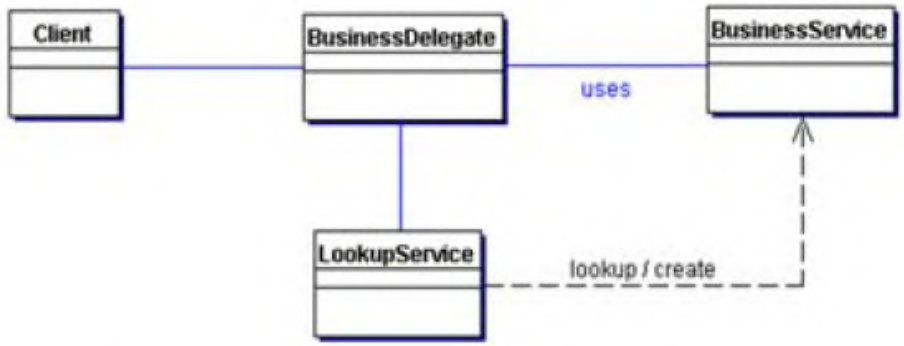
6-Patrón delegado del negocio

- Propósito
  - Evita que los clientes tengan que tratar con detalles de acceso a componentes distribuidos en una aplicación multicapa
- También conocido como:
  - *Business delegate*

6-Patrón delegado del negocio

- Motivación
  - Cuando se implementa la capa de negocio con componentes distribuidos los clientes de dicha capa (p.e. la capa de presentación) tienen que tratar con detalles de conexión y acceso al servidor de aplicaciones
  - El patrón delegado se encarga de estos detalles, permitiendo que los clientes se abstraigan de la implementación del negocio

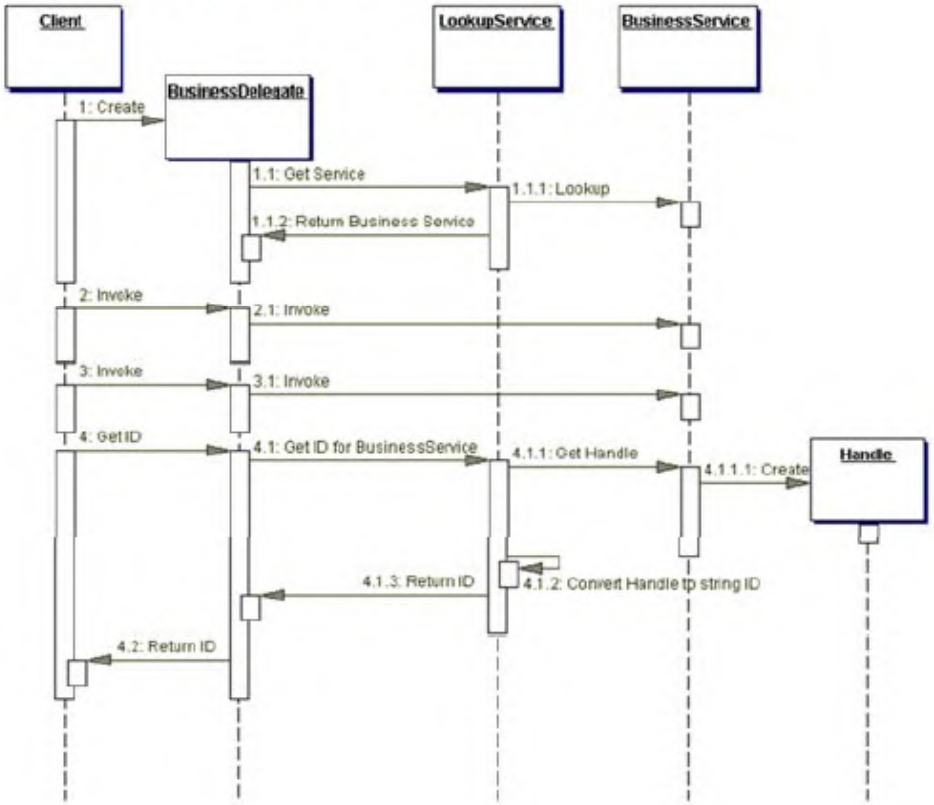
6-Patrón delegado del negocio



Estructura del patrón delegado

6-P. delegado del negocio

Interacción entre objetos relacionados por el patrón delegado



6-Patrón delegado del negocio

- Consecuencias
  - Ventajas
    - Oculta detalles
    - Independiza capas
  - Inconvenientes
    - Introduce otro nivel más de indirección

7-Patrón servicio de aplicacion

- Propósito
  - Centraliza lógica del negocio
- También conocido como:
  - *Application service*

7-Patrón servicio de aplicacion

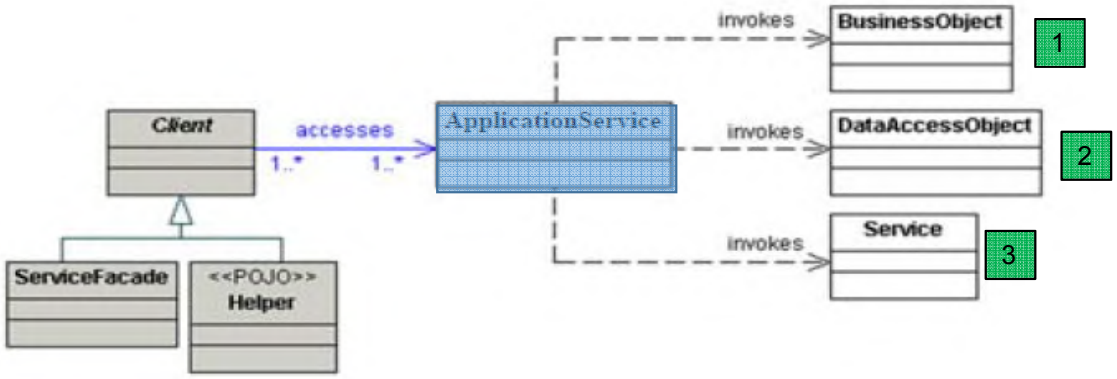
- Motivación
  - En una arquitectura multicapa, la lógica del negocio debe estar en algún sitio.
  - Dejarla en los clientes, vulneraría una estructura multicapa.
  - Incluirla en la fachada corrompería su naturaleza
  - Por eso la incluimos en servicios de la aplicación.

7-Patrón servicio de aplicacion

- Debe aplicarse cuando
  - Se quiera representar una lógica del negocio que actúe sobre distintos servicios u *objetos del negocio*.
  - Se quiera agrupar funcionalidades relacionadas.
  - Se quiera encapsular lógica no representada por objetos del negocio.

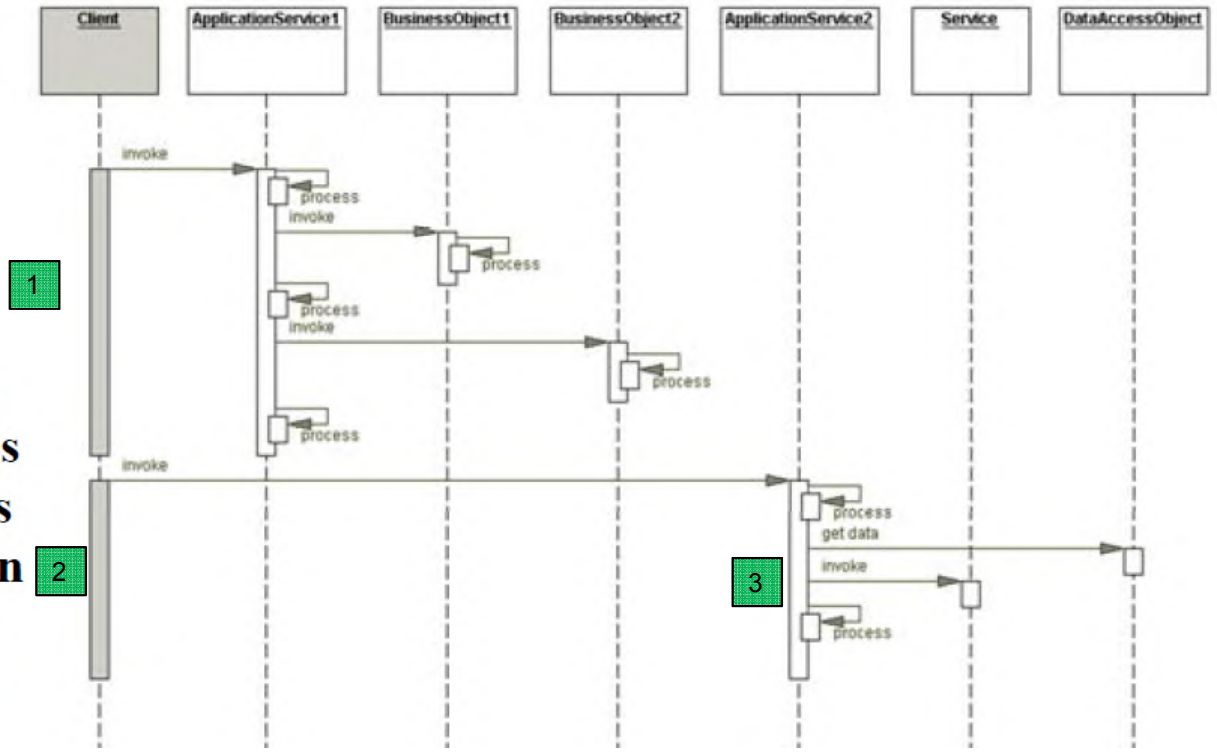
7-Patrón servicio de aplicacion

- Estructura



7-Patrón servicio de aplicacion

Interacción entre objetos relacionados por el patrón servicio de aplicación



7-Patrón servicio de aplicacion

- Consecuencias
  - Ventajas
    - Centraliza lógica del negocio.
    - Mejora la reusabilidad del código.
    - Evita duplicación de código.
    - Simplifica la implementación de fachadas
  - Inconvenientes
    - Introduce un nivel más de indirección

7- Patrón servicio de aplicacion

- Nota
  - Los servicios de aplicación no suelen tener atributos para hacerlos *más ligeros*
  - Entonces, ¿dónde están los objetos que tienen atributos y operaciones del negocio?
  - Estos objetos son los *objetos del negocio*

8- Patrón objeto del negocio

- Propósito
  - Representar la lógica del negocio y el modelo del dominio en términos orientados a objetos
- También conocido como:
  - *Business object*

8-Patrón objeto del negocio

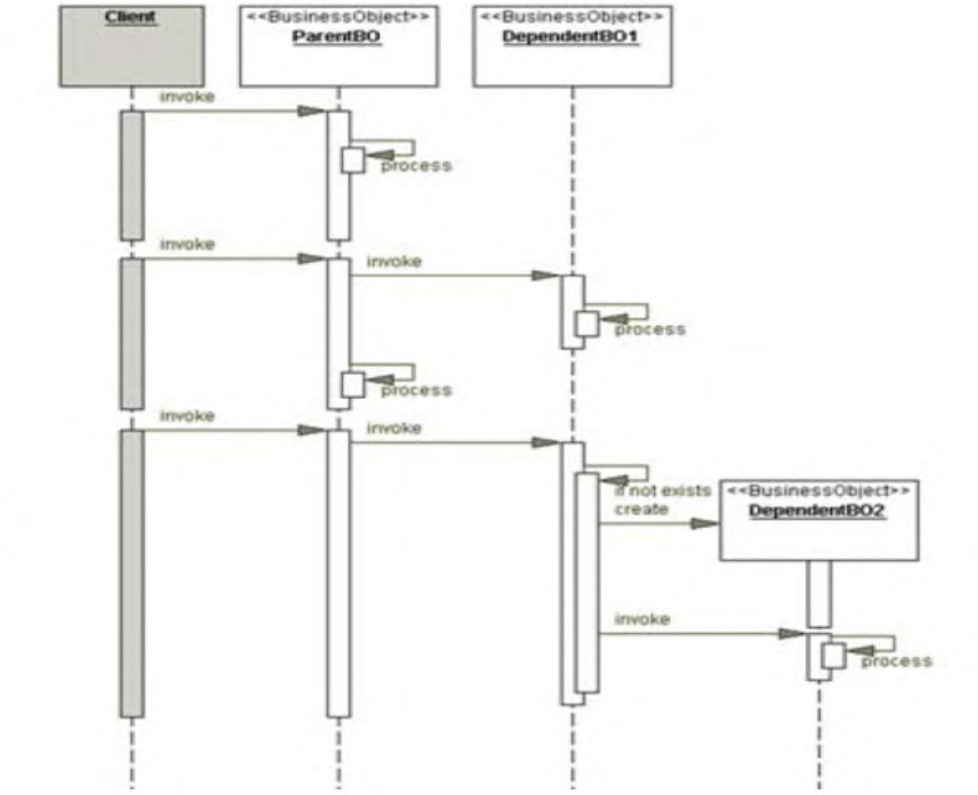
- Motivación
  - Cuando la lógica del negocio es poca o inexistente, las aplicaciones pueden permitir a los clientes acceder directamente a la capa de datos.
  - Así, un componente de la capa de negocio (ej. `ServiciosUsuarioImp`) podría acceder directamente a un DAO.

8-Patrón objeto del negocio

– Sin embargo, si en el cliente hay una gran cantidad de procesos computacionales asociados a los datos, dichos procesos deberían encapsularse en un objeto que representase un objeto del negocio

8-Patrón objeto del negocio

Interacción entre objetos relacionados por el patrón objeto del negocio



8-Patrón objeto del negocio

- Consecuencias
  - Ventajas
    - Promueve una aproximación orientada a objetos en la implementación del modelo del negocio.
    - Centraliza el comportamiento del negocio, promoviendo la reusabilidad
    - Evita la duplicación de código

8-Patrón objeto del negocio

- Inconvenientes
  - Añade una capa de indirección.
  - Puede producir objetos "inflados" de funcionalidad
  - Persistencia de dichos objetos del negocio.

9-Patrón almacén del dominio

- Propósito
  - Se desea separar la persistencia del modelo de objetos
- También conocido como:
  - *Domain store*
  - *Unit of work + Query object + Data mapper + Table data gateway + Dependent mapping + Domain model + Data transfer object + Identity map + Lazy load*

- Mejor sería relacionar los objetos del negocio con su persistencia a través de un *almacén del dominio*

En el caso de **persistencia de objetos** la información que persiste **son los valores que contienen los atributos en un momento determinado**, no necesariamente la funcionalidad que proveen sus métodos.

9-Patrón almacén del dominio

- Motivación
  - Muchas veces se dispone de un modelo de objetos del negocio muy rico.
  - El simple uso de *transfers* no es suficiente.
  - Se desea dar persistencia al modelo de objetos de forma independiente de éste

9- Patrón almacén del dominio

- Debe aplicarse cuando
  - Se deseen omitir detalles de persistencia en los objetos del negocio.
  - No se desee utilizar EJBs de entidad. (* Enterprise Java Bean´s)
  - La aplicación podría ejecutarse en un contenedor web.
  - El modelo de objetos utiliza herencia y relaciones complejas

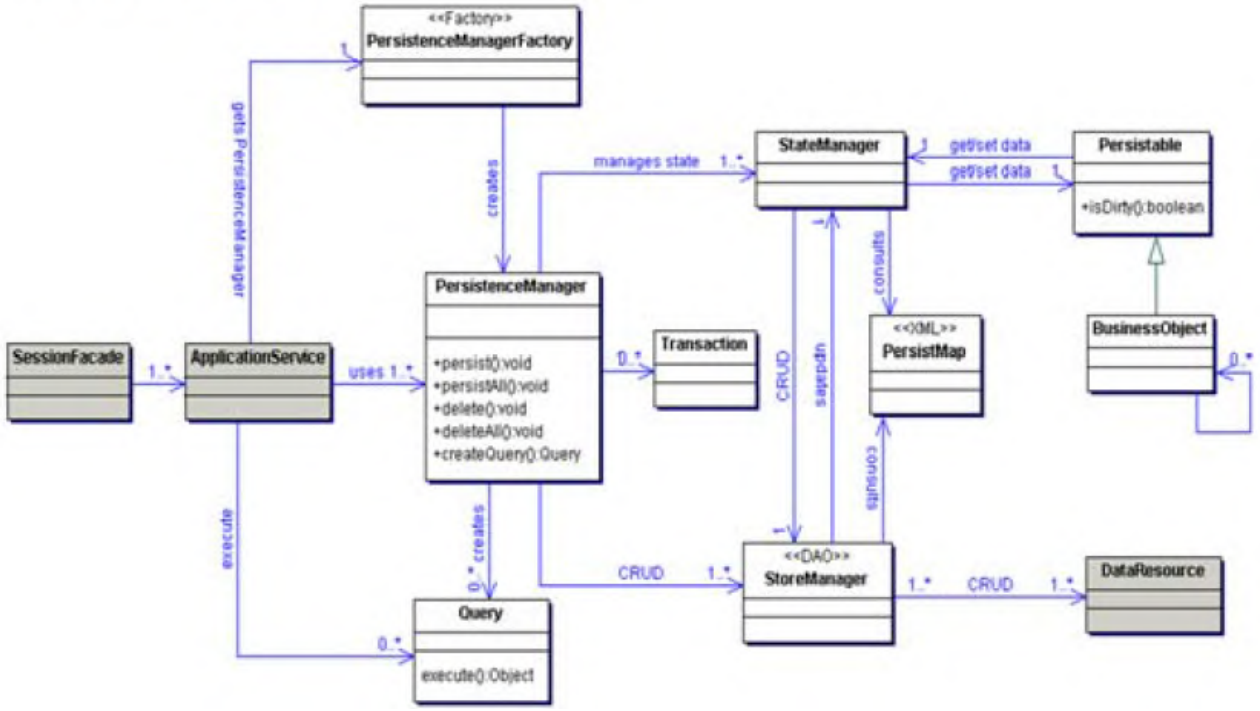
*** EJBs:** son una de las interfaces de programación de aplicaciones (API) que forman parte del estándar de construcción de aplicaciones empresariales J2EE. Su especificación detalla cómo los servidores de aplicaciones proveen objetos desde el lado del servidor.

9- Patrón almacén del dominio

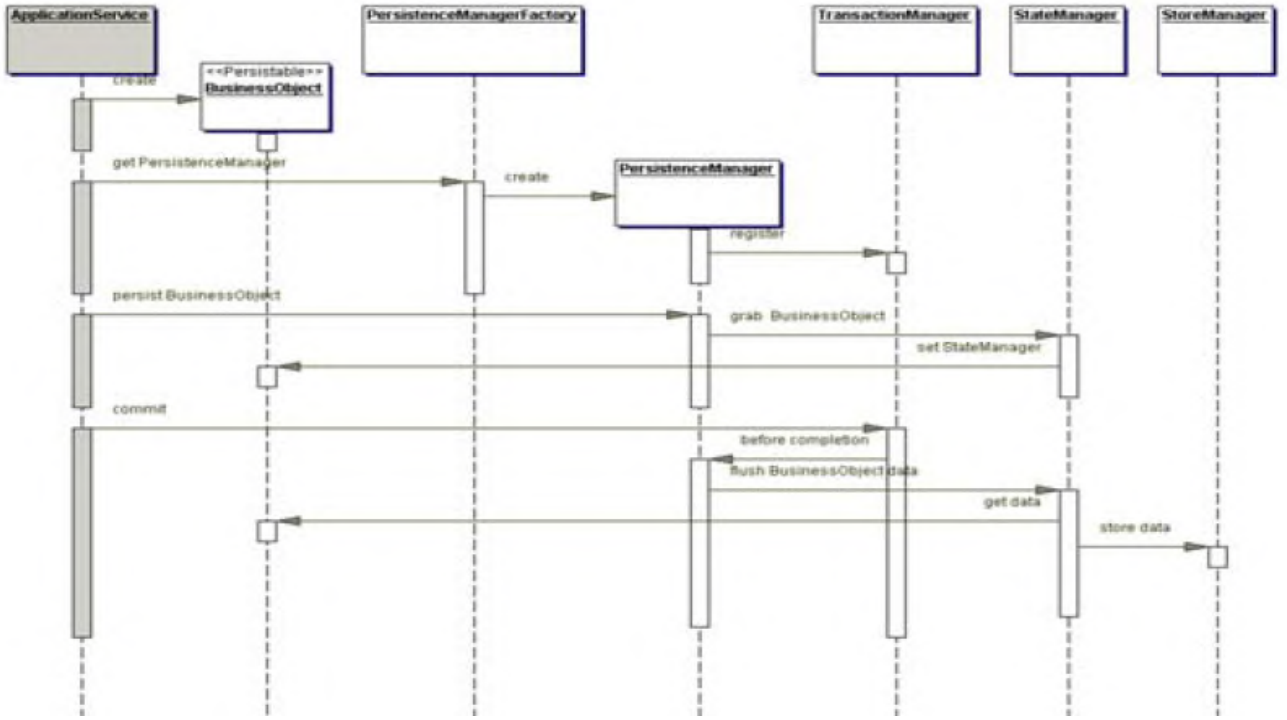
– La implementación del almacén del dominio puede hacerse:
  • Construyéndolo.
  • Utilizando algún framework de persistencia.

9-Patrón almacén del dominio

- Estructura:

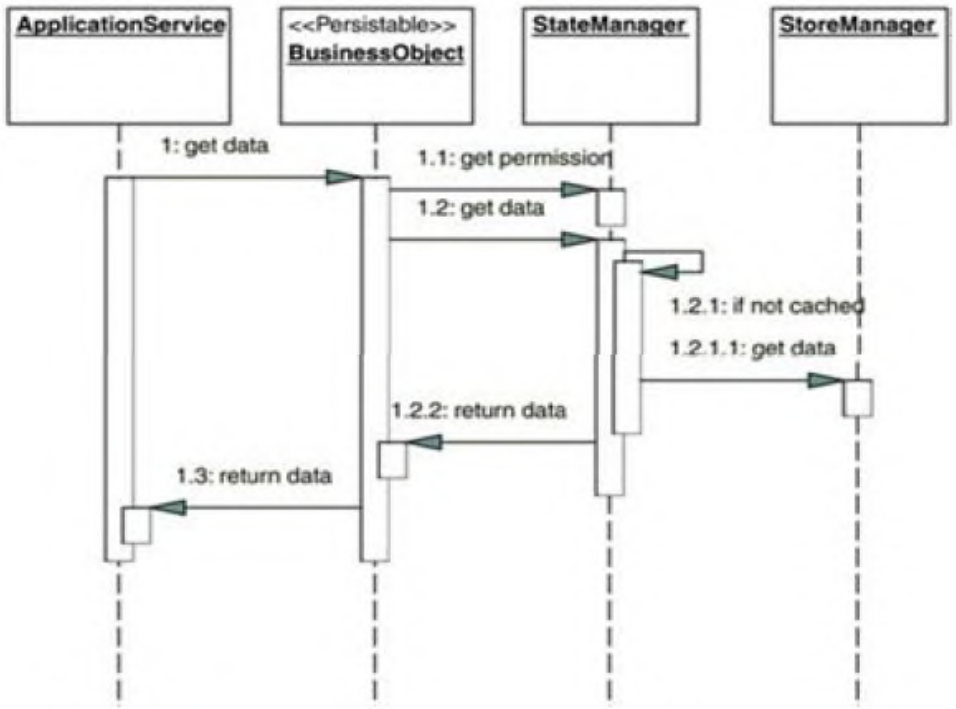


9-Patrón almacén del dominio



Persistencia de un objeto del negocio

9-Patrón almacén del dominio



Recuperación de atributos de un objeto del negocio

9-Patrón almacén del dominio

- Consecuencias
  - Ventajas:
    - Independiza el modelo de objetos de su persistencia.
    - Es capaz de construir el grafo de dependencias entre objetos bajo demanda.
    - Control de transacciones.
  - Inconvenientes:
    - Número de objetos.
    - Complejidad.