Patrones de diseño

Sesión 3:

Patrones de diseño J2EE para arquitecturas distribuidas con EJBs

Índice

- **Catálogo de patrones. Diferencias con aplicaciones locales**
- **Patrones para la capa de integración**
- **Patrones para la capa de negocio**

Diferencias con aplicaciones locales

- El principal “peso” a optimizar son las llamadas remotas
  - Utilizar aplicaciones con EJBs locales
  - Si ello no es posible, intentar aislar las llamadas remotas y minimizar su número
- En realidad los patrones que se utilizan son más o menos los mismos (solo que ahora se hace énfasis en que los patrones permiten disminuir las llamadas remotas u ocultar su complejidad)

Catálogo de patrones

- Capa de integración
  - Fast lane Reader
- Capa de negocio
  - Session Façade
  - Data Transfer Object
  - Composite entity (no recomendado por muchos)
  - Service Locator
  - Business Delegate
- Capa de presentación
  - Exactamente lo mismo que en aplicaciones locales (lógico, ¿no?)

Índice

- Catálogo de patrones. Diferencias con aplicaciones locales
- Patrones para la capa de integración
- Patrones para la capa de negocio

Capa de integración con EJBs

- Los EJBs de entidad con CMP nos permiten olvidarnos de la persistencia (hasta cierto punto)

- Las relaciones manejadas por el contenedor (CMR) requieren interfaces locales…hay que apañárselas para acceder desde clientes remotos (ya lo visteis en el módulo EJB)

Fast Lane Reader

- Problema: los Entity Beans son objetos “pesados”
- Las operaciones de solo lectura de gran cantidad de datos son muy frecuentes
  - Buscar de usuarios, productos,
  - Navegar por las categorías de un catálogo
  - Leer los mensajes de un foro
- ¿Para qué queremos EJBs de entidad con CMP en este caso?
  - No hay necesidad de transacciones
- Solución: utilizar un DAO (acceso a datos con JDBC o similar) para ciertas operaciones de solo lectura

Índice

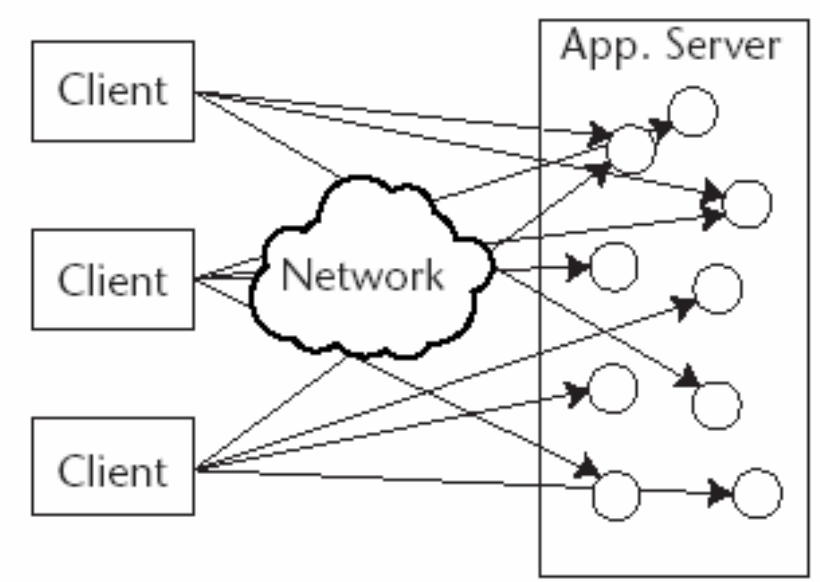
- Catálogo de patrones. Diferencias con aplicaciones locales
- Patrones para la capa de integración
- Patrones para la capa de negocio

Session Façade

- Este patrón ya aparecía en aplicaciones locales con el nombre de “service façade”.
- En aplicaciones locales es la fachada de la capa de negocio (el API de la capa de negocio)
- En aplicaciones remotas su principal justificación adicional es la reducción de llamadas remotas.
- Es el patrón más importante y conocido en aplicaciones remotas

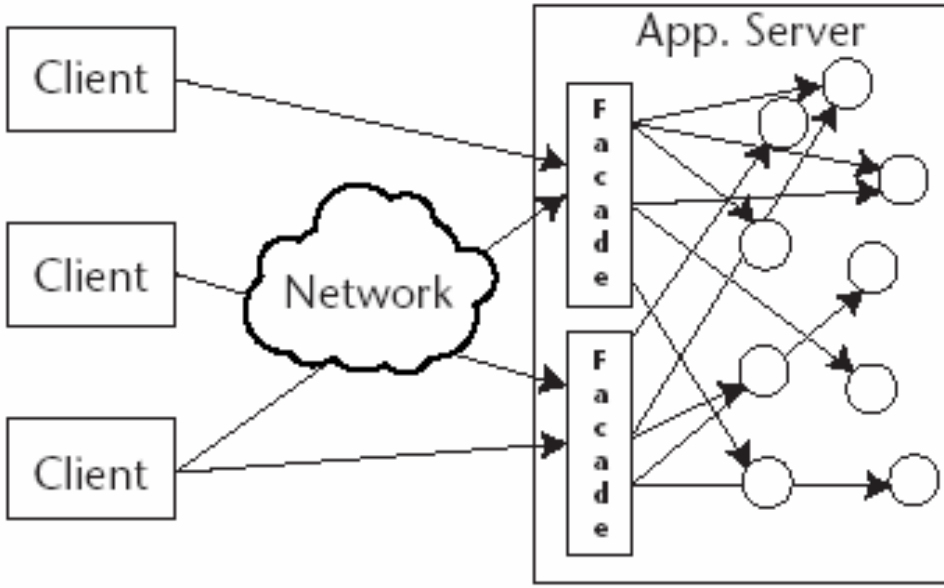
Sin Session Façade…

- **Operación: envío de mensaje. Implicados**
  - **Remitente**
  - **Mensaje**
  - **Destinatario**
- **Se multiplica el número de llamadas remotas**
- **¿Qué objeto debería implementar el método de envío?**
- **Problemas de integridad transaccional: cada EJB de entidad tiene su transacción. Hay que gestionar la transacción a mano.**



Con Session Façade…

- **EJB de sesión encargado de realizar la operación en el contenedor EJB por delegación del cliente.**
- **Mejoras**
  - **Se hace una única llamada remota**
  - **La lógica común a varios objetos la implementa el propio Façade**
  - **Las transacciones las gestiona el propio Façade (para eso es un EJB)**



Posibles problemas si se utiliza mal

- **Clases demasiado grandes**
  - **Solución: agrupar casos de uso similares y adjudicar cada grupo a un Session Façade distinto**
- **Implementación de código propio de los objetos fuera de ellos**
  - **Solución: cuidado en distinguir lógica común de particular**
- **Duplicación de código**
  - **Solución: planificación cuidadosa o refactorización del código periódicamente para extraer la lógica común a varios casos de uso**

Data Transfer Object

- Paquetes de datos para transferir información
- En su forma más simple son JavaBeans
- En aplicaciones locales su justificación es organizativa
  - 1 parámetro en lugar de N
  - El DTO encapsula los cambios
- En aplicaciones distribuidas su justificación principal es reducir el número de llamadas remotas
  - 1 llamada `getUsuarioTO()` en lugar de `getNombre()`, `getDireccion()`, ...

Data Transfer Object (II)

- En aplicaciones remotas hay dos diferencias en cuanto al uso
  - Los DTOs se utilizan para transferir datos entre la capa de negocio y la cliente, pero en otros sitios no son necesarios
    - En EJB 1.1. TODAS las llamadas a EJBs eran remotas. Dentro de la capa EJB salía a cuenta usar DTOs
    - En EJB 2.X se pueden usar llamadas locales entre el Session Façade y los otros EJBs. No hay diferencia considerable de velocidad, se puede acceder con "grano fino"
  - Los DTOs deben ser serializables, ya que van a viajar por la red (`implements serializable`)

Composite Entity

- Necesitamos un modelo de objetos del dominio (Usuario, Tarjeta, Cuenta,…)
- En EJB 1.X el acceso era SIEMPRE remoto. Modelar todos los objetos como EJBs de entidad suponía un coste no tolerable
- Solución:
  - Los objetos independientes (ej. Usuario) serán EJBs de entidad
  - Los dependientes (ej. Tarjeta) serán POJOs

Composite Entity (II)

- Con EJB 2.0 el problema de eficiencia se desvanece en gran medida: Todo el modelo puede estar hecho en base a EJBs (¡¡locales, por supuesto!!!!)
- No obstante, Core J2EE Patterns sigue hablando de este patrón extensamente (como si nada)
- Razón para su uso:
  - complejidad de desarrollo con EJBs
  - Que todo sea un EJB de entidad puede llevar a dependencia del modelo de datos (siguiendo la costumbre típica de EJB=tabla)
- Razón para no usarlo: si se emplea, no podemos aprovecharnos de la persistencia manejada por el contenedor

Patrones prácticamente iguales

- **Service Locator: si se usan EJBs el Locator pasa a responsabilizarse de localizarlos y cachear su interfaz Home**

- **Business Delegate: sigue ocultando la implementación de la capa de presentación (que debería ser ignorante de que esto es una aplicación distribuida con EJBs)**
  - **Particularidad del Business Delegate: físicamente es local al cliente, pero desde el punto de vista lógico es de la capa de negocio (se ocuparían de él los desarrolladores de esta capa)**