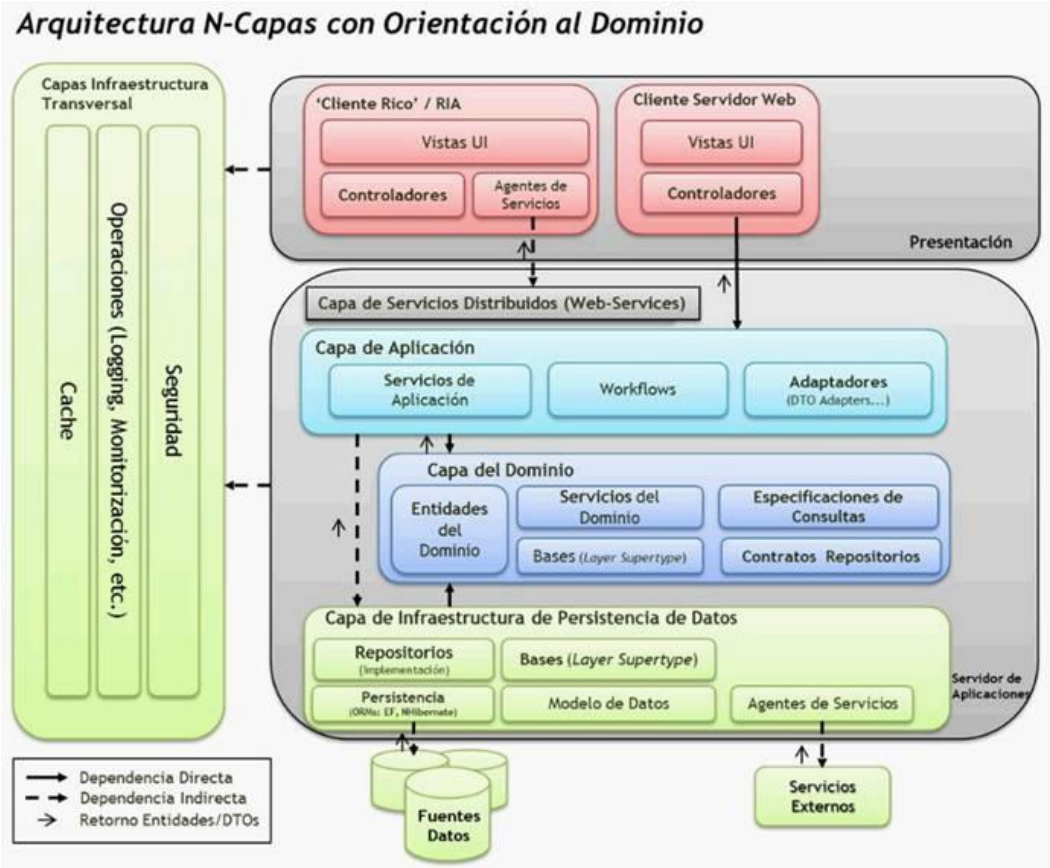
Contenido

1. Orientación a tendencias de Arquitectura DDD (Domain Driven Design)

El objetivo de esta arquitectura marco es proporcionar una base consolidada y guías de arquitectura para un tipo concreto de aplicaciones: “Aplicaciones empresariales complejas”. Este tipo de aplicaciones se caracterizan por tener una vida relativamente larga y un volumen de cambios evolutivos considerable. Por lo tanto, en estas aplicaciones es muy importante todo lo relativo al mantenimiento de la aplicación, la facilidad de actualización, o la sustitución de tecnologías y frameworks por otras versiones más modernas o incluso por otros diferentes. El objetivo es que todo esto se pueda realizar con el menor impacto posible sobre el resto de la aplicación.

2. Arquitectura Marco N-Capas con Orientación al Dominio

El objetivo de esta arquitectura marco es estructurar de una forma limpia y clara la complejidad de una aplicación empresarial basada en las diferentes capas de la arquitectura, siguiendo el patrón N-Layered y las tendencias de arquitecturas en DDD. El patrón N-Layered distingue diferentes capas y sub-capas internas en una aplicación, delimitando la situación de los diferentes componentes por su tipología. Por supuesto, esta arquitectura concreta N-Layer es personalizable según las necesidades de cada proyecto y/o preferencias de Arquitectura. Simplemente proponemos una Arquitectura marco a seguir que sirva como punto base a ser modificada o adaptada por arquitectos según sus necesidades y requisitos. En concreto, las capas y sub-capas propuestas para aplicaciones "N-Layered con Orientación al Dominio" son:



- **Capa de Presentación**
  - Subcapas de Componentes Visuales (Vistas)

- Subcapas de Proceso de Interfaz de Usuario (Controladores y similares)

- **Capa de Servicios Distribuidos (Servicios-Web)**
  - Servicios-Web publicando las Capas de Aplicación y Dominio
- **Capa de Aplicación**
  - Servicios de Aplicación (Tareas y coordinadores de casos de uso)
  - Adaptadores (Conversores de formatos, etc.)
- **Capa del Modelo de Dominio**
  - Entidades del Dominio
  - Servicios del Dominio
  - Contratos/Interfaces de *Repositorios*
- **Capa de Infraestructura de Acceso a Datos**
  - Implementación de Repositorios"
  - Modelo lógico de Datos
  - Infraestructura tecnología ORM
  - Agentes de Servicios externos
- **Componentes/Aspectos Horizontales de la Arquitectura**
  - Aspectos horizontales de Seguridad, Gestión de operaciones, Monitorización, Correo Electrónico automatizado, etc.

2.1. Capa de Infraestructura de Persistencia de Datos

Esta Capa de Persistencia, siguiendo las tendencias de Arquitectura DDD, forma realmente parte de las Capas de Infraestructura tal y como se define en la Arquitectura DDD propuesta por Eric-Evans, puesto que estará finalmente ligada a ciertas tecnologías específicas (de acceso a datos, en este caso). Sin embargo, debido a la importancia que tiene el acceso a datos en una aplicación y al cierto paralelismo y relación con la Capa de Dominio.

Los componentes de persistencia de datos proporcionan acceso a datos que están hospedados dentro de las fronteras de nuestro sistema (p.e. nuestra base de datos principal), pero también datos expuestos fuera de las fronteras de nuestro sistema, como Servicios Web de sistemas externos. Contiene, por lo tanto, componentes de tipo "Repositorio" que proporcionan la funcionalidad de acceder a datos hospedados dentro de las fronteras de nuestro sistema o bien "Agentes de Servicio" que consumirán Servicios Web que expongan otros

sistemas *back-end* externos. Adicionalmente, esta capa dispondrá normalmente de componentes/clases base con código reutilizable por todas las clases “repositorio”.

Elementos de la Capa de Persistencia y Acceso a Datos

La Capa de Persistencia de datos suele incluir diferentes tipos de componentes. A continuación explicamos brevemente las responsabilidades de cada tipo de elemento propuesto para esta capa:

1. **Repositorios**
   Estos componentes son muy similares en algunos aspectos a los componentes de “Acceso a Datos” (DAL) de Arquitecturas tradicionales N-*Layered*. Básicamente, los Repositorios son clases/componentes que encapsulan la lógica requerida para acceder a las fuentes de datos de la aplicación. Centralizan por lo tanto funcionalidad común de acceso a datos de forma que la aplicación pueda disponer de un mejor mantenimiento y desacoplamiento entre la tecnología y la lógica de las capas “Aplicación” y “Dominio”. Si se hace uso de tecnologías base tipo O/RM (*Object/Relational Mapping frameworks*), se simplifica mucho el código a implementar y el desarrollo se puede focalizar exclusivamente en los accesos a datos y no tanto en la tecnología de acceso a datos (conexiones a bases de datos, sentencias SQL, etc.) que se hace mucho más transparente en un O/RM. Por el contrario, si se utilizan componentes de acceso a datos de más bajo nivel, normalmente es necesario disponer de clases utilidad propias que sean reutilizables para tareas similares de acceso a datos.

2. **Modelo de Datos**
   Este concepto existe a veces en la implementación de la Capa de Persistencia para poder definir e, incluso visualizar gráficamente el modelo de datos “entidad-relación” de la aplicación. Este concepto suele ser proporcionado completamente por la tecnología O/RM concreta que se utilice, por lo que está completamente ligado a una infraestructura/tecnología específica (p.e. Entity Framework proporciona una forma de definir un modelo entidad-relación o incluso de crearlo a partir de una base de datos existente).

3. **Tecnología de Persistencia (O/RM, etc.)**
   Es simplemente la capa de infraestructura/tecnología utilizada internamente por nuestros Repositorios. Normalmente será, por lo tanto, la propia tecnología que hayamos escogido, bien un O/RM como Entity Framework o NHibernate, o simplemente tecnología de más bajo nivel como ADO.NET.

4. **Agentes de Servicios Distribuidos externos**
   Cuando un componente del dominio debe acceder a datos proporcionados por un servicio distribuido externo (p.e. un Servicio-Web), debemos implementar código que gestione la semántica de comunicación con dicho servicio en particular. Estos Agentes de Servicio implementan precisamente componentes de acceso a datos que encapsulan y aíslan los requerimientos de los Servicios distribuidos e incluso pueden soportar aspectos adicionales como cache, soporte off-line y mapeos básicos entre el formato de datos expuesto en los Servicios distribuidos externos y el formato de datos requerido/utilizado por nuestra aplicación.

2.2. Capa de Modelo de Dominio

Esta capa debe ser responsable de representar conceptos de negocio, información sobre la situación de los procesos de negocio e implementación de las reglas del dominio. También debe contener los estados que reflejan la situación de los procesos de negocio, aun cuando los detalles técnicos de almacenamiento se delegan a las capas inferiores de infraestructura (Repositorios, etc.)
Esta capa, "Modelo del Dominio", es el corazón del software.
Así pues, estos componentes implementan la funcionalidad principal del sistema y encapsulan toda la lógica de negocio relevante (genéricamente llamado lógica del Dominio según nomenclatura DDD). Básicamente suelen ser clases en el lenguaje seleccionado que implementan la lógica del dominio dentro de sus métodos, aunque también puede ser de naturaleza diferente, como sistemas dinámicos de reglas de negocio, etc.
Siguiendo los patrones de Arquitectura en DDD, esta capa tiene que ignorar completamente los detalles de persistencia de datos. Estas tareas de persistencia deben ser realizadas por las capas de infraestructura.
La principal razón de implementar capas de lógica del dominio (negocio) radica en diferenciar y separar muy claramente entre el comportamiento de las reglas del dominio (reglas de negocio que son responsabilidad del modelo del dominio) de los detalles de implementación de infraestructura (acceso a datos y repositorios concretos ligados a una tecnología específica como pueden ser ORMs, o simplemente librerías de acceso a datos o incluso de

aspectos horizontales de la **arquitectura)**. De esta forma (aislando el Dominio de la aplicación) incrementaremos drásticamente la mantenibilidad de nuestro sistema y podríamos llegar a sustituir las capas inferiores (acceso a datos, ORMs, y bases de datos) sin que el resto de la aplicación se vea afectada.

1. **Entidades del Dominio**

   Las ENTIDADES representan objetos del dominio y están definidas fundamentalmente por su identidad y continuidad en el tiempo de dicha identidad y no solamente por los atributos que la componen. Las entidades normalmente tienen una correspondencia directa con los objetos principales de negocio/dominio, como cliente, empleado, pedido, etc. Así pues, lo más normal es que dichas entidades se persistan en bases de datos, pero esto depende completamente del dominio y de la aplicación. No es una obligación. Pero precisamente el aspecto de "continuidad" tiene que ver mucho con el almacenamiento en bases de datos. La continuidad significa que una entidad tiene que poder "sobrevivir" a los ciclos de ejecución de la aplicación. Si bien, cada vez que la aplicación se re-arranca, tiene que ser posible reconstituir en memoria/ejecución estas entidades.

2. **Contratos/Interfaces de Repositorios dentro de la Capa de Dominio**

   La implementación de los Repositorios no es parte del Dominio sino parte de las capas de Infraestructura (puesto que los Repositorios están ligados a una tecnología de persistencia de datos, como ORMs tipo Entity Framework), sin embargo, el "contrato" de cómo deben ser dichos Repositorios (Interfaces a cumplir por dichos Repositorios), eso sí debe formar parte del Dominio. Por eso lo incluimos aquí. Esto es así porque dicho contrato especifica qué debe ofrecer el Repositorio, sin importarme como está implementado por dentro. Dichos interfaces sí son agnósticos a la tecnología. Así pues, los interfaces de los Repositorios es importante que estén definidos dentro de las Capas del Dominio. Este punto es algo precisamente recomendado en las arquitecturas DDD y está basado en el patrón "Separated Interface Pattern" definido por Martin Fowler. Lógicamente, para poder cumplir este punto, es necesario que las "Entidades del Dominio" y los "Value-Objects" sean POCO/IPOCO, es decir, también completamente agnósticos a la tecnología de acceso a datos. Hay que tener en cuenta que las entidades del dominio son, al final, los "tipos de

datos” de los parámetros enviados y devueltos por y hacia los Repositorios.
En definitiva, con este diseño (Persistence Ignorance) lo que buscamos es que las clases del dominio “no sepan nada directamente” de los repositorios. Cuando se trabaja en las capas del dominio, se debe ignorar como están implementados los repositorios.

3. SERVICIOS del Modelo de Dominio

En la mayoría de los casos, nuestros diseños incluyen operaciones que no pertenecen conceptualmente a objetos de ENTIDADES del Dominio. En estos casos podemos incluir/agrupar dichas operaciones en SERVICIOS explícitos del Modelo del Dominio.
Nota:

Es importante destacar que el concepto SERVICIO en capas N-Layer DDD no es el de SERVICIO-DISTRIBUIDO (Servicios Web normalmente) para accesos remotos. Es posible que un Servicio-Web “envuelva” y publique para accesos remotos a la implementación de Servicios del Dominio, pero también es posible que una aplicación web disponga de servicios del dominio y no disponga de Servicio Web alguno.

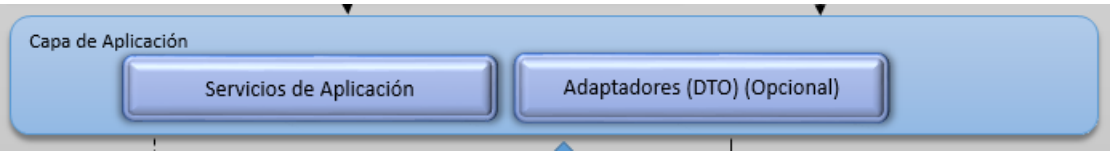
Dichas operaciones que no pertenecen específicamente a ENTIDADES del Dominio, son intrínsecamente actividades u operaciones, no características internas de entidades del Dominio. Pero debido a que nuestro modelo de programación es orientado a objetos, debemos agruparlos también en objetos. A estos objetos les llamamos SERVICIOS.

El forzar a dichas operaciones del Dominio (en muchos casos son operaciones de alto nivel y agrupadoras de otras acciones) a formar parte de objetos ENTIDAD distorsionaría la definición del modelo del dominio y haría aparecer ENTIDADES artificiales.

Un SERVICIO es una operación o conjunto de operaciones ofrecidas como un interfaz que simplemente está disponible en el modelo.

La palabra “Servicio” del patrón SERVICIO precisamente hace hincapié en lo que ofrece: “Qué puede hacer y qué acciones ofrece al cliente que lo consuma y enfatiza la relación con otros objetos del Dominio (Englobando varias Entidades, en muchos casos)”.

2.3. Capa de Aplicación



Esta Capa de Aplicación, siguiendo las tendencias de Arquitectura DDD, debe ser una Capa delgada que coordina actividades de la Aplicación como tal, pero es fundamental que no incluya lógica de negocio ni tampoco por lo tanto estados de negocio/dominio. Si puede contener estados de progreso de tareas de la aplicación.

Los SERVICIOS que viven típicamente en esta capa (recordar que el patrón SERVICIO es aplicable a diferentes capas de la Arquitectura), son servicios que normalmente coordinan SERVICIOS de otras capas de nivel inferior.

El caso más normal de un Servicio de Aplicación es un Servicio que coordine toda la "fontanería" de nuestra aplicación, es decir, orquestación de llamadas a Servicios del Dominio y posteriormente llamadas a Repositorios para realizar la persistencia, junto con creación y uso de UoW, transacciones, etc.

1. Servicios de Aplicación

El SERVICIO de Aplicación es otro tipo más de Servicio, cumpliendo con las directrices de su patrón (SERVICE pattern). Básicamente deben ser objetos sin estados que coordinen ciertas operaciones, en este caso operaciones y tareas relativas a la Capa de Aplicación (Tareas requeridas por el software/aplicación, no por la lógica del Dominio).

Otra función de los Servicios de Aplicación es encapsular y aislar a la capa de infraestructura de persistencia de datos. Así pues, en la capa de aplicación realizaremos la coordinación de transacciones y persistencia de datos (solo la coordinación o llamada a los Repositorios), validaciones de datos y aspectos de seguridad como requerimientos de autenticación y autorización para ejecutar componentes concretos, etc.

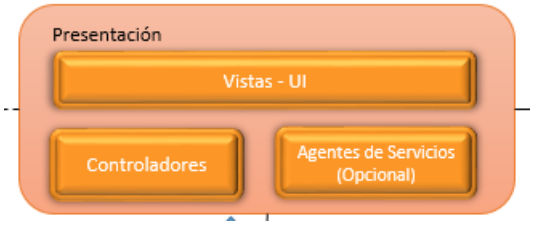
También los SERVICIOS deben ser normalmente el único punto o tipo de componente de la arquitectura por el que se acceda a las clases de infraestructura de persistencia de datos (Repositorios) desde capas superiores. No debería, por ejemplo, 216 Guía de Arquitectura N-Capas Orientada al Dominio con .NET 4.0 poder invocar directamente desde la Capa de Presentación (p.e. Web) objetos Repositorios de la Capa de Persistencia y Acceso a Datos.

El aspecto fundamental de esta capa es no mezclar requerimientos de Software (coordinación de la persistencia, conversiones a diferentes formatos de datos, optimizaciones, Calidad de Servicio, etc.) con la Capa de Dominio que solo debe contener pura lógica de Negocio.

2.4. Capa de Servicios Distribuidos

- La Capa de Servicios normalmente incluye lo siguiente: Interfaces/Contratos de Servicios: Los Servicios exponen un interfaz de servicio al que se envían los mensajes de entrada. En definitiva, los servicios son como una fachada que expone la lógica de aplicación y del dominio a los consumidores potenciales, bien sea la Capa de Presentación o bien sean otros Servicios/Aplicaciones remotas.
- Mensaje de Tipos: Para intercambiar datos a través de la capa de Servicios, es necesario hacerlo mediante mensajes que envuelven a estructuras de datos. La capa de servicio también incluirá tipos de datos y contratos que definan los tipos de datos utilizados en los mensajes.

2.5. Capa de Presentación

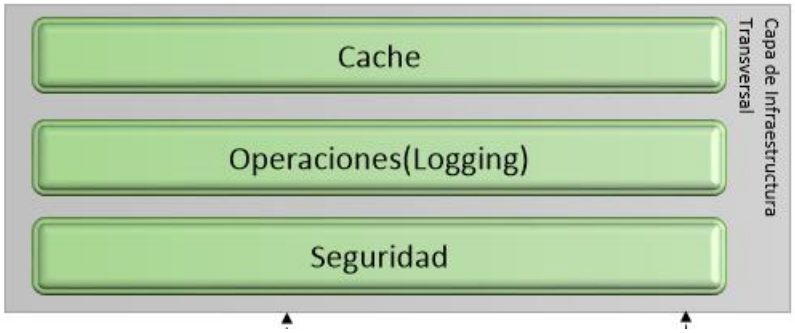


La responsabilidades de esta capa son las de presentar al usuario los conceptos de negocio mediante una interfaz de usuario (IU), facilitar la explotación de dichos procesos, informar sobre el estado de los mismos e implementar las reglas de validación de dicha interfaz. Al fin y al cabo, desde el punto de vista del usuario final, esta capa es la “Aplicación” en sí, y de nada sirve el haber planteado la mejor arquitectura del mundo si no podemos facilitar la explotación de ella de la manera más satisfactoria posible para el usuario.

Una de las peculiaridades de las interfaces de usuario es que necesitan de unas habilidades que están fuera del ámbito del desarrollador, como pueden ser las habilidades de diseño artístico, los conocimientos de accesibilidad y de usabilidad, y el control de la localización de las aplicaciones. Por tanto, lo más recomendable es que un profesional de este ámbito como puede ser un diseñador gráfico, trabaje junto al

desarrollador para lograr un resultado de alta calidad. Es responsabilidad de esta capa el facilitar esta colaboración entre ambos roles. El desarrollador programará en el lenguaje orientado a objetos elegido creando la lógica de la capa de presentación, y el diseñador usará otras herramientas y tecnologas como puede ser HTML o XAML para crear la parte visual y de interacción entre otras cosas.

2.6. Capa de Infraestructura Transversal



La mayoría de aplicaciones contienen funcionalidad común que se utiliza en los las diferentes Capas tradicionales e incluso en diferentes Niveles físicos (Tiers). Este tipo de funcionalidad normalmente abarca operaciones como autenticación, autorización, cache, gestión de excepciones, logging/registros, trazas, instrumentalización y validación. A este tipo de funcionalidad normalmente se le denomina “Aspectos Transversales” o “Aspectos Horizontales”, porque afectan a la aplicación entera, y deben estar por lo tanto centralizados en una localización central si es posible, para favorecer la reutilización de componentes entre las diferentes capas. Por ejemplo, el código que genera trazas en los ficheros de log de una aplicación, probablemente se utilice desde muchos puntos diferentes de la aplicación (desde diferentes capas y niveles físicos). Si centralizamos el código encargado de realizar las tareas específicas de generar trazas (u otra acción), seremos capaces en el futuro de cambiar el comportamiento de dicho aspecto cambiando el código solamente en un área concreta.