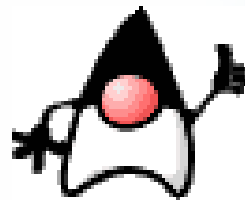




PATRONES

Felipe Conejera C.
Rodrigo Loyola A.





PATRONES

🚀 Introducción a Patrones.

🚀 Qué es un Patrón??.

SOLUCIÓN *recurrente a un*
PROBLEMA, *en un*
CONTEXTO *determinado.*





PATRONES

- **Contexto:**
 - Entorno, situación o condición donde es necesario aplicar.....

- **Problema:**
 - Insatisfacción. Necesario investigar y resolver

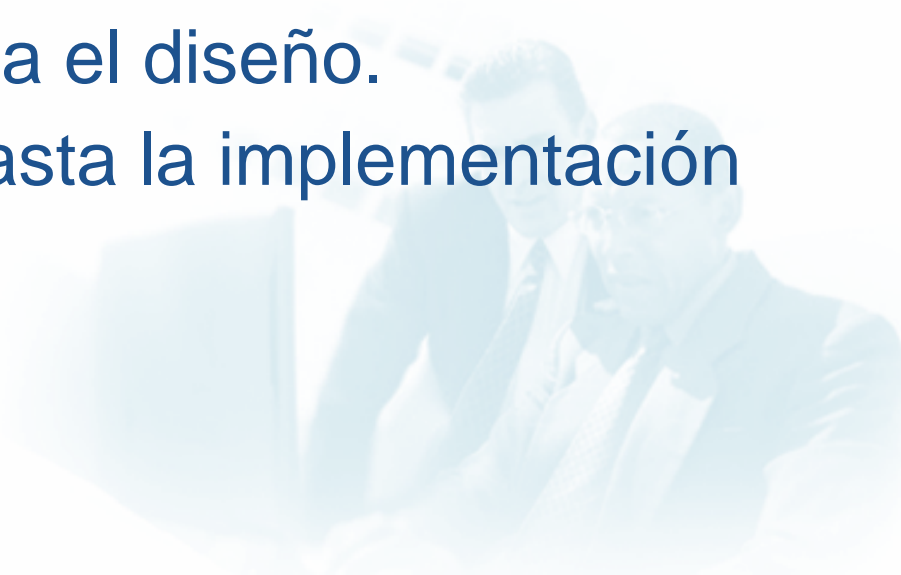
- **Solución:**
 - Respuesta al problema, detectado en dicho contexto.





PATRONES




- Solucionan problemas, existentes en distintos niveles de abstracción.
- Existen patrones para distintas etapas del desarrollo.
 - Desde el análisis, hasta el diseño.
 - Desde Arquitectura, hasta la implementación





PATRONES

Categorías para los Patrones:

-  **Creacionales:** Forma de crear instancias de objetos.
-  **Estructurales:** Forma de combinar las clases y objetos, para formar una estructura mayor.
-  **Comportamiento:** Definen la comunicación e interacción entre los objetos de un sistema.





PATRONES

- **Planilla de Patrón:**
 - Contexto.
 - Problema.
 - Solución.
 - Estructura.
 - Estrategia.
 - Consecuencias
 - Patrones Relacionados.





ARQUITECTURA J2EE

- **5 Capas:**
 - Cliente.
 - Presentación.
 - Negocio.
 - Integración.
 - Recursos.





CAPA PRESENTACIÓN

- Intercepting Filtre.
- Front Controller.
- View Helper.
- Composite View.
- Service to Worker.
- Dispatcher View.





CAPA PRESENTACIÓN

INTERCEPTING FILTER





INTERCEPTING FILTER

- Un objeto que está entre el cliente y los componentes Web. Este procesa las peticiones y las respuestas.





INTERCEPTING FILTER

■ Contexto.

- Manejo de peticiones a la capa de presentación.
- Algunas peticiones simplemente requieren reenvío, mientras otras deben ser modificadas, auditadas o descomprimidas antes de su procesamiento.





INTERCEPTING FILTER

■ Problema:

- Se requiere un **pre** y **post** procesamiento, para las peticiones o respuesta de un cliente web.
- Test de entrada:
 - Se ha autorizado al cliente?.
 - Tiene el cliente una sesión válida?.
 - La dirección IP del cliente, es conocida?.
 - Viola alguna restricción el Path de la petición?.
- Mejor forma, método simple para agregar o quitar componentes de procesamiento.



INTERCEPTING FILTER

- **Causas.**

- Procesamiento Común.
- Se desea la Centralización de la Lógica común.
- Se debe facilitar la adición o eliminación de servicios, sin afectar a los componentes existentes.





INTERCEPTING FILTER

■ Solución.

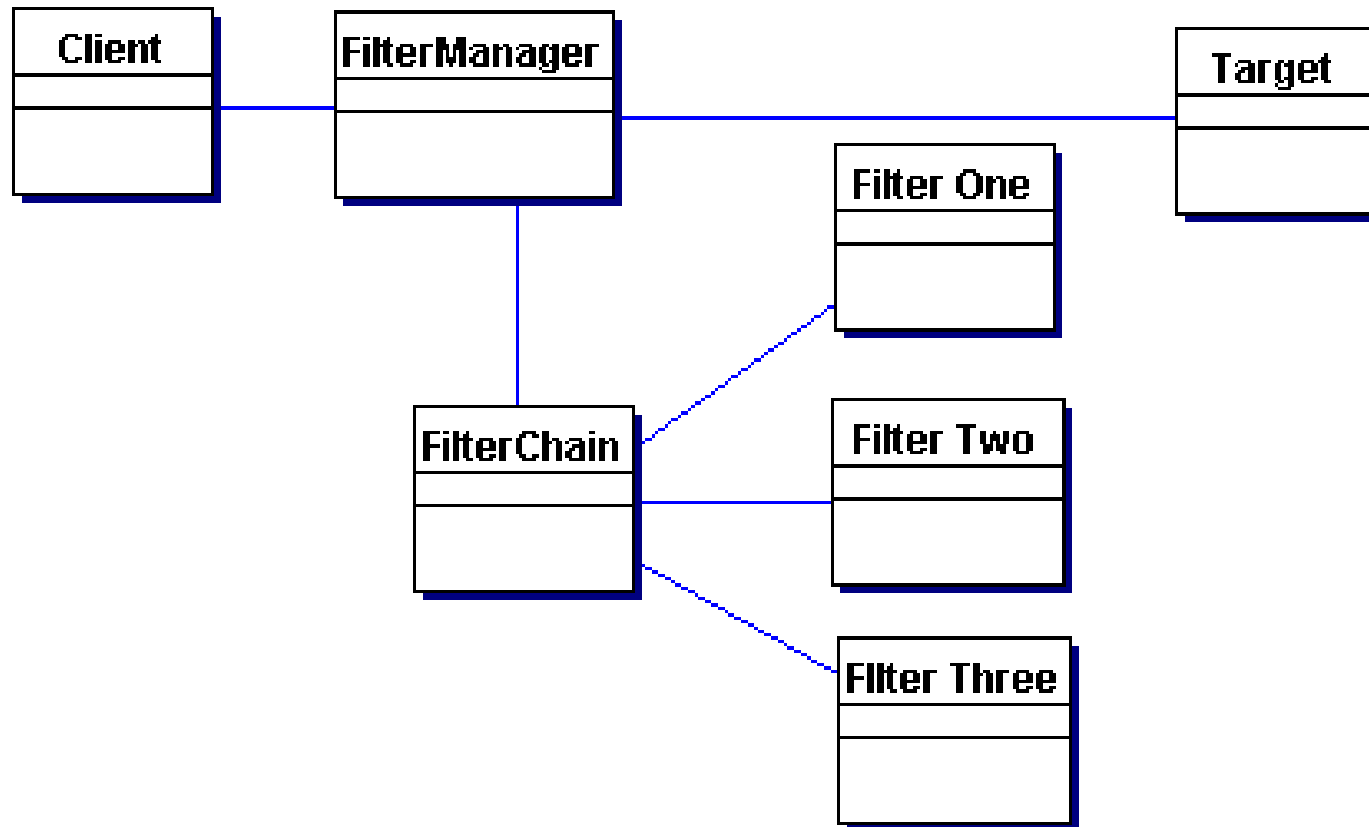
- Se crean **FILTROS conéctables**, para procesar servicios comunes de una forma estándar.
- Los filtros interceptan las peticiones entrantes y las respuestas salientes, permitiendo el pre y post-procesamiento.
- Añadir o eliminar filtros, sin la necesidad de modificar nuestro código.





INTERCEPTING FILTER

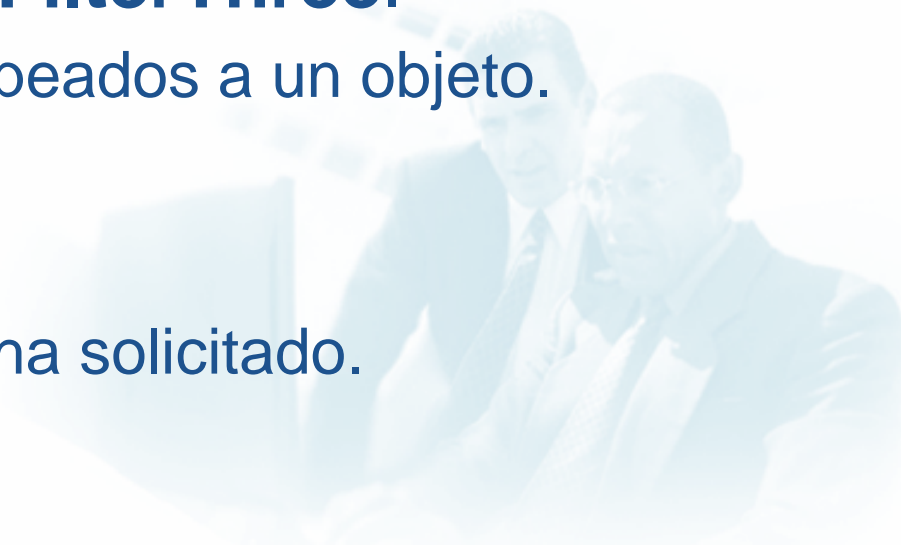
■ Estructura.





INTERCEPTING FILTER

- **Participantes y Responsabilidad.**
 - **FilterManager.**
 - Maneja el procesamiento de filtros.
 - **FilterChain.**
 - **FilterOne, FilterTwo, FilterThree.**
 - Filtros individuales, mapeados a un objeto.
 - **Target (Objetivo).**
 - Recurso que el cliente ha solicitado.





INTERCEPTING FILTER

- **Estrategias.**

- **Custom Filter.**

- Implementada según el programador. Poco flexible.

- **Estándar Filter.**

- Presente en Servlet 2.3. Preprocesa peticiones de cualquier tipo.

- **Base Filter.**

- Es una base para todos los demás filtros.

- **Template Filter.**





INTERCEPTING FILTER

■ Consecuencias.

- Centraliza el control.
- Mejora la reutilización.
- Configuración Declarativa y Flexible.
- La información no es bien compartida.





FRONT CONTROLLER





FRONT CONTROLLER

- Es un objeto que acepta todos los requerimientos de un cliente y los direcciona a manejadores apropiados.
- Podríamos dividir la funcionalidad en 2 objetos:
 - **El Front Controller:** Acepta todos los requerimientos de un cliente y realiza la autenticación.
 - **Dispatcher:** Direcciona los requerimientos al manejador apropiado.



FRONT CONTROLLER

■ Contexto.

- Peticiones de todos los usuarios, controladas a través de mecanismos de varias peticiones.
- Dichos mecanismos de control se pueden elaborar de una manera centralizada o distribuida (descentralizada).





FRONT CONTROLLER

■ Problema.

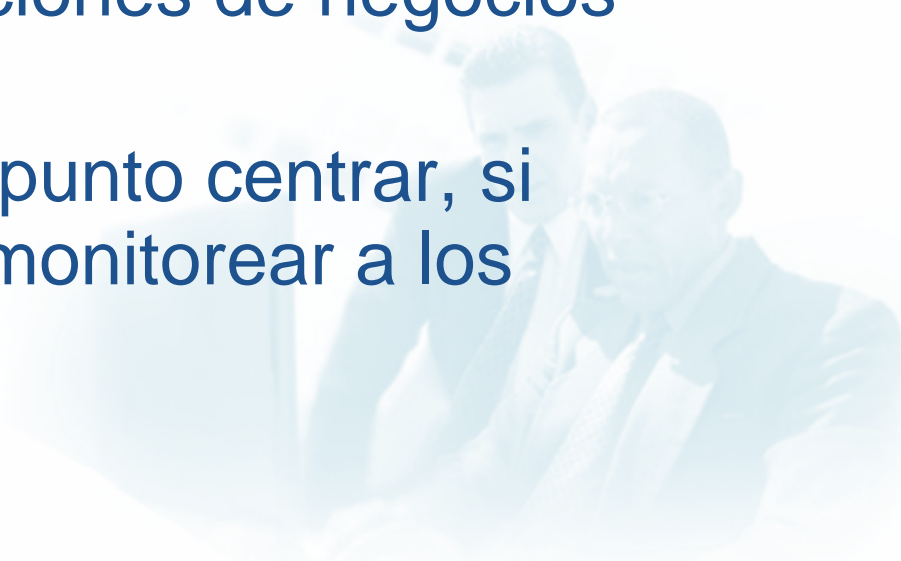
- Se **requiere** un sistema de acceso centralizado, si no:
 - Se requeriría que cada vista proporcione sus propios servicios de sistema. Implica duplicación de código.
 - La vista, se deja a los visores, lo cual generaría una mezcla entre Contenidos y Navegación.
- Accesos distribuidos, difíciles de mantener.



FRONT CONTROLLER

■ Causas.

- La lógica se maneja mejor si está centralizada.
- Puntos de decisión en la recuperación y manipulación de datos.
- Vistas distintas, a peticiones de negocios similares.
- Puede ser muy útil un punto central, si queremos controlar y monitorear a los usuarios.





FRONT CONTROLLER

■ Solución.

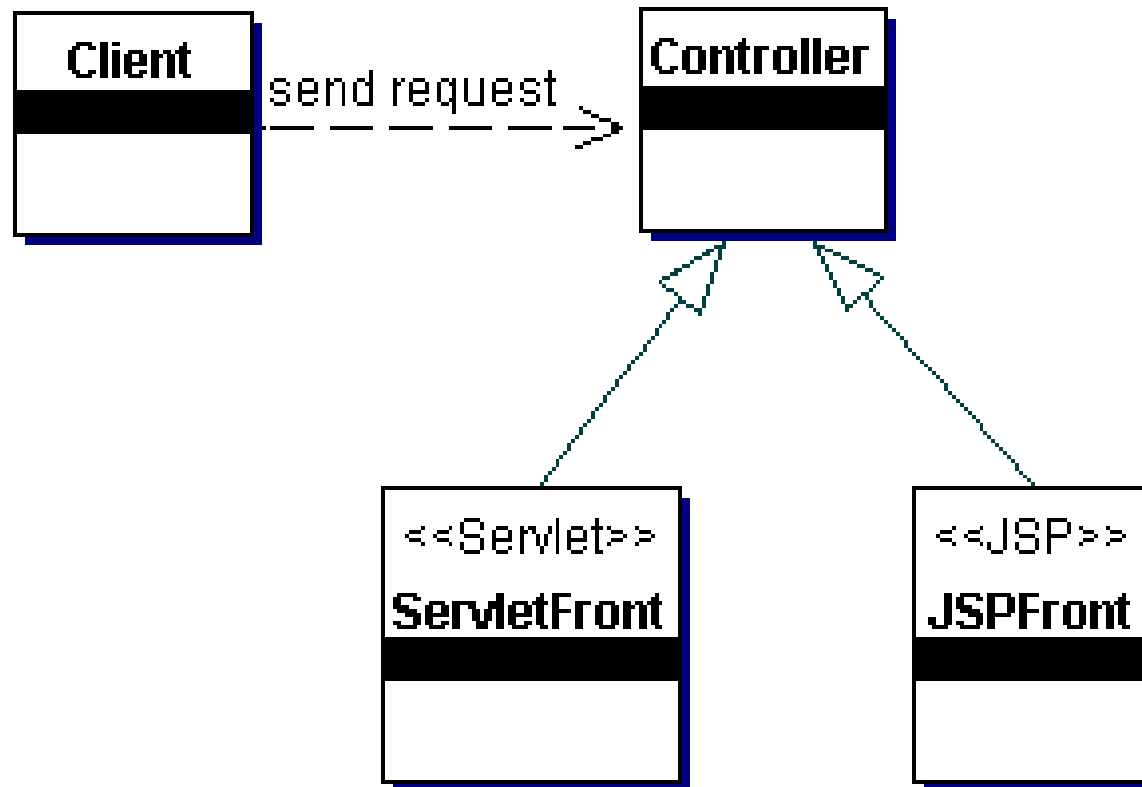
- Usar un controlador como el punto inicial de contactos, para manejar las peticiones.
- Con el controlador, y reduciendo la lógica de negocios en la vista, se logra la reutilización de código.
- Aun cuando se sugiere la centralización, no se limita el número de manejadores en el sistema.





FRONT CONTROLLER

■ Estructura.





FRONT CONTROLLER

■ Participantes y Responsabilidades.

○ Controller.

- Es el punto de contacto inicial, que maneja todas las peticiones al sistema.

○ Dispatcher.

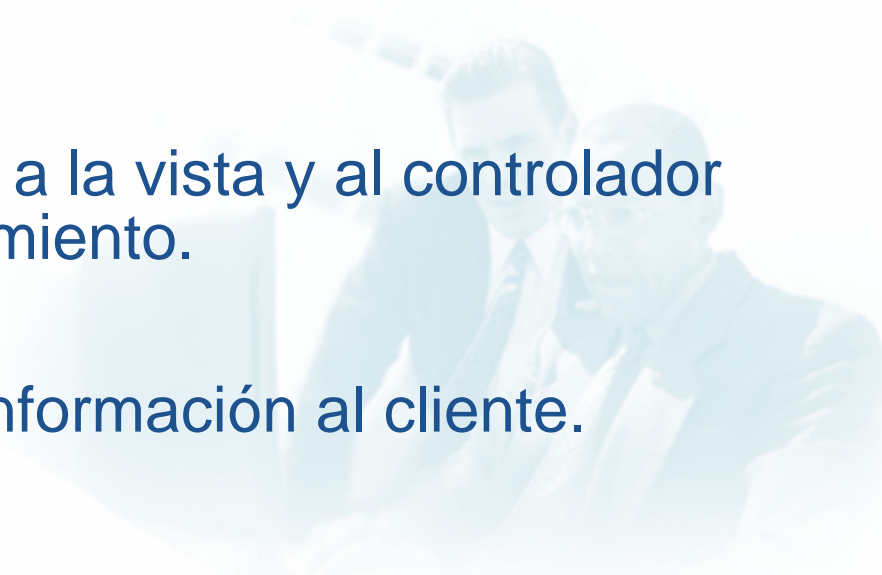
- Responsable del manejo de la vista y de la navegación.

○ Helper.

- Responsable de ayudar a la vista y al controlador a completar su procesamiento.

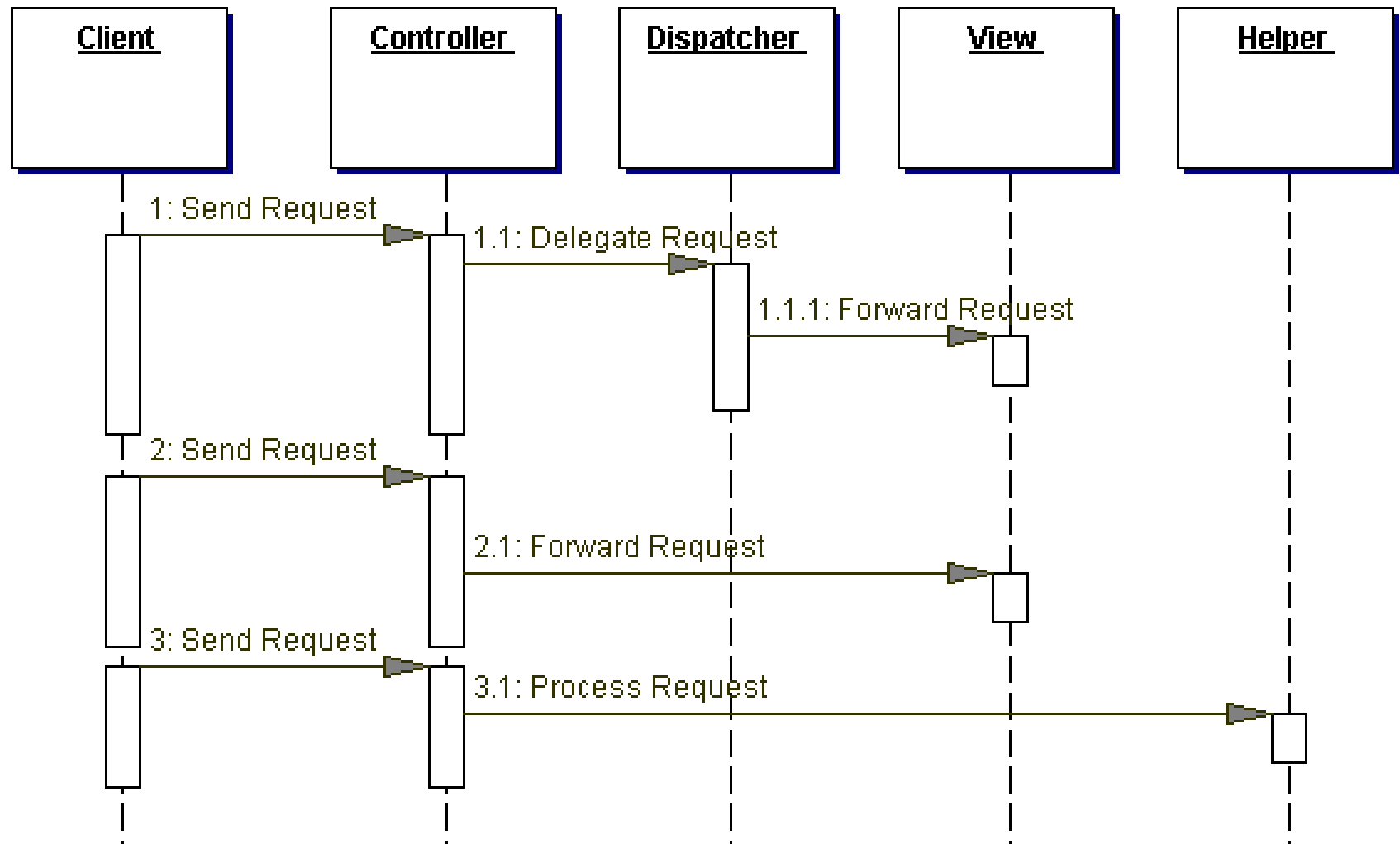
○ View.

- Representa y muestra información al cliente.





FRONT CONTROLLER





FRONT CONTROLLER

■ Estrategias.

- **Servlet Front:** Sugiere la utilización de un servlet como **controlador**.
- **JSP Front:** No se recomienda. Se requiere que el desarrollador trabaje en base a paginas de etiquetas.
- **Command and Controller:** Sugiere una interfase genérica para los componentes *helper*.



FRONT CONTROLLER

- **Consecuencias.**

- Centraliza el control.
- Mejora la manejabilidad de la seguridad.
- Mejora la Reutilización.





VIEW HELPER





VIEW HELPER

- Es un objeto que encapsula la lógica de acceso a datos en beneficio de los componentes de la presentación.
- Ejemplo, los **JavaBeans** pueden ser usados como patrón ***View Helper*** para las páginas JSP.





■ Contexto.

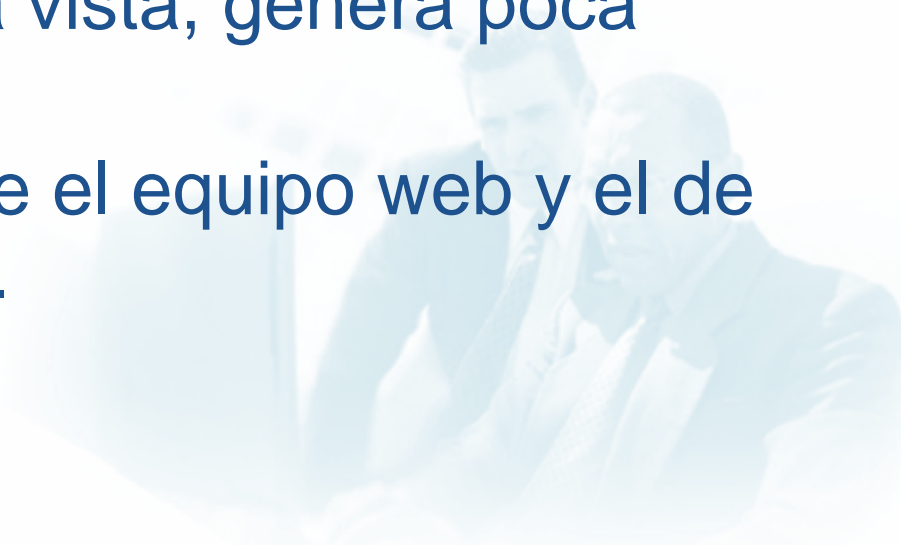
- El sistema CREA el contenido de la presentación, lo que requiere el procesamiento de datos de negocio dinámico.





■ Problema.

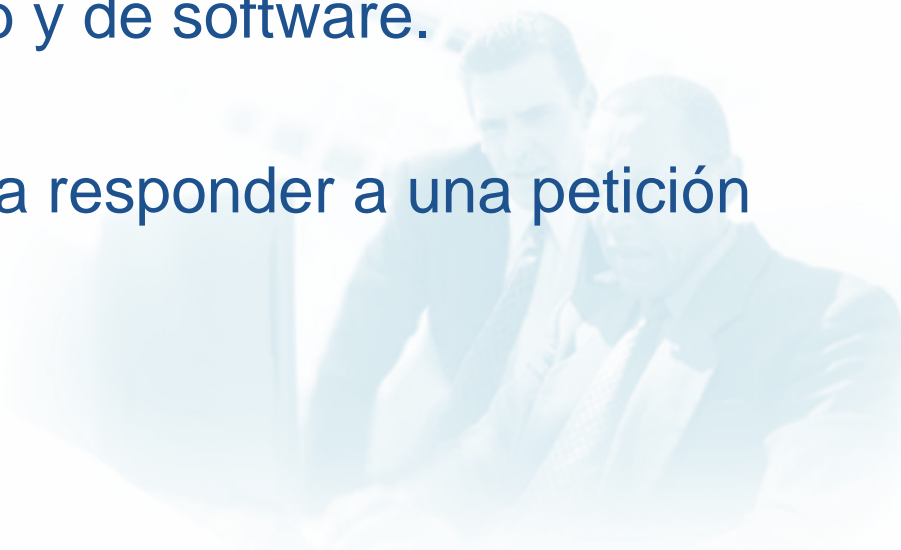
- Cambios en la lógica de la capa de presentación ocurren muy frecuentemente.
- Mezclar lógica de negocio y de sistema, con el procesamiento de la vista, genera poca modularidad.
- Pobre separación entre el equipo web y el de desarrollo de software.





■ Causas.

- Requerimientos no triviales en la lógica.
 - Embeber la lógica de negocio, genera errores, ya que por lo general es de ***copy and paste***.
 - Se desea una separación limpia y efectiva entre los desarrolladores web y de software.
 - Una vista es usada para responder a una petición de negocio particular.





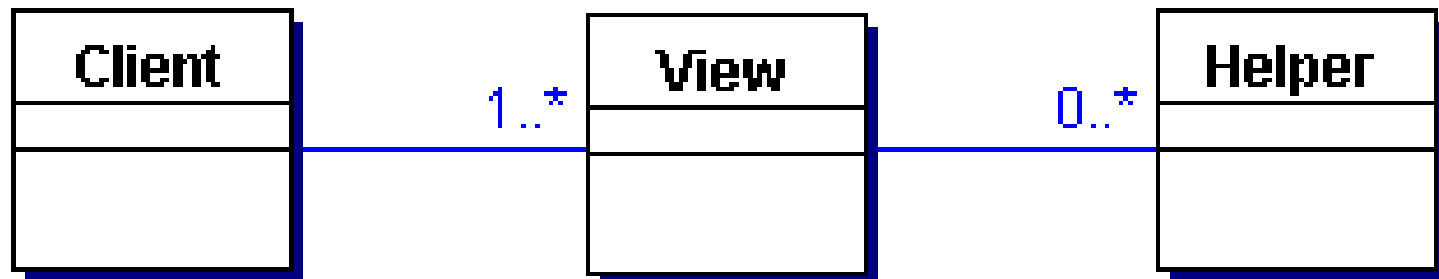
■ Solución.

- Una vista contiene código de formateo, delegando sus responsabilidades de procesamiento en sus clases de ayuda, implementadas como JavaBeans o etiquetas personalizadas.
- Existen varias estrategias para implementar la vista. JSP View es el más usado.
- Encapsular la lógica de negocio en un **help**, hace de la aplicación más modular todavía.
- Este **patrón** se enfoca en formas de particionar la responsabilidad de la aplicación



VIEW HELPER

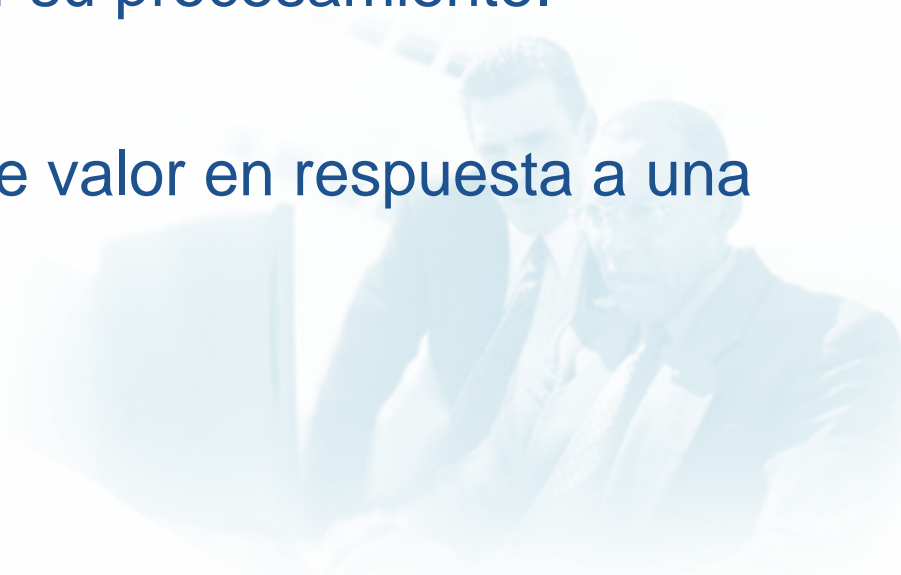
■ Estructura.





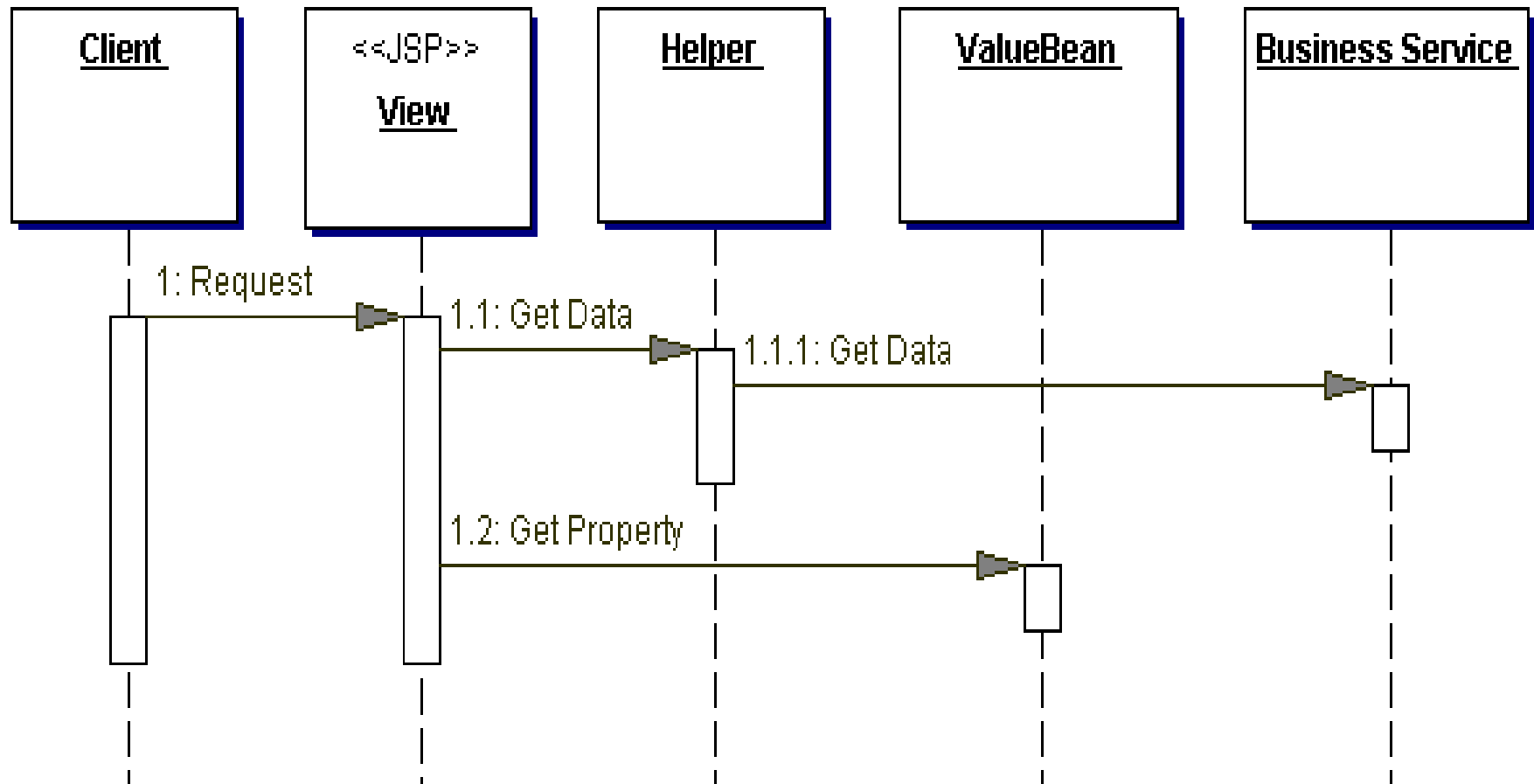
VIEW HELPER

- **Participación y Responsabilidad.**
 - View.
 - Representa y muestra información al cliente.
 - Helper.
 - Es el responsable de ayudar a la vista y al controlador a completar su procesamiento.
 - ValueBean.
 - Se devuelve un bean de valor en respuesta a una petición.
 - BusinessService.





VIEW HELPER





■ Estrategias.

- ***JSP View***: Sugiere la utilización de un componente **jsp** como una vista.
- ***Servlet View***: Utilización de un Servlet como una vista.
- ***JavaBeans Helper***: Resulta en una separación limpia de la vista, y el procesamiento de negocio.





■ Consecuencias.

- Mejora el particionamiento de la Aplicación, la Reutilización y el Mantenimiento:

Clara separación de la vista, del procesamiento de negocios.

- Mejora la separación de roles:

Separación de la lógica de formateo y la lógica de negocio.





COMPOSITE VIEW





COMPOSITE VIEW

- Es un objeto vista que está compuesto de otros objetos del mismo tipo.
- Ejemplo, una página JSP que incluye otras páginas JSP y HTML usando la directiva *include* o el *action include*, es un patrón **Composite View**.





COMPOSITE VIEW

■ Contexto.

- Las pagina web actuales, presentan contenidos de varias fuentes de datos.
- Además, distitas personas trabajan en el mantenimiento de dichos sitios, no teniendo siempre los mismos conocimientos y habilidades.





COMPOSITE VIEW

■ Problema.

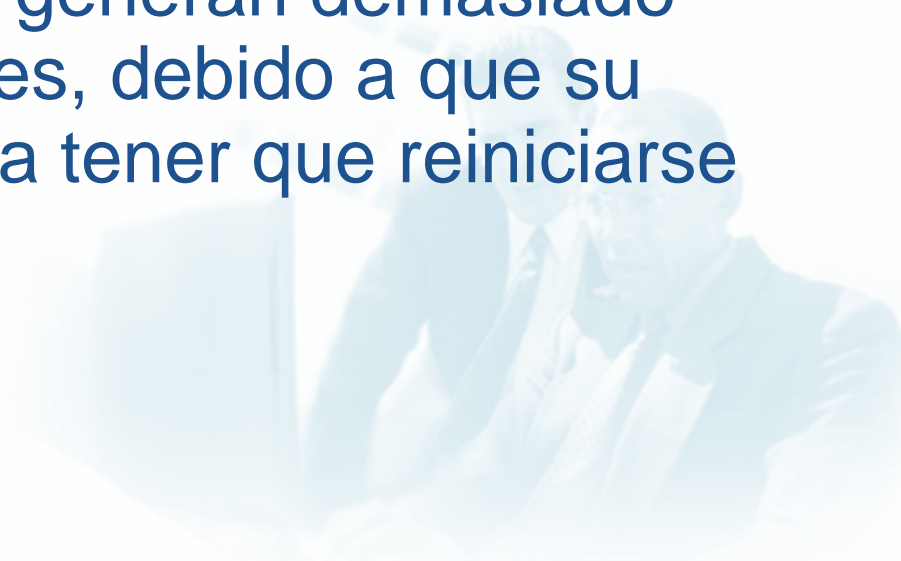
- En vez de generar el trabajo final, en base a mecanismos de combinación modular, donde proporciones atómicas se unen para componer una pagina completa, la pagina se construye, embebiendo código desarrollado por distintos entes.
- Difícil modificación de múltiples vistas.
Propenso a errores (*duplicación de código*).



COMPOSITE VIEW

■ Causas.

- Las partes del todo, cambian continuamente.
- Los cambios de plataforma son difíciles de manejar y mantener, por la naturaleza de código (embebido).
- Las plantillas de texto, generan demasiado trabajo en los servidores, debido a que su actualización los lleva a tener que reiniciarse en algunas ocasiones.





COMPOSITE VIEW

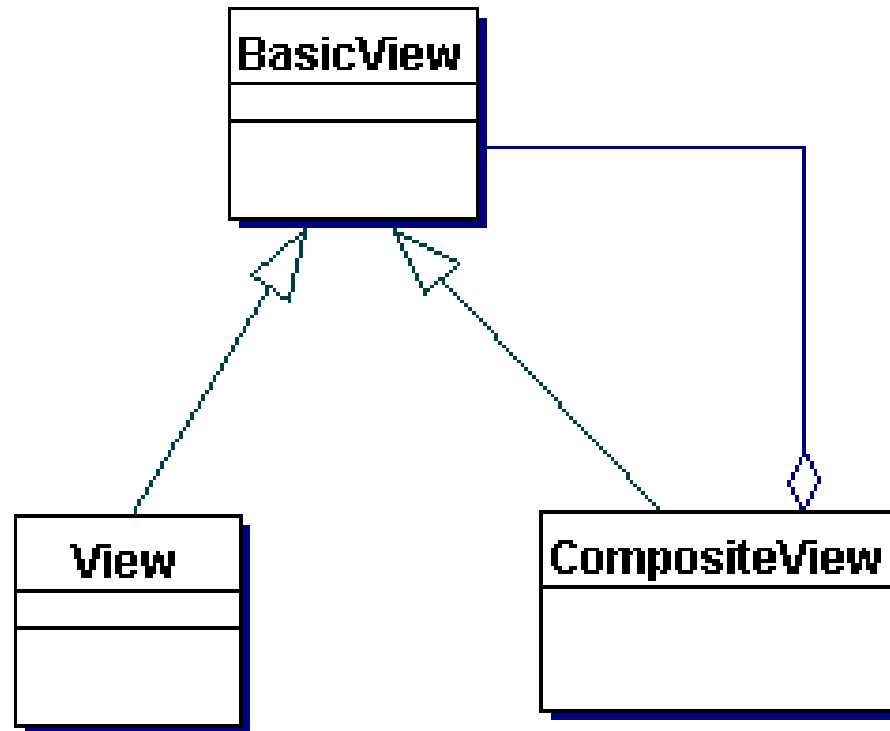
■ Solución.

- Utilizar vistas compuestas, que se generan en base a varias subvistas atómicas.
- Cada componente es incluido en forma dinámica, y la distribución de la pagina se maneja en forma independiente del contenido.
- Promueve la creación de vistas compuestas, basadas en la inclusión y sustitución de fragmentos de plantillas.



COMPOSITE VIEW

■ Estructura.





COMPOSITE VIEW

- **Participantes y Responsabilidades.**

- **View Manager.**

El controlador de Vista maneja la inclusión de porciones de fragmentos de plantilla en la vista compuesta.

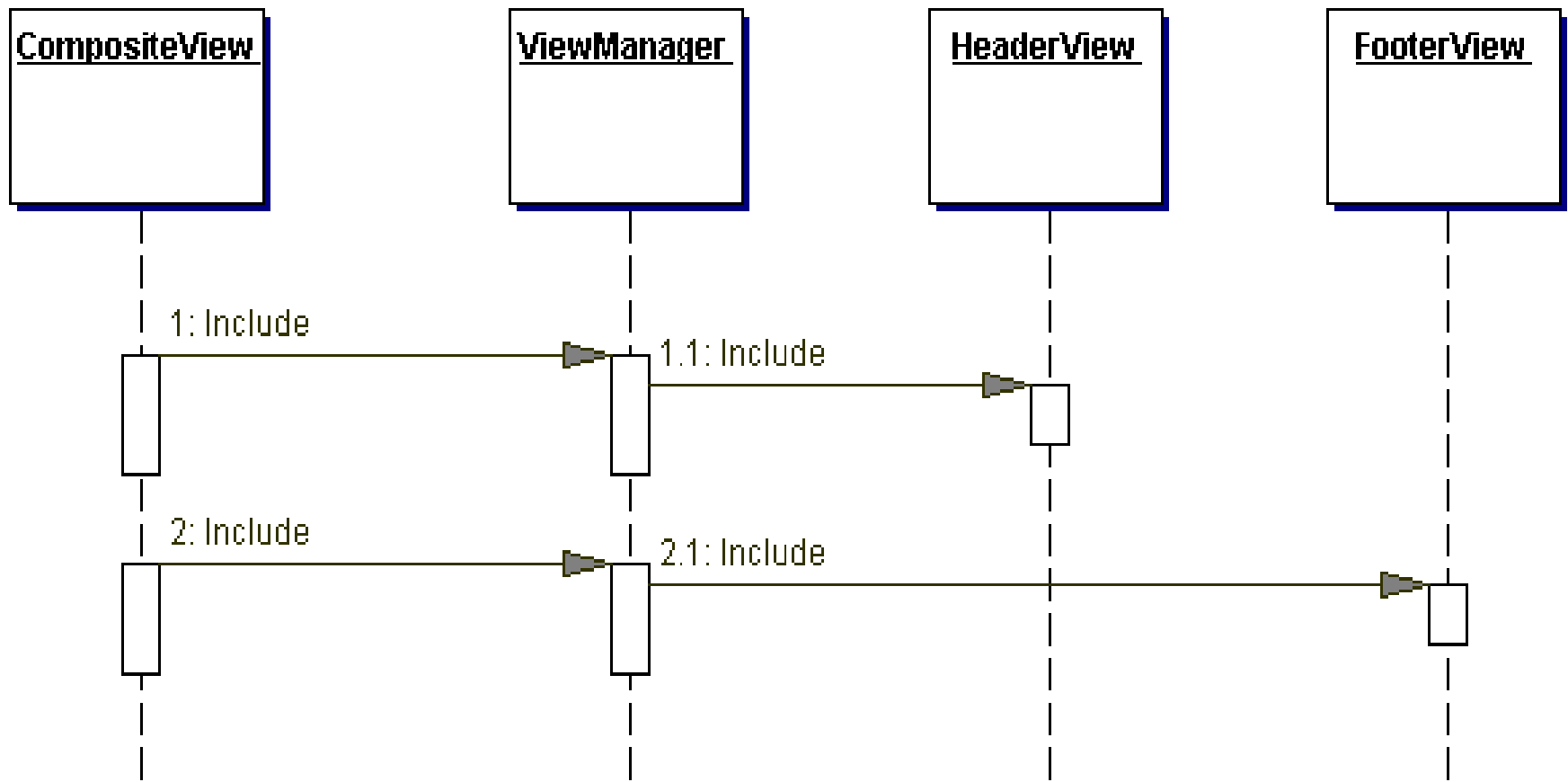
- **Included View.**

Es una vista atómica, que compone una vista mayor.





COMPOSITE VIEW

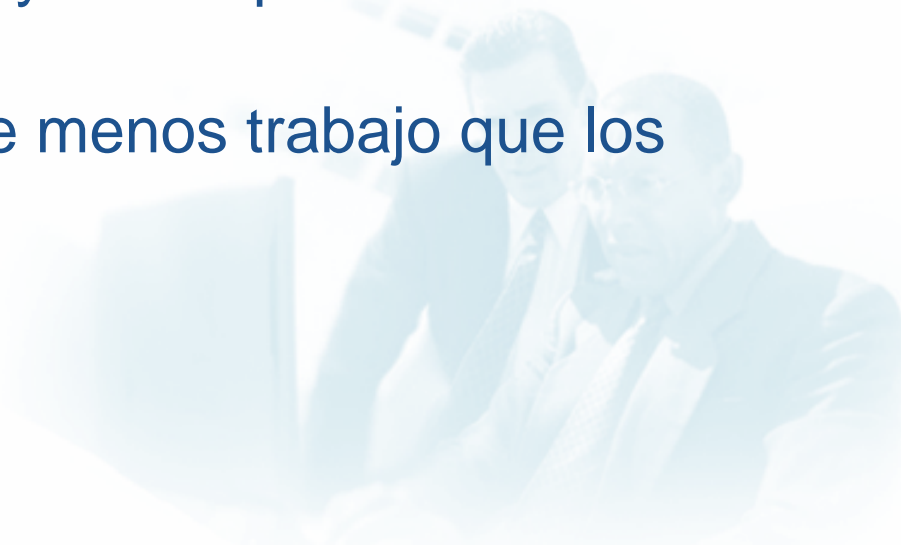




COMPOSITE VIEW

- **Estrategias.**

- **JSP page View.** (Como en el Patrón anterior)
- **Servlet View.** (Como en el patrón anterior)
- **JavaBeans View Management.**
 - El javaBeans implementa la lógica para el controlar la distribución y la composición de la vista.
 - Esta estrategia requiere menos trabajo que los casos anteriores.





COMPOSITE VIEW

■ Consecuencias.

- Mejora la Modularidad y la reutilización.
- Mejora la flexibilidad.
- Mejora el mantenimiento y la Manejabilidad.
- Reduce la Manejabilidad.
- Impacta de forma positiva en el rendimiento.





SERVICE to WORKER





SERVICE to WORKER

- Es como el patrón de diseño MVC con el Controlador actuando como **Front Controller**, pero con una cosa importante:
 - Aquí el Dispatcher (el cual es parte del Front Controller) usa View Helpers a gran escala y ayuda en el manejo de la vista.





SERVICE to WORKER

■ Contexto.

- El sistema controla el flujo de ejecución y accede a los datos de negocio, desde los que crea el contenido de presentación.
- Este patrón, describe una combinación común de otros patrones de catalogo.





SERVICE to WORKER

■ Problema.

- No existe un componente centralizado para el manejar el control de acceso, la recuperación de contenido, o el manejo de la vista.
- La mezcla de lógica de negocios con el procesamiento de la vista, genera poca modularidad.

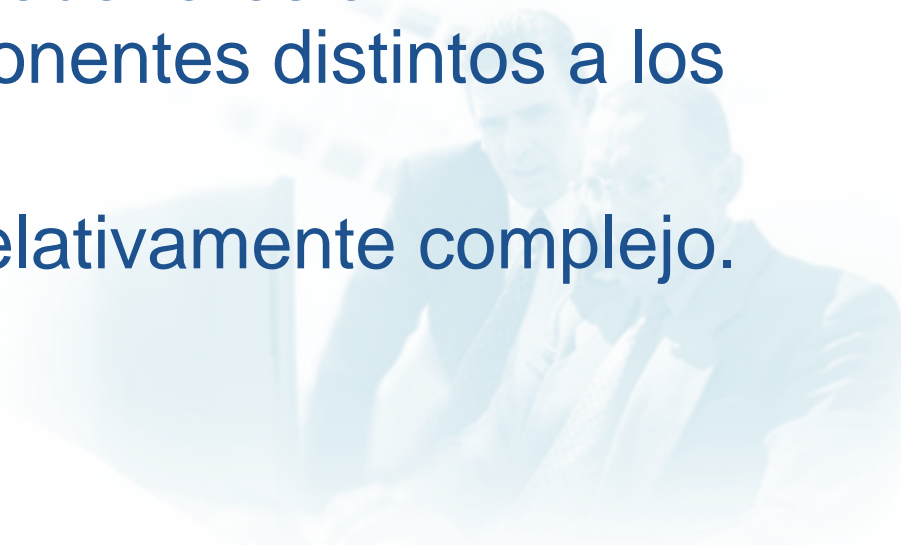




SERVICE to WORKER

■ Causas.

- Los chequeos de autenticación y autorización se ejecutan en cada petición.
- El código scriptlet dentro de la vistas se debería minimizar.
- La lógica de negocio debería esta encapsulada en componentes distintos a los de la vista.
- El control de flujo es relativamente complejo.





SERVICE to WORKER

■ Solución.

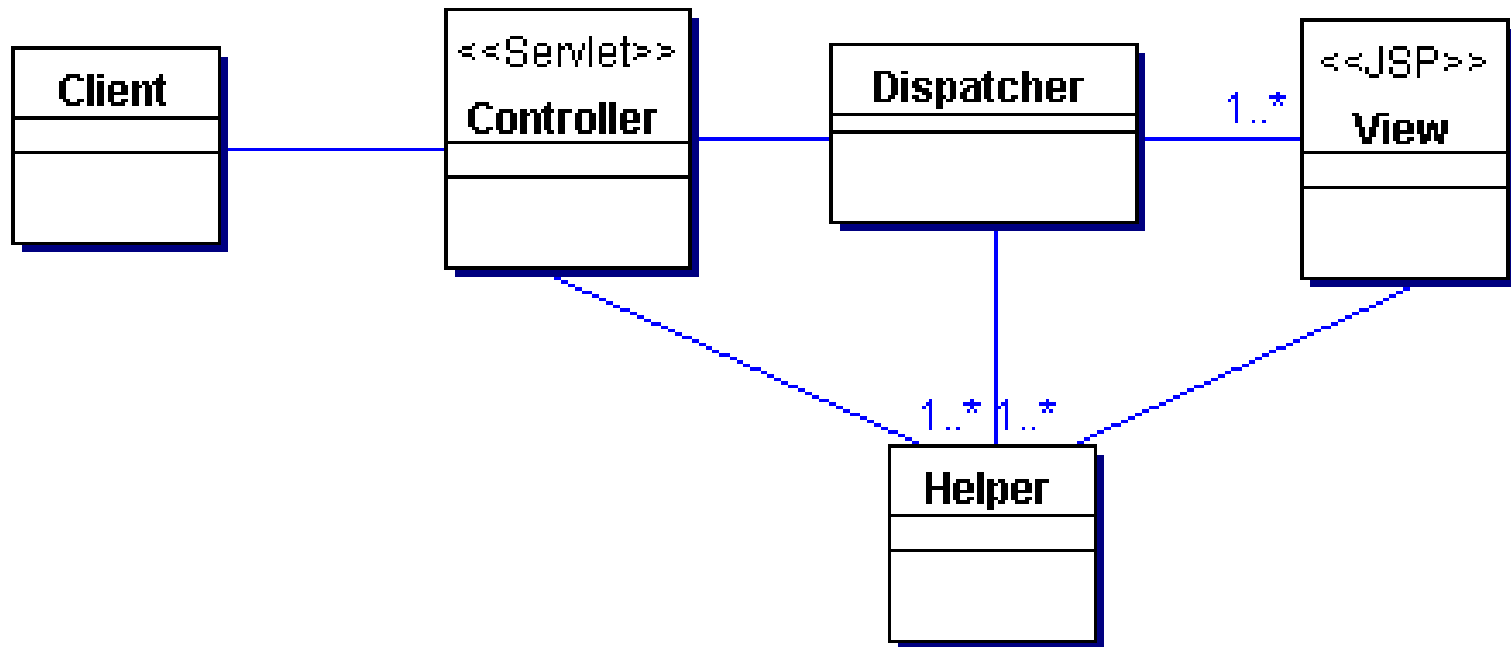
- Combinar un controlador y un dispatcher con vistas y *helper* para manejar peticiones de clientes, y generar una presentación dinámica como respuesta.
- Los controladores delegan la recuperación de contenido a los ***helpers***, que completan el modelo intermedio para la vista.





SERVICE to WORKER

■ Estructura.





SERVICE to WORKER

■ Participantes y Responsabilidades.

- Controller.
- Dispatcher.
- View.
- Helper.
- ValueBean.
- BusinessService.





SERVICE to WORKER

- **Estrategia.**
 - **Servelet Front.**
 - **JSP Front.**
 - **JSP page View.**
 - **Servlet View.**
 - **JavaBean Help.**





SERVICE to WORKER

■ Consecuencias.

- Centraliza el control, y mejora la modularidad y la reutilización.
- Mejora el particionamiento de las aplicaciones.
- Mejora la separación de Roles.





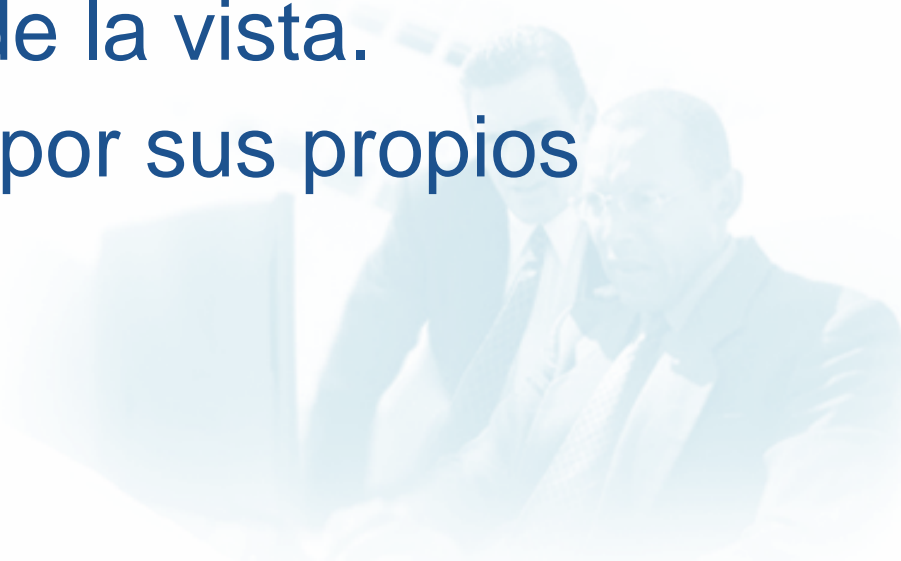
DISPATCHER VIEW





DISPATCHER VIEW

- Patrón de diseño MVC con el controlador actuando como ***Front Controller***.
- Importante Diferencia: Aquí el Dispatcher (el cual es parte del Front Controller).
- No usa ***View Helpers*** y realiza muy poco trabajo en el manejo de la vista.
- La vista es manejada por sus propios componentes.





DISPATCHER VIEW

■ Contexto.

- El sistema controla el flujo de ejecución y accede a los datos de negocio, desde los que crea el contenido de presentación.
- Este patrón, describe una combinación común de otros patrones de catalogo.





DISPATCHER VIEW

- **Problema.**

- Resuelve la combinación de los problema resueltos por **Font Controller** y **View Helper** de la capa de presentación.





Capa de Negocios





Business Delegate

- Un objeto que reside en la capa de presentación y en beneficio de los otros componentes de la capa de presentación llama a métodos remotos en los objetos de la capa de negocios.





Business Delegate

■ Contexto

- Un sistema multi-capa distribuido requiere invocación remota de métodos para enviar y recibir datos entre las capas. Los clientes están expuestos a la complejidad de tratar con componentes distribuidos.





Business Delegate

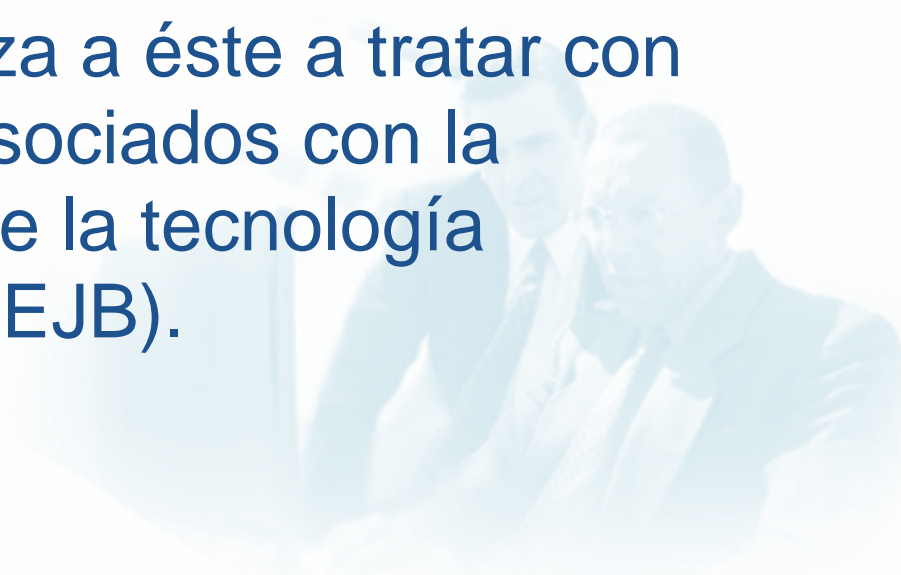
■ Problema

- Los componentes de la capa de presentación interactúan directamente con servicios de negocio, esto expone los detalles de la implementación del API del servicio de negocio a la capa de presentación. Como resultado cuando cambia la implementación del servicio de negocio, la implementación del código expuesto en la capa de presentación también debe cambiar.



Business Delegate

- Además, podría haber una reducción de rendimiento en la red porque los componentes de la capa de presentación que utilizan el API de servicio de negocio hacen demasiadas invocaciones sobre la red.
- Por último, exponer directamente los APIs de servicios al cliente fuerza a éste a tratar con los problemas de red asociados con la naturaleza distribuida de la tecnología Enterprise JavaBeans (EJB).

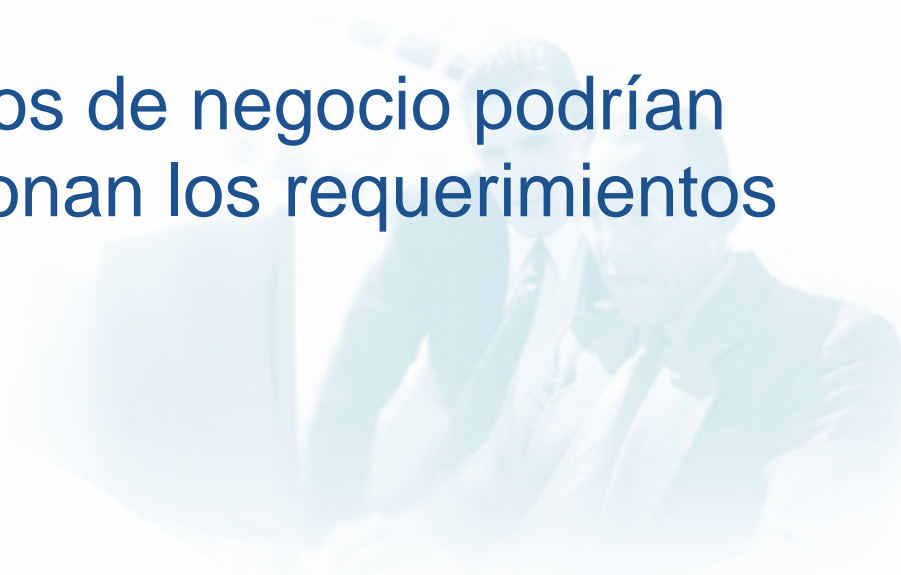




Business Delegate

■ Causas

- Los clientes de la capa de presentación necesitan acceder a servicios de negocio.
- Diferentes clientes, dispositivos, clientes Web, y programas, necesitan acceder a los servicios de negocio.
- Los APIs de los servicios de negocio podrían cambiar según evolucionan los requerimientos del negocio.
- ...





Business Delegate

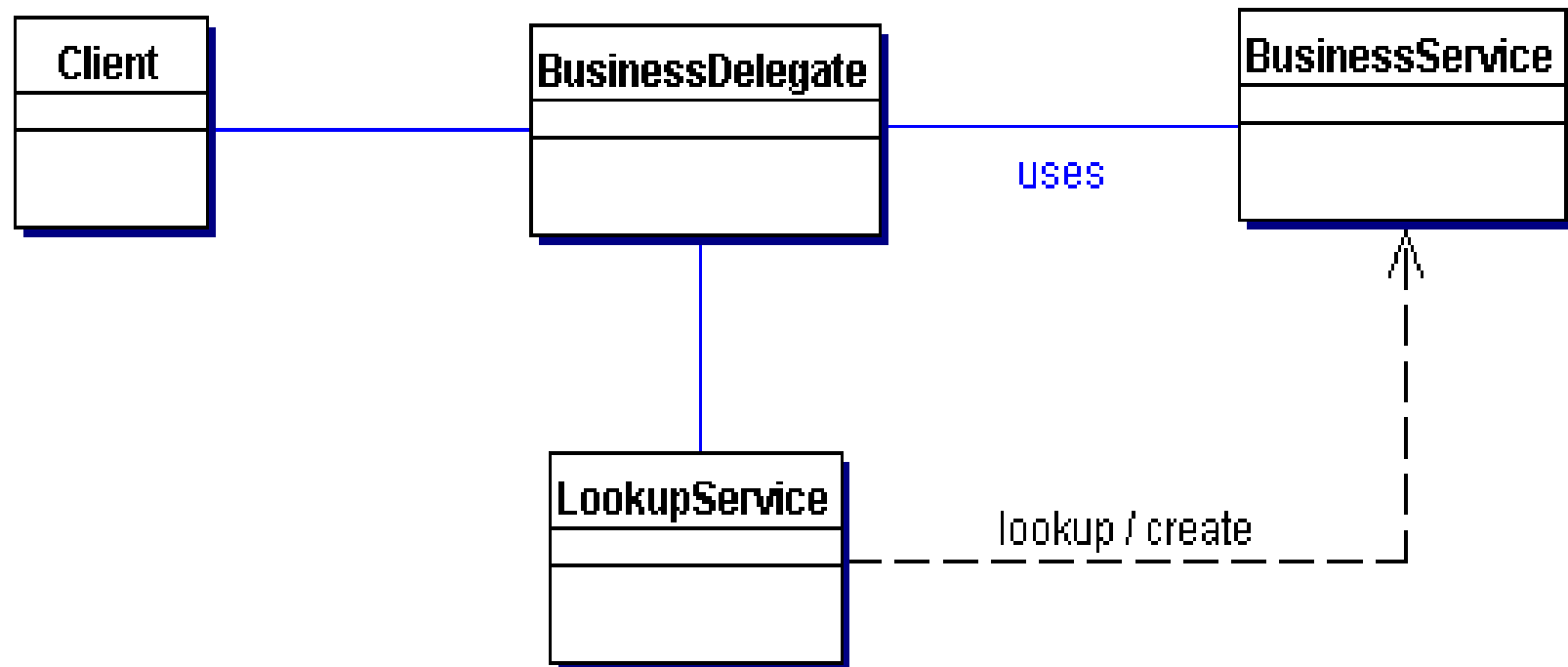
■ Solución

- Se utiliza un Business Delegate para reducir el acoplamiento entre los clientes de la capa de presentación y los servicios de negocio. El Business Delegate oculta los detalles de la implementación del servicio de negocio, como los detalles de búsqueda y acceso de la arquitectura EJB.



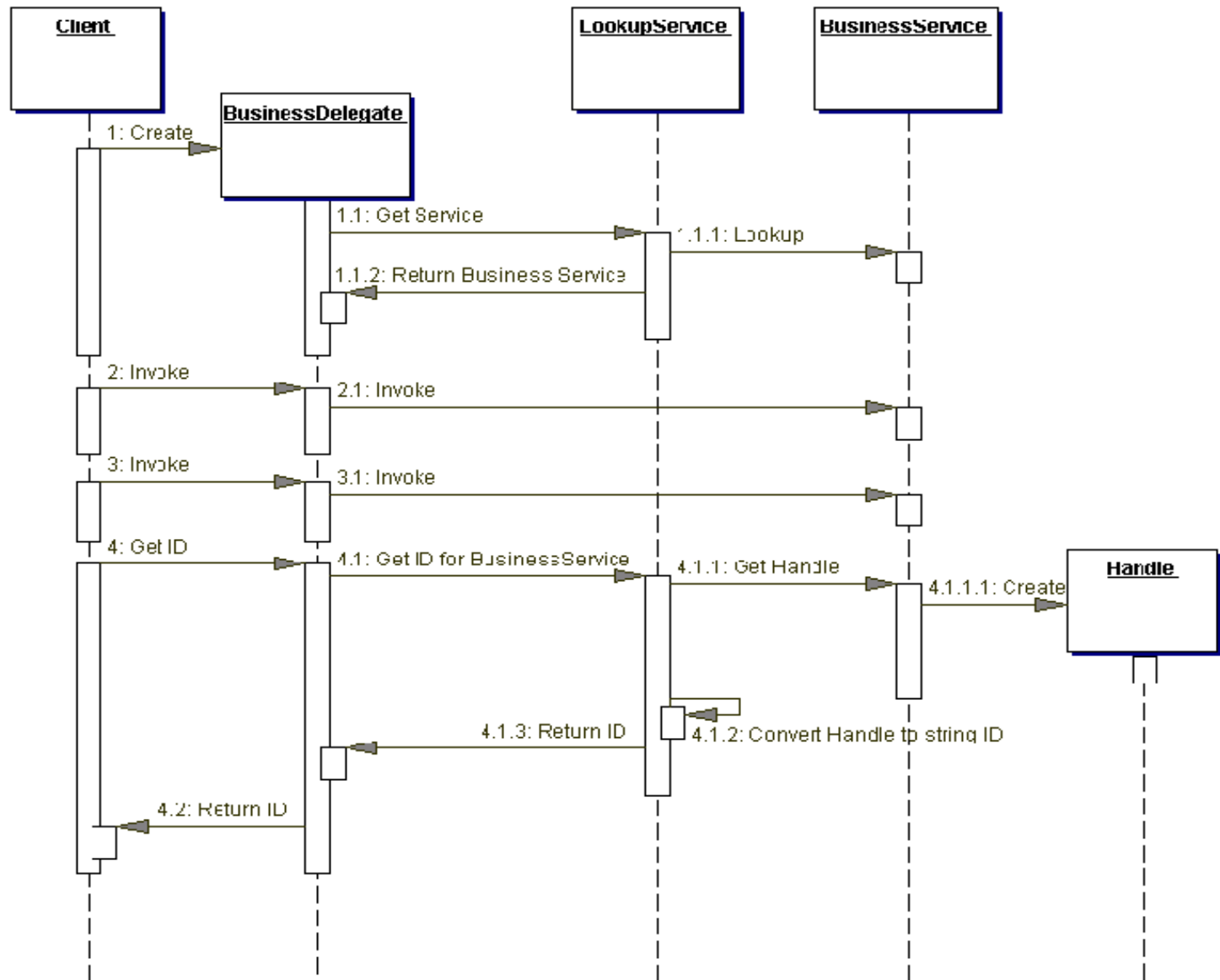


Business Delegate





Business Delegate





Business Delegate

■ Consecuencias

- Reduce el Acoplamiento, Mejora la Manejabilidad
- Traduce las Excepciones del Servicio de Negocio
- Implementa Recuperación de Fallos y Sincronización de Threads
- Expone un Interface Simple y Uniforme a la Capa de Negocio
- Impacto en el Rendimiento
- Presenta una Capa Adicional
- Oculta los elementos Remotos





Service Locator

- Consiste en utilizar un objeto Service Locator para abstraer toda la utilización JNDI (Java Naming and Directory Interface) y para ocultar las complejidades de la creación del contexto inicial, de búsqueda de objetos home EJB y recreación de objetos EJB. Varios clientes pueden reutilizar el objeto Service Locator para reducir la complejidad del código, proporcionando un punto de control.



Service Locator

- Contexto

- La búsqueda y creación de servicios implican interfaces complejos y operaciones de red.





Service Locator

■ Problemas

- Los clientes utilizan repetidamente el servicio JNDI, el código JNDI aparece varias veces en esos clientes. Esto resulta en una duplicación de código innecesaria en los clientes que necesitan buscar servicios.
- Crear un objeto de contexto JNDI y realizar una búsqueda para objeto home EJB utiliza muchos recursos. Si varios clientes requieren de forma repetitiva el mismo objeto home de un bean, puede impactar de forma negativa el rendimiento de la aplicación.



Service Locator

■ Causas

- Los clientes EJB necesitan utilizar el API JNDI para buscar objetos EJBHome utilizando el nombre registrado del bean enterprise.
- Los clientes JMS necesitan utilizar el API JNDI para buscar componentes JMS utilizando los nombres registrados en JNDI para esos componentes JMS.
- ...

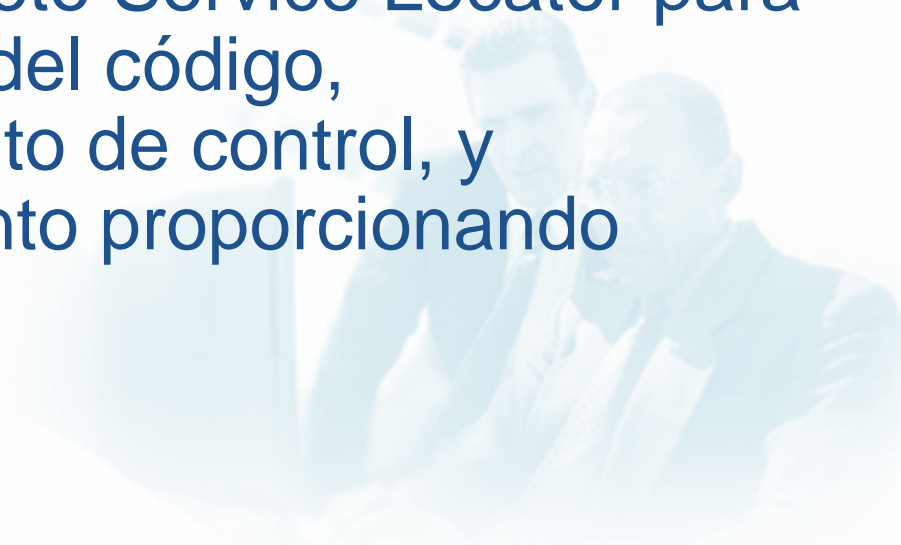




Service Locator

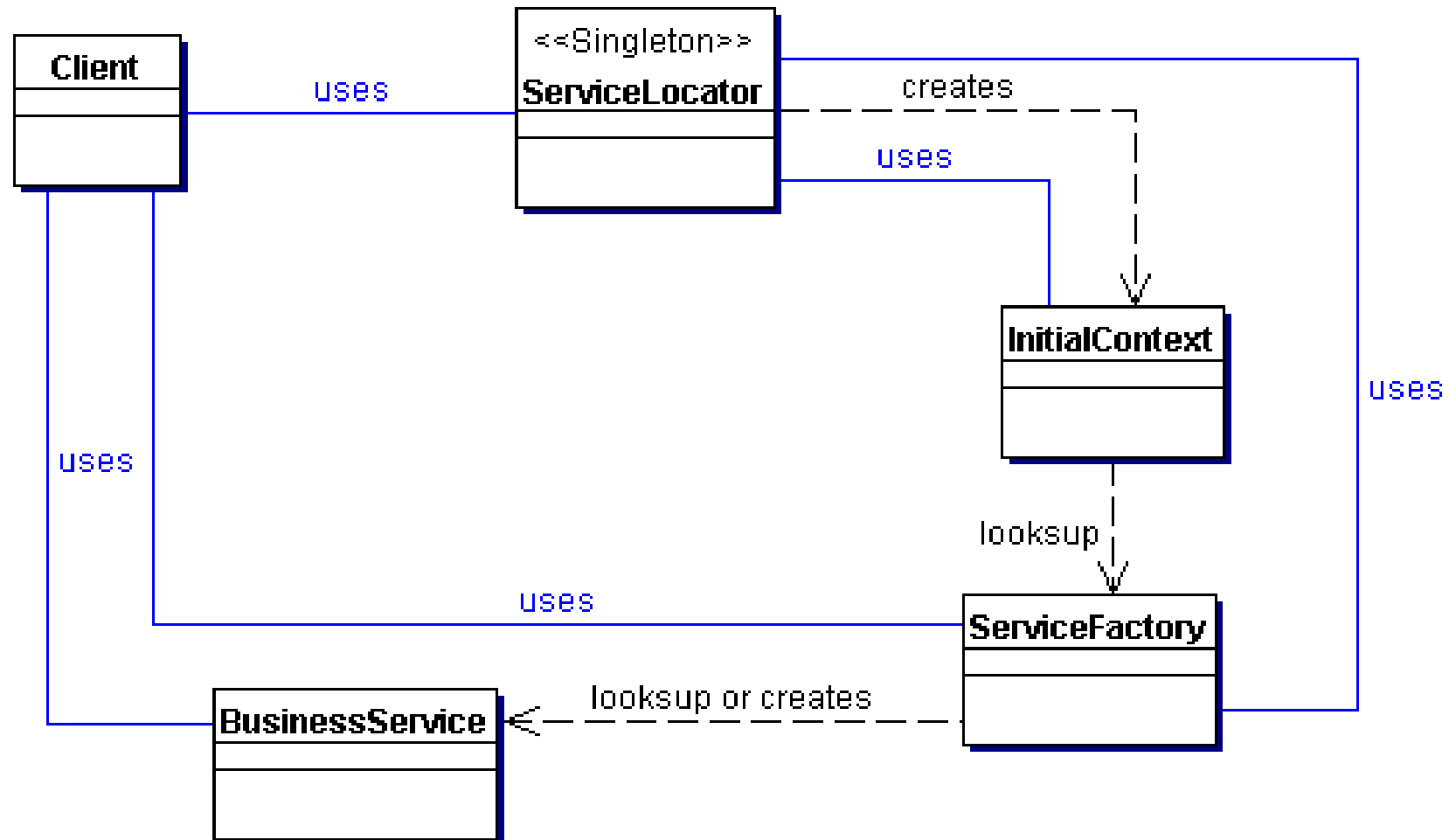
■ Solución

- Utilizar un objeto Service Locator para abstraer toda la utilización JNDI y para ocultar las complejidades de la creación del contexto inicial, de búsqueda de objetos home EJB y de re-creación de objetos EJB. Varios clientes pueden reutilizar el objeto Service Locator para reducir la complejidad del código, proporcionando un punto de control, y mejorando el rendimiento proporcionando facilidades de caché.

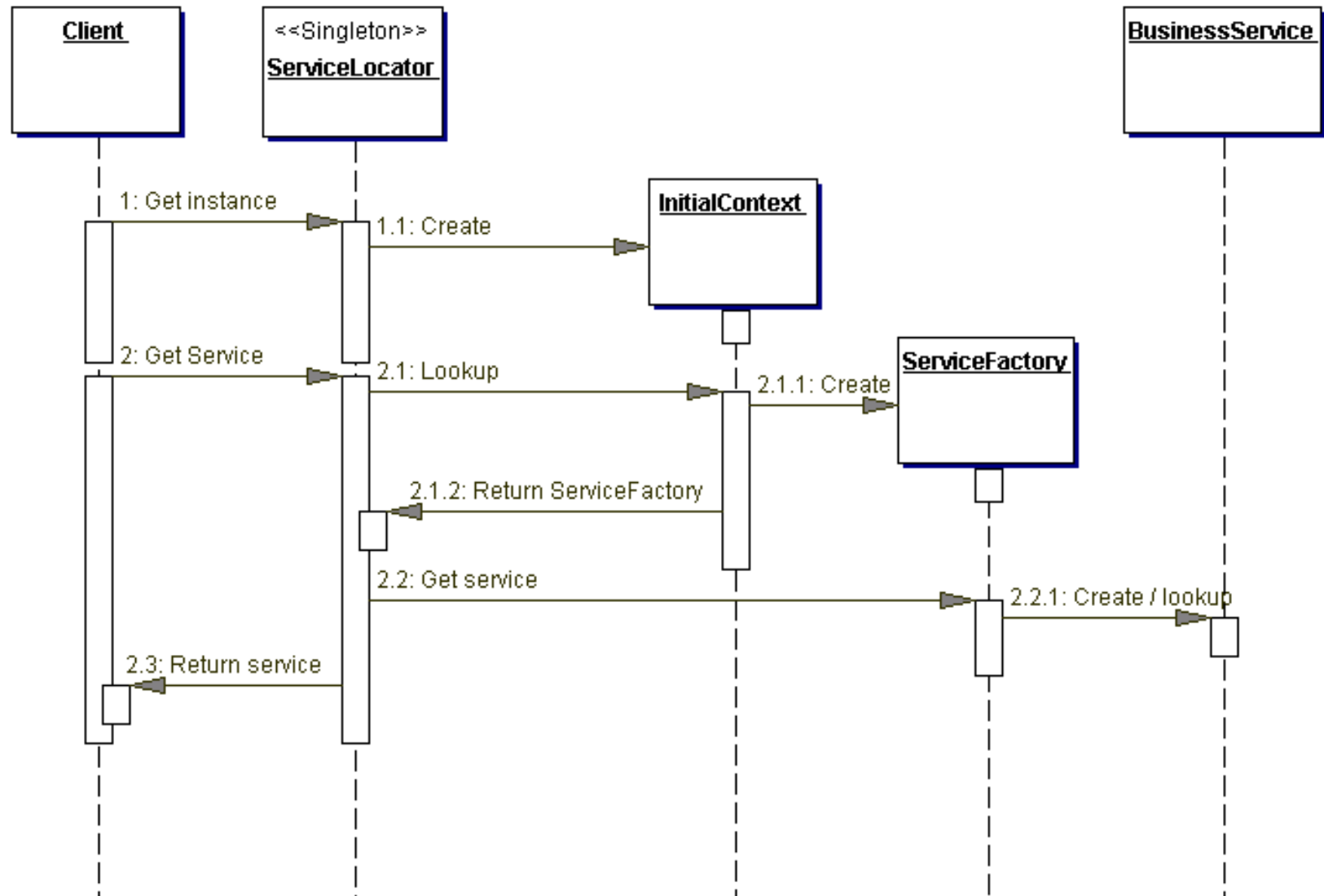




Service Locator



Service Locator





Service Locator

- Consecuencias
 - Abstrae la Complejidad
 - Proporciona a los Clientes un Acceso Uniforme a los Servicios
 - Facilita la Adicción de Nuevos Componentes de Negocio
 - Mejora el Rendimiento de la Red
 - Mejora el Rendimiento del Cliente mediante el Caché





Session Facade

- El uso de un bean de sesion como una fachada (facade) para encapsular la complejidad de las interacciones entre los objetos de negocio y participantes en un flujo de trabajo. El Session Facade maneja los objetos de negocio y proporciona un servicio de acceso uniforme a los clientes.





Session Facade

■ Contexto

- Los Beans Enterprise encapsulan lógica y datos de negocio y exponen sus interfaces, y con ellos la complejidad de los servicios distribuidos, a la capa de cliente.





Session Facade

■ Problema

- Acoplamiento fuerte, que provoca la dependencia directa entre los clientes y los objetos de negocio.
- Demasiadas llamadas a métodos entre el cliente y el servidor, abocando a problemas de rendimiento de la red.
- Falta de una estrategia de acceso uniforme de los clientes, exponiendo los objetos de negocio a una mala utilización.



Session Facade

■ Causas

- Proporcionar a los clientes un interface sencillo que oculte todas interacciones complejas entre los componentes de negocio.
- Reducir el número de objetos de negocio que se exponen al cliente a través de la capa de servicio sobre la red.
- ...





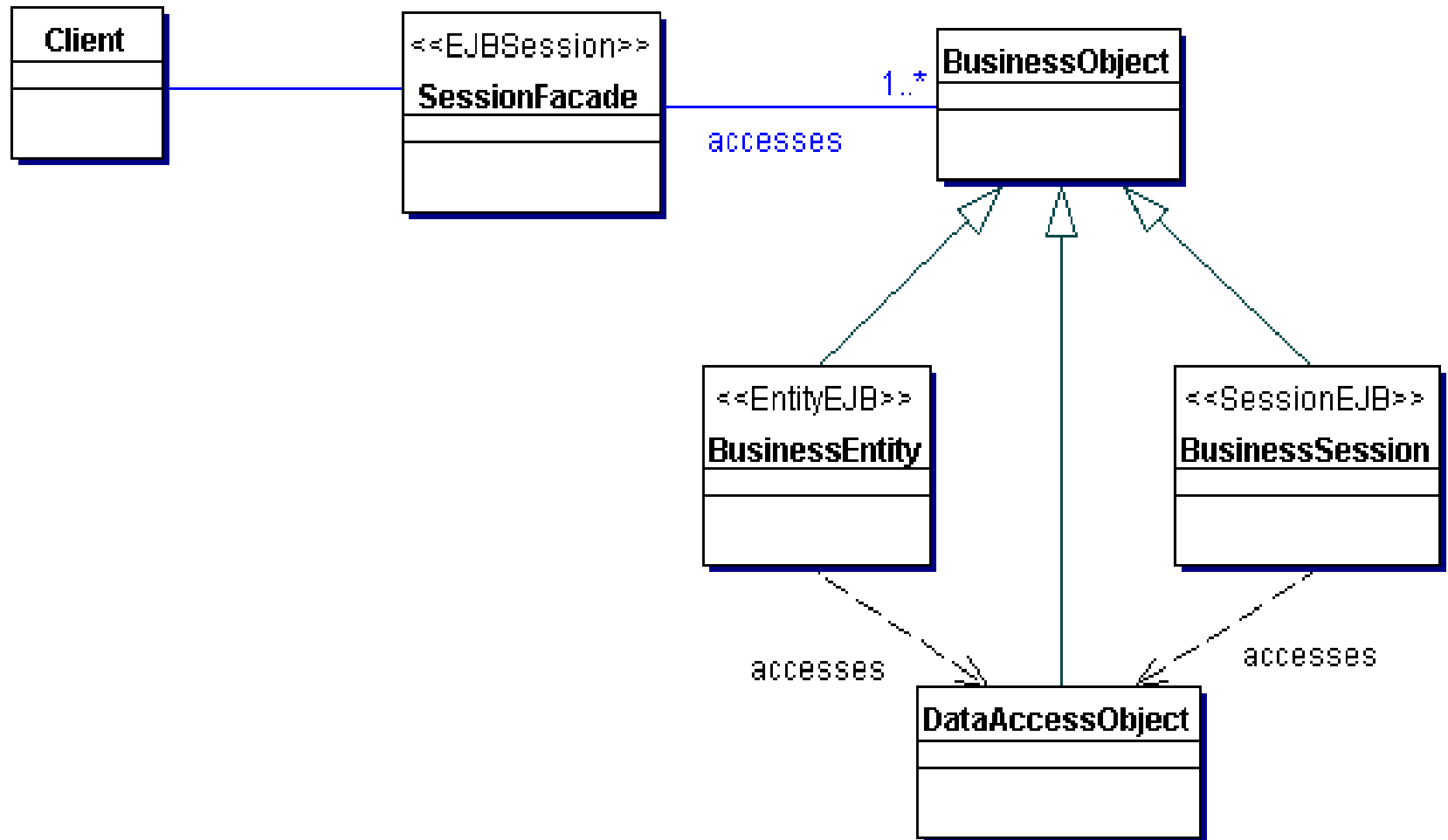
Session Facade

■ Solución

- Usar un bean de sesión como una fachada (facade) para encapsular la complejidad de las interacciones entre los objetos de negocio participantes en un flujo de trabajo. El Session Facade maneja los objetos de negocio y proporciona un servicio de acceso uniforme a los clientes.

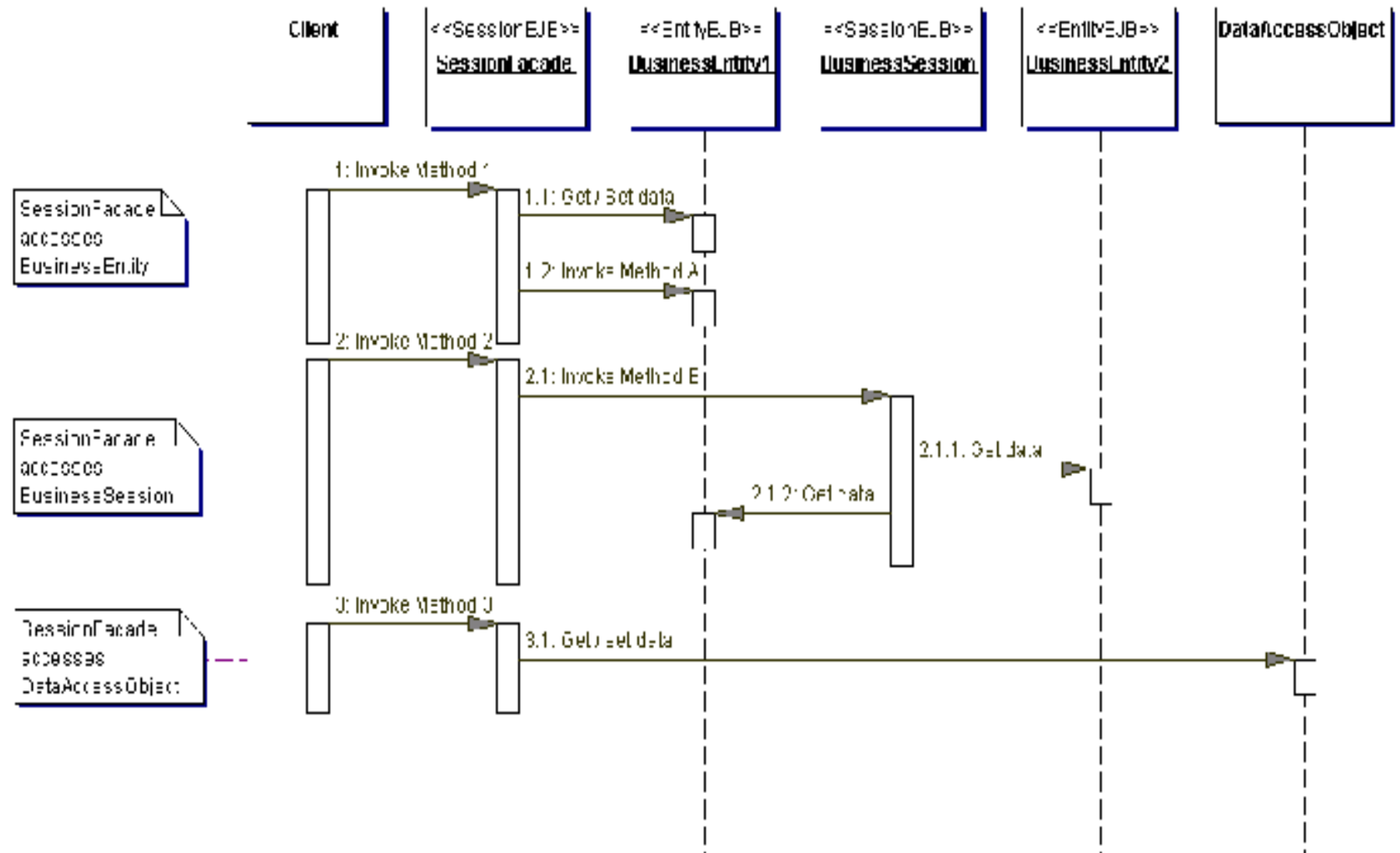


Session Facade





Session Facade





Session Facade

■ Consecuencias

- Introduce la Capa Controladora para la Capa de Negocio
- Expone un Interface Uniforme
- Reduce el Acoplamiento, Incrementa la Manejabilidad
- Mejora el Rendimiento, Reduce los Métodos Específicos
- Proporciona Acceso Genérico
- Centraliza el Control de Seguridad
- Centraliza el Control de Transacciones
- Expone Menos Interfaces Remotos



Transfer Object

- Un objeto serializable para la transferencia de datos sobre la red. También se le conoce como Data Transfer Object.
- Contexto
 - Las aplicaciones cliente necesitan intercambiar datos con EJB.





Transfer Object

■ Problemas

- Toda llamada a método hecha al objeto de servicio de negocio, ya sea a un bean de entidad o a un bean de sesión, potencialmente es una llamada remota. Así, en una aplicación EJB dichas llamadas remotas usan la capa de red sin importar la proximidad del cliente al bean, creando una sobrecarga en la red.





Transfer Object

- Utilizar varias llamadas a métodos get que devuelven simples valores de atributos es ineficiente para obtener valores de datos desde un bean enterprise.





Transfer Object

■ Causas

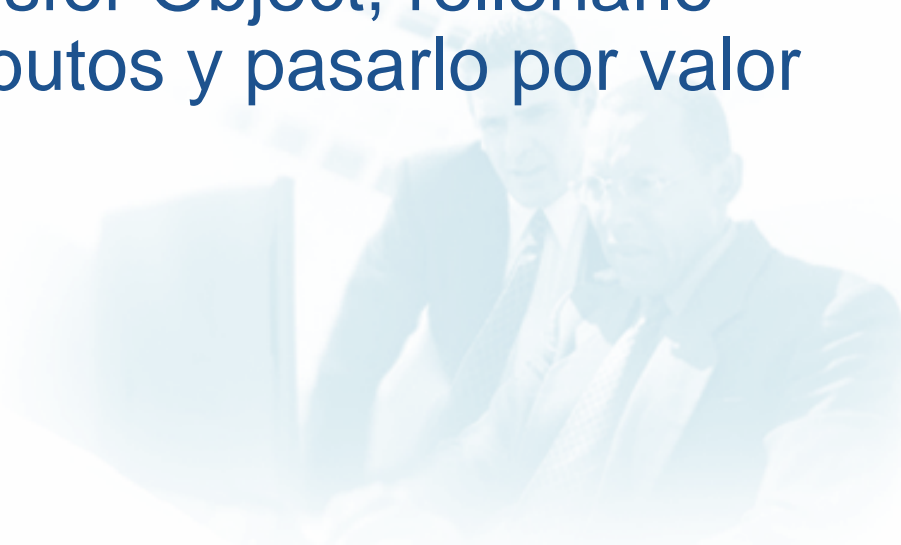
- Todos los accesos a un bean enterprise se realizan mediante interfaces remotos. Cada llamada a un bean enterprise potencialmente es una llamada a un método remoto con sobrecarga de red.
- Las aplicaciones tienen transacciones de lectura con mayor frecuencia que las de actualización. El cliente solicita los datos desde la capa de negocio para su presentación, y otros tipos de procesamientos de sólo-lectura.



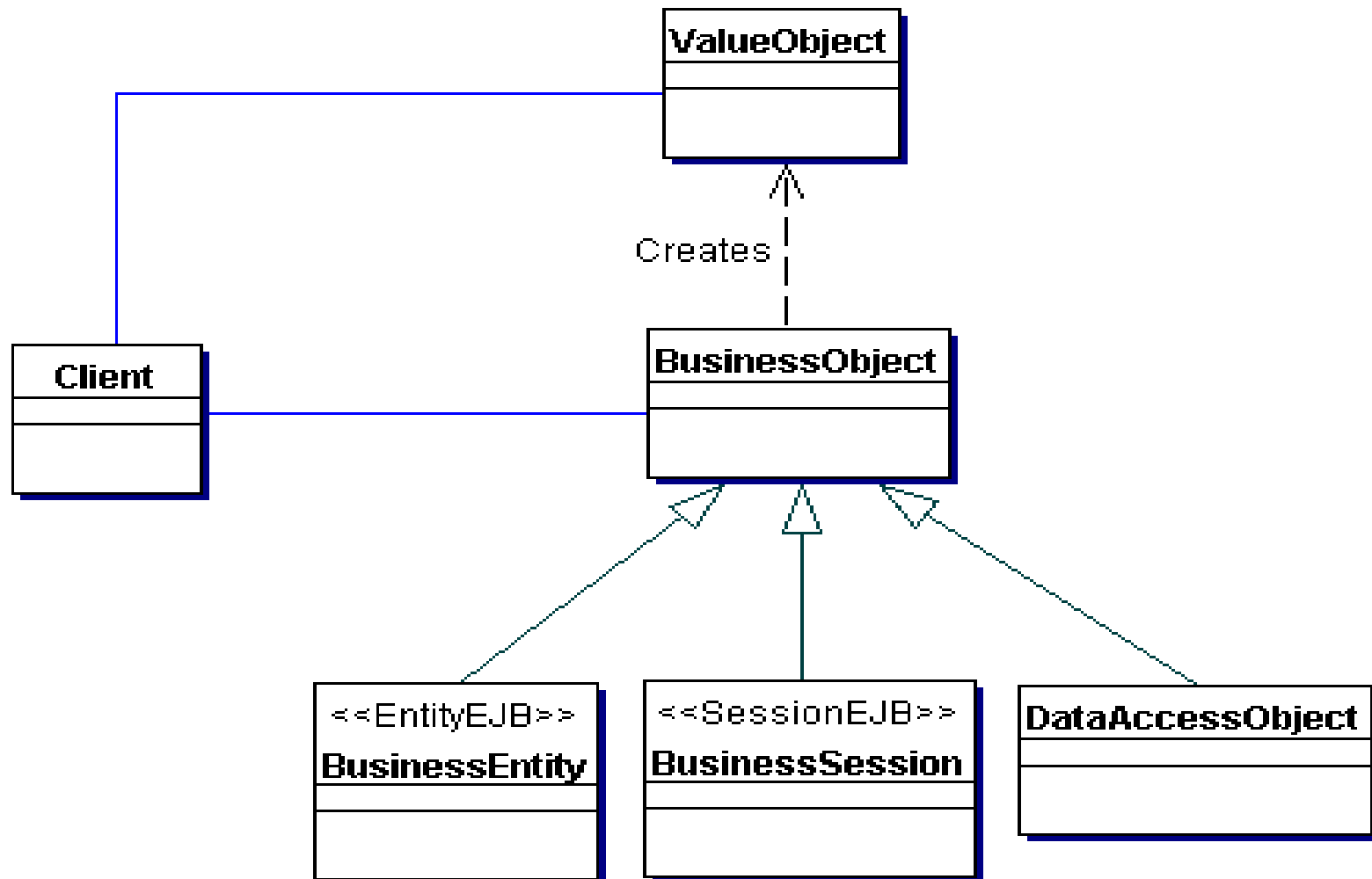
Transfer Object

■ Solución

- Utilizar un Transfer Object para encapsular los datos de negocio. Se utiliza una única llamada a un método para enviar y recuperar el Transfer Object. Cuando el cliente solicita los datos de negocio al bean enterprise, éste puede construir el Transfer Object, rellenarlo con sus valores de atributos y pasarlo por valor al cliente.

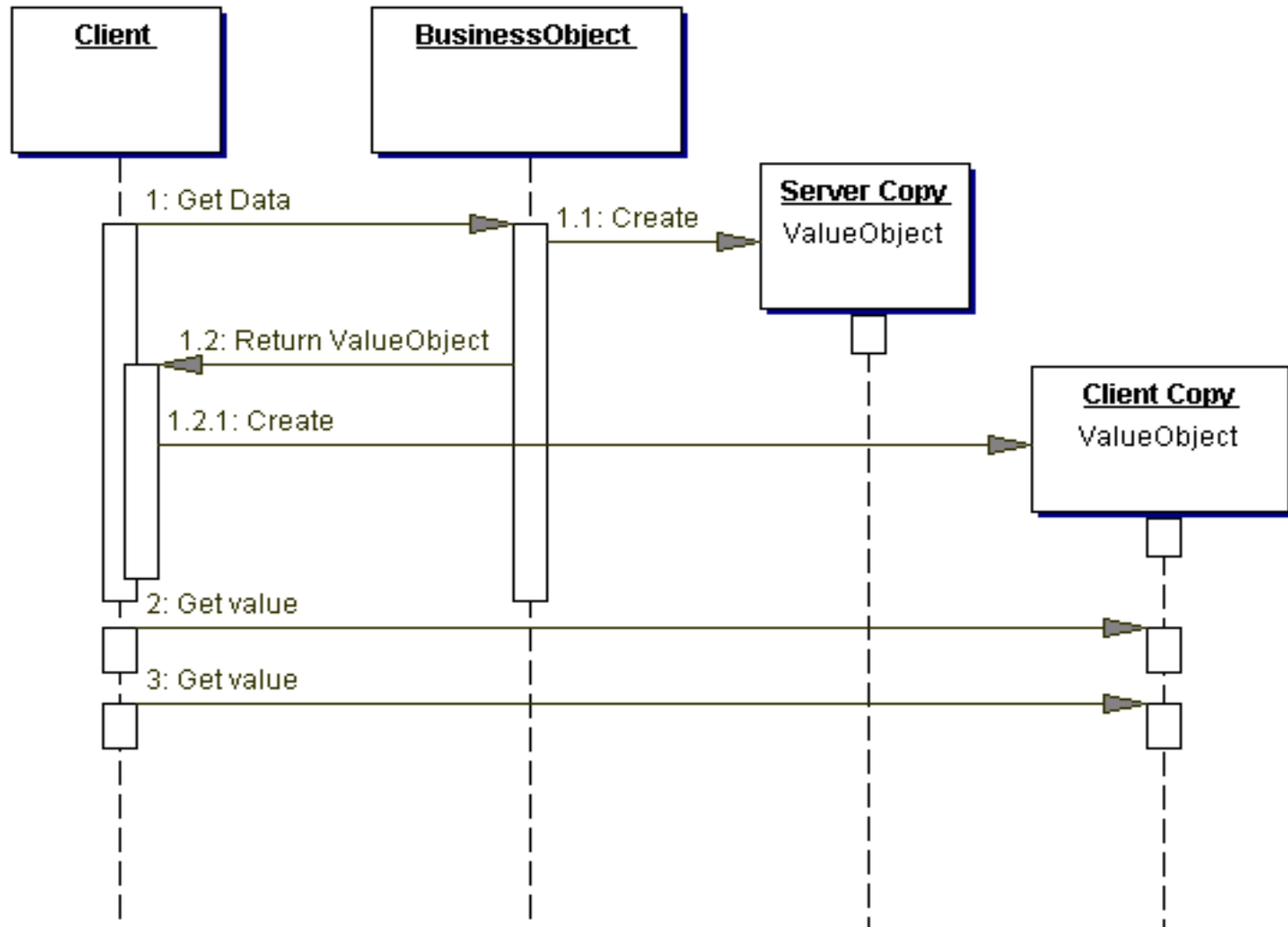


Transfer Object





Transfer Object





Transfer Object

■ Consecuencias

- Simplifica el Bean de Entidad y el Interface Remoto
- Transfiere Más Datos en Menos Llamadas Remotas
- Reduce el Tráfico de Red
- Reduce la Duplicación de Código
- Podría Introducir Transfer Objects Obsoletos
- ...





Transfer Object Assembler

- Un objeto que reside en la capa de negocios y crea Value Objects cuando es requerido.
- Contexto
 - En una aplicación de la plataforma J2EE, los componentes de negocio del lado del servidor se implementan utilizando beans de sesión, beans de entidad, DAOs, etc. Los clientes de la aplicación necesitan acceder a datos que frecuentemente se componen de múltiples objetos.



Transfer Object Assembler

■ Problema

- Como el cliente debe acceder a todos los componentes distribuidos individualmente, hay un acoplamiento fuerte entre el cliente y los componentes de negocio del modelo sobre la red.
- El cliente accede a los componentes distribuidos mediante la capa de red, y eso puede provocar una degradación del rendimiento si el modelo es complejo y con numerosos componentes distribuidos.



Transfer Object Assembler

■ Causas

- Se requiere la separación de la lógica de negocio entre el cliente y los componentes del lado del servidor.
- Como el modelo consta de componentes distribuidos, el acceso a cada componente está asociado con una sobrecarga de red.
- Incluso si el cliente quiere realizar un actualización, normalmente sólo actualiza ciertas partes del modelo, y no el modelo completo.



Transfer Object Assembler

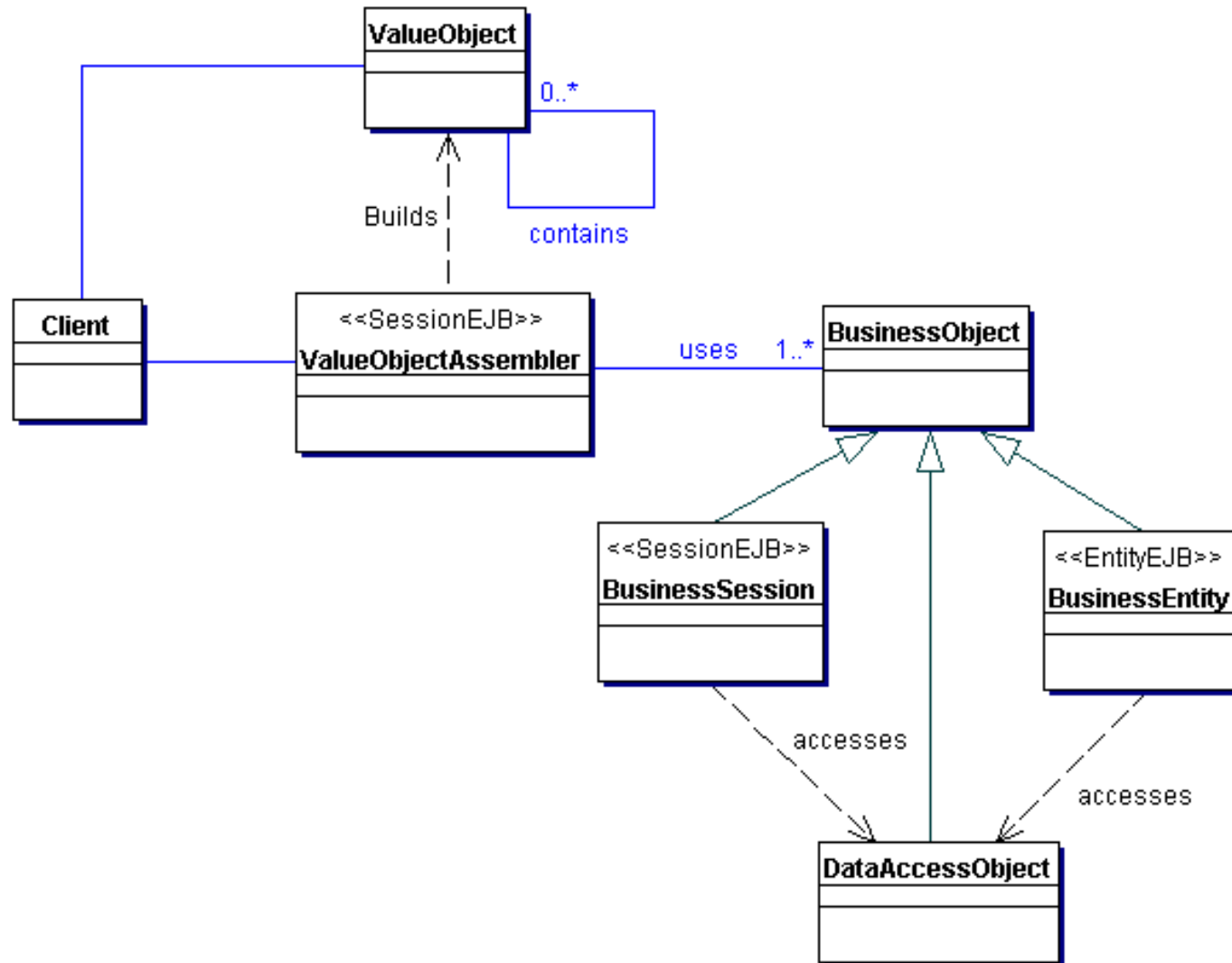
■ Solución

- Utilizar un Transfer Object Assembler para construir el modelo o submodelo requerido. El Transfer Object Assembler usa Transfer Objects para recuperar los datos de los distintos objetos de negocio y otros objetos que definen el modelo o una parte del modelo.



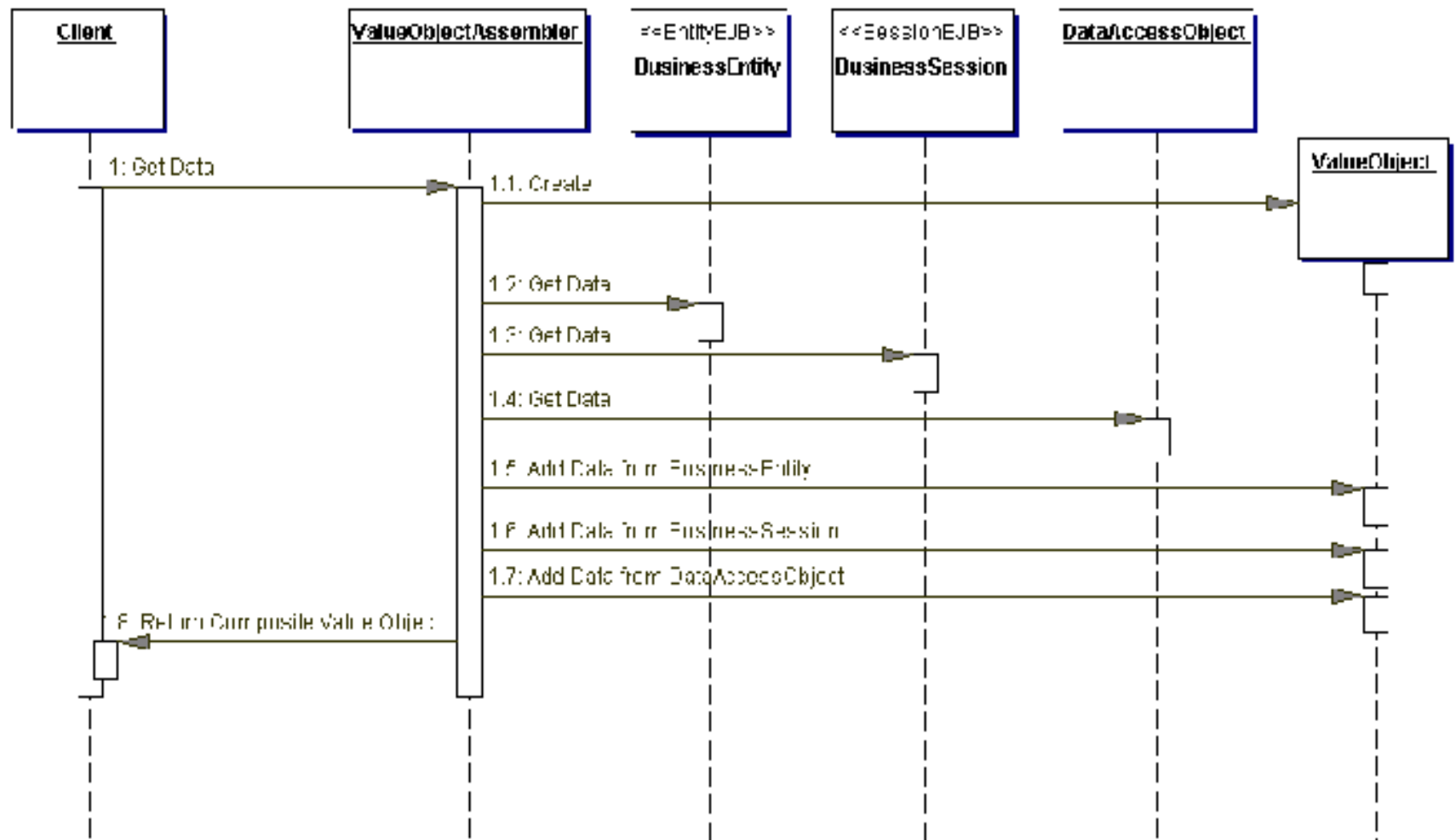


Transfer Object Assembler





Transfer Object Assembler





Transfer Object Assembler

■ Consecuencias

- Independiza la Lógica de Negocio
- Reduce el Acoplamiento entre los Clientes y el Modelo de la Aplicación
- Mejora el Rendimiento de Red
- Mejora el Rendimiento del Cliente
- Mejora el Rendimiento de la Transacción
- Podría Presentar Transfer Objects Obsoletos





Value List Handler

- Es un objeto que maneja la ejecución de consultas SQL, caché y procesamiento del resultado. Usualmente implementado como beans de sesión.
- Contexto
 - El cliente le pide al servicio una lista de ítems para su presentación. El número de ítems de la lista no se conoce y puede ser muy grande en muchas circunstancias.





Value List Handler

■ Problema

- La mayoría de las aplicaciones de la plataforma J2EE tienen un requerimiento de búsqueda y consulta para buscar y listar ciertos datos. En algunos casos, dichas operaciones de búsqueda y consulta podrían traer resultados que pueden ser bastante grandes. No es práctico devolver toda la hoja de resultados cuando los requerimientos del cliente son moverse por los resultados, en vez de procesar el conjunto completo.



Value List Handler

■ Causas

- La aplicación cliente necesita una facilidad de consulta eficiente para evitar tener que llamar al método `ejbFind()` de cada bean e invocar a cada objeto remoto devuelto.
- Se necesita un mecanismo de caché en la capa-servidor para servir a los clientes que no pueden recibir y procesar el conjunto de resultados completo.
- El cliente quiere moverse hacia adelante o hacia atrás dentro de la hoja de resultados.



Value List Handler

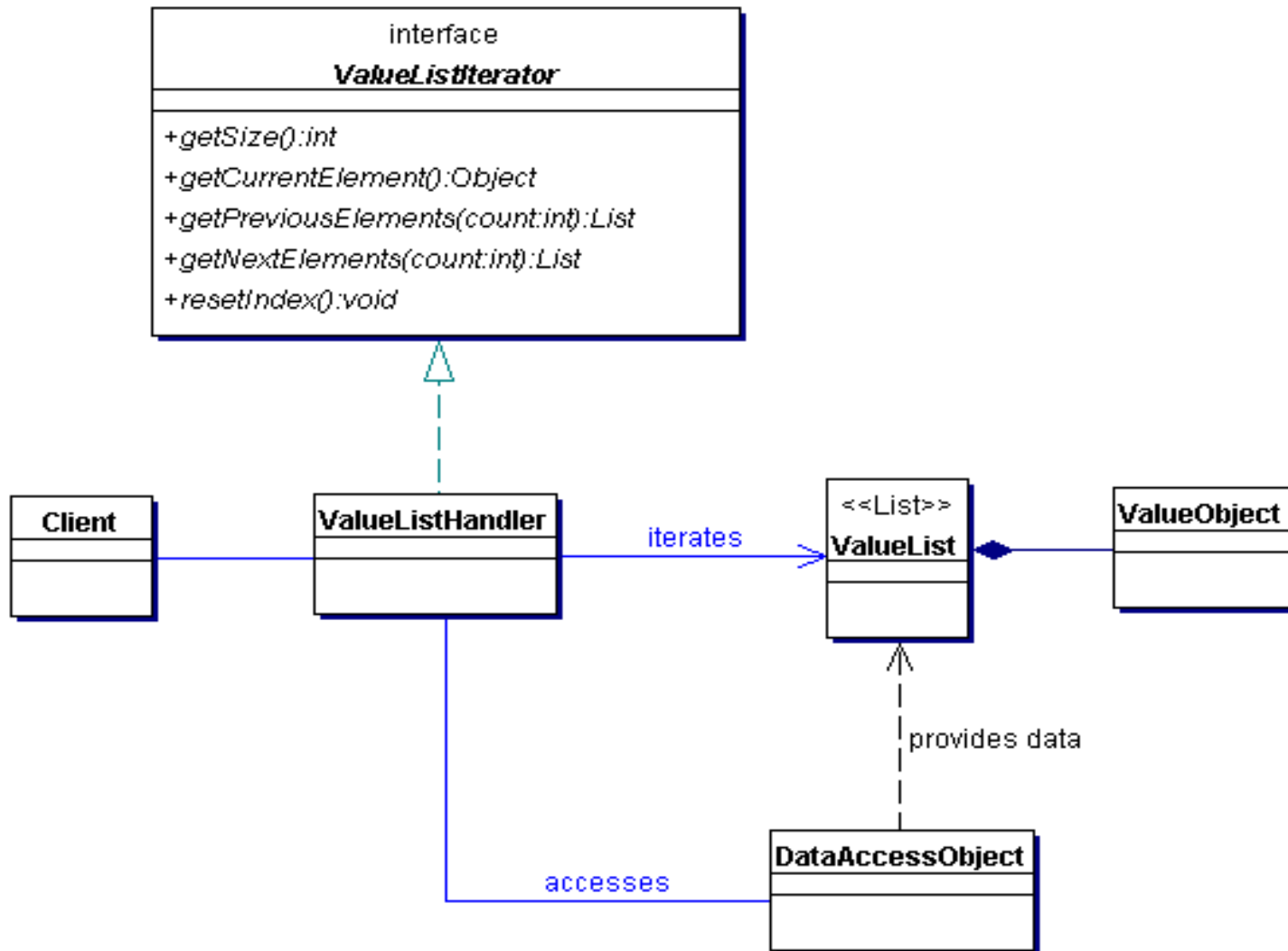
■ Solución

- Utilizar un Value List Handler para controlar la búsqueda, hacer un caché con los resultados, y proporcionar los resultados al cliente en una hoja de resultados cuyo tamaño y desplazamiento cumpla los requerimientos del cliente.



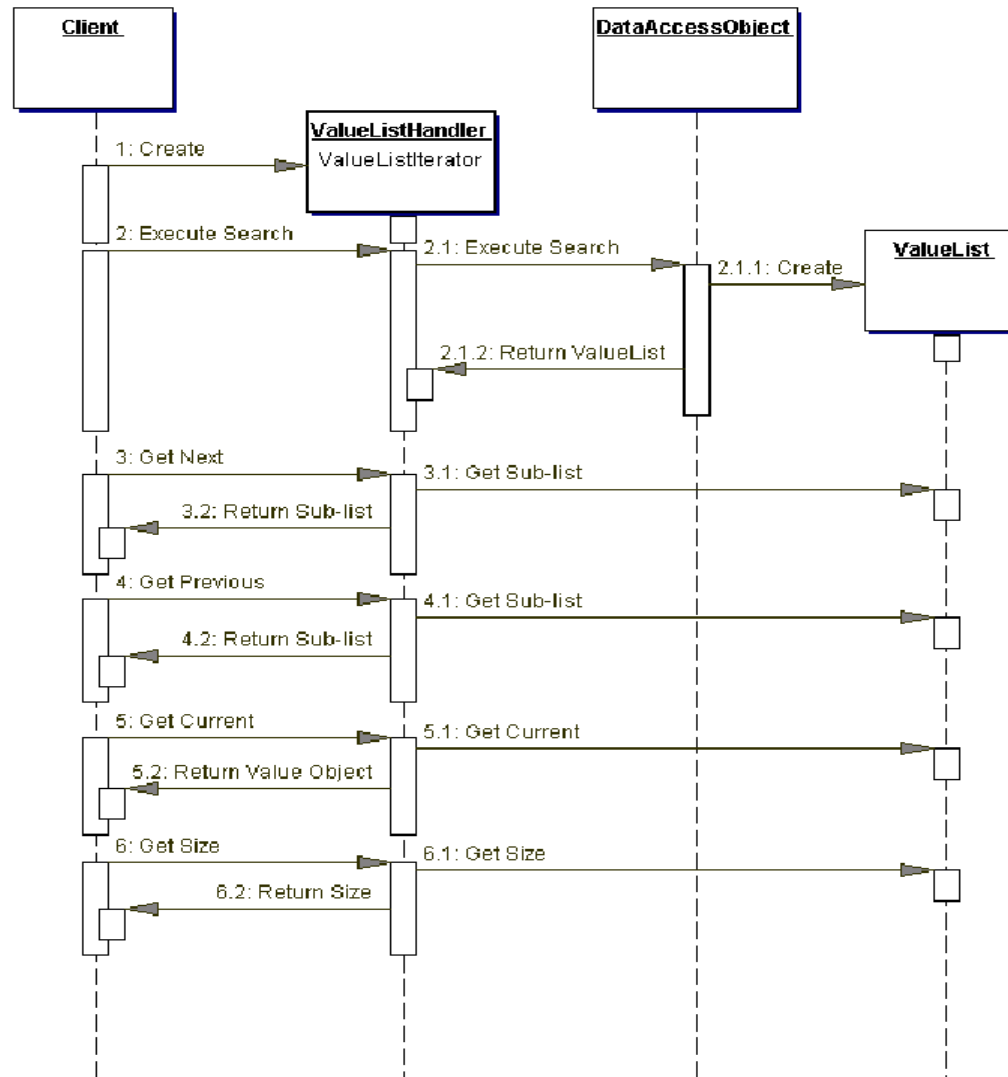


Value List Handler





Value List Handler





Value List Handler

■ Consecuencias

- Proporciona Alternativas a los métodos find() de EJB para Grandes Consultas
- Crea un Caché de Resultados de la Consulta en el Lado del Servidor
- Proporciona una Mayor Flexibilidad de Consulta
- Mejora el Rendimiento de Red
- Permite Atrasar las Transacciones del Bean de Entidad





Composite Entity

- Un bean entidad que es construido o es agregado a otros beans de entidad.
- Contexto
 - Los beans de entidad no se han pensado para representar todos los objetos persistentes del modelo. Los beans de entidad son mejores para objetos de negocio persistentes genéricos.





Composite Entity

■ Problemas

- En una aplicación de la plataforma J2EE, los clientes acceden a los beans de entidad mediante sus interfaces remotos. Cuando los beans enterprise son objetos específicos, los clientes tienden a invocar los métodos del bean de forma más individual, resultando en una gran sobrecarga en la red.





Composite Entity

■ Causas

- Los beans de entidad son mejores para implementar objetos genéricos. Todo bean de entidad se implementa utilizando muchos objetos, como el objeto home, el objetoremote, la implementación del bean, y la clave primaria, y todos son controlados por los servicios del contenedor.
- ...





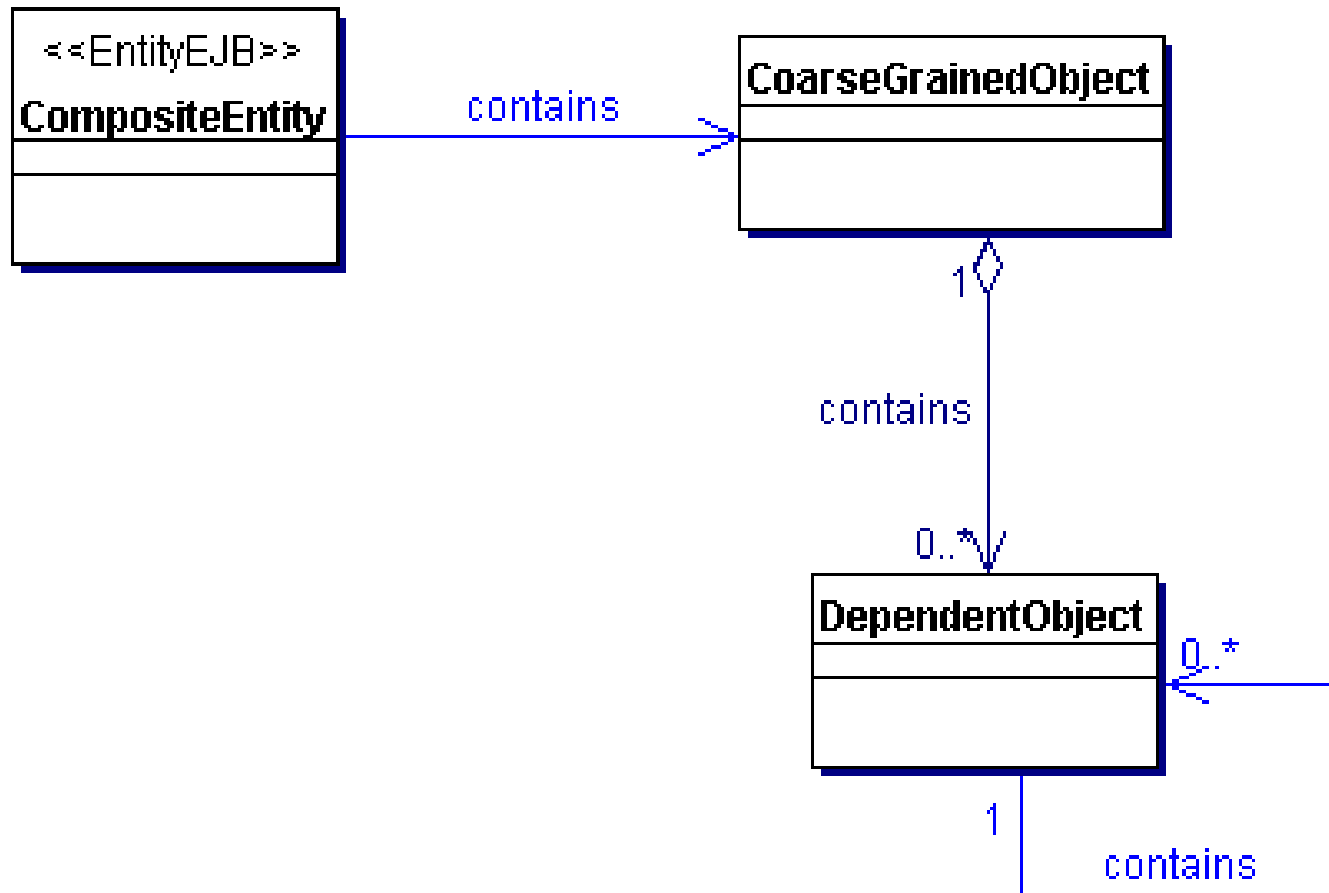
Composite Entity

■ Solución

- Utilizar Composite Entity para modelar, representar y manejar un conjunto de objetos persistentes relacionados en vez de representarlos como beans de entidad específicos individuales. Un bean Composite Entity representa un grupo de objetos.

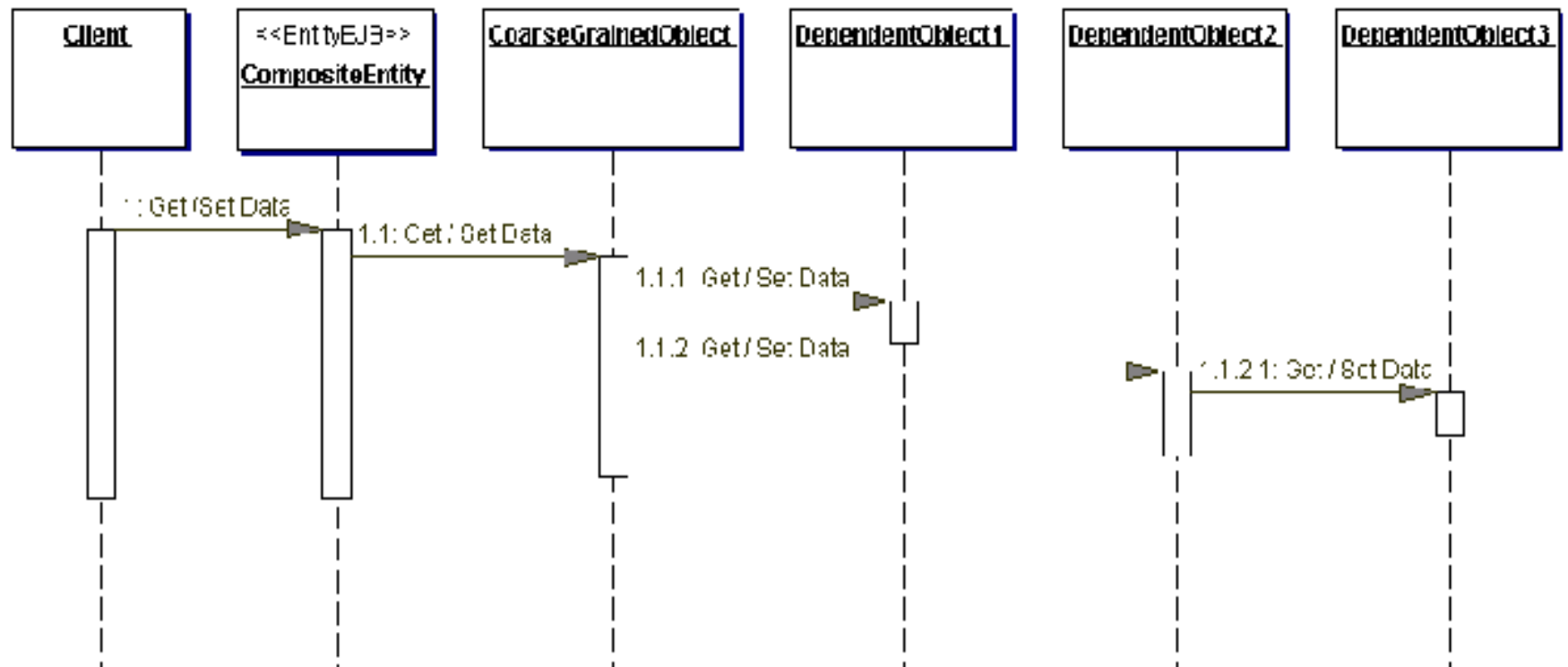


Composite Entity





Composite Entity





Composite Entity

■ Consecuencias

- Elimina las Relaciones Inter-Entidades
- Mejora la Manejabilidad Reduciendo los Beans de Entidad
- Mejora el Rendimiento de Red
- Reduce la Dependencia del Esquema de la Base de Datos
- Incrementa la Generalidad del Objeto
- Facilita la Creación de Transfer Object Compuestos
- Sobrecarga de Grupos de Objetos Dependientes Multi-Nivel



Capa de integracion





Data Access Object

- Consiste en utilizar un objeto de acceso a datos para abstraer y encapsular todos los accesos a la fuente de datos. El DAO maneja la conexión con la fuente de datos para obtener y almacenar datos.





Data Access Object

■ Contexto

- El acceso a los datos varía dependiendo de la fuente de los datos. El acceso al almacenamiento persistente, como una base de datos, varía en gran medida dependiendo del tipo de almacenamiento (bases de datos relacionales, bases de datos orientadas a objetos, ficheros planos, etc.) y de la implementación del vendedor.





Data Access Object

■ Problema

- La poca homogeneidad del almacenamiento persistente produce distintas implementaciones para lograr la accesibilidad . Un cambio en el sistema de almacenamiento puede conducir a una reimplentación de los componentes de datos y componentes vinculados.
- El acceso a datos incurre en la utilización de un API no propietario, por lo que se aumenta la dependencia entre el código de la aplicación y el código del acceso a datos.



Data Access Object

- Con el API de JDBC se puede acceder de una forma estándar a los servidores de bases de datos relacional. Sin embargo, la implementación del lenguaje SQL puede variar según el propietario, aunque se suponga que es un estándar.
- Introducir el código de conectividad en todos los componentes que requieren de acceso hace difícil la mantención y la migración cuando se desea cambiar la fuente de datos.



Data Access Object

■ Causas

- Los componentes de la aplicación necesitan recuperar y almacenar información desde almacenamientos persistentes y otras fuentes de datos variadas. Además, puede variar los proveedores, por lo que se tiene el problema del SQL propietario.
- Una API de datos que exponga al cliente un conjunto de operaciones comunes lograría reducir el acomplamiento y dependencias de la implementación.



Data Access Object

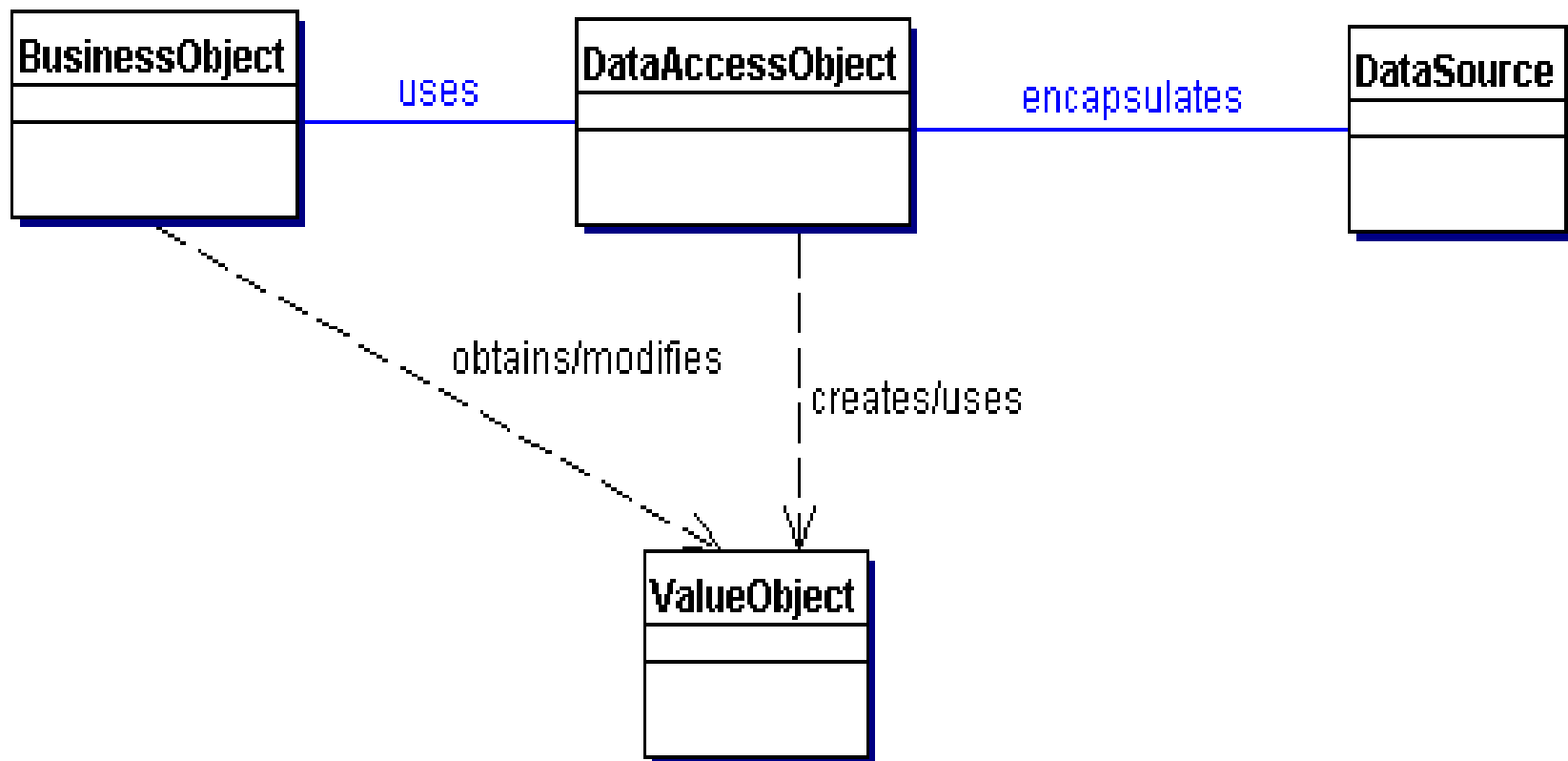
■ Solución

- Utilizar un Data Access Object (DAO) para abstraer y encapsular todos los accesos a la fuente de datos. El DAO maneja la conexión con la fuente de datos para obtener y almacenar datos.



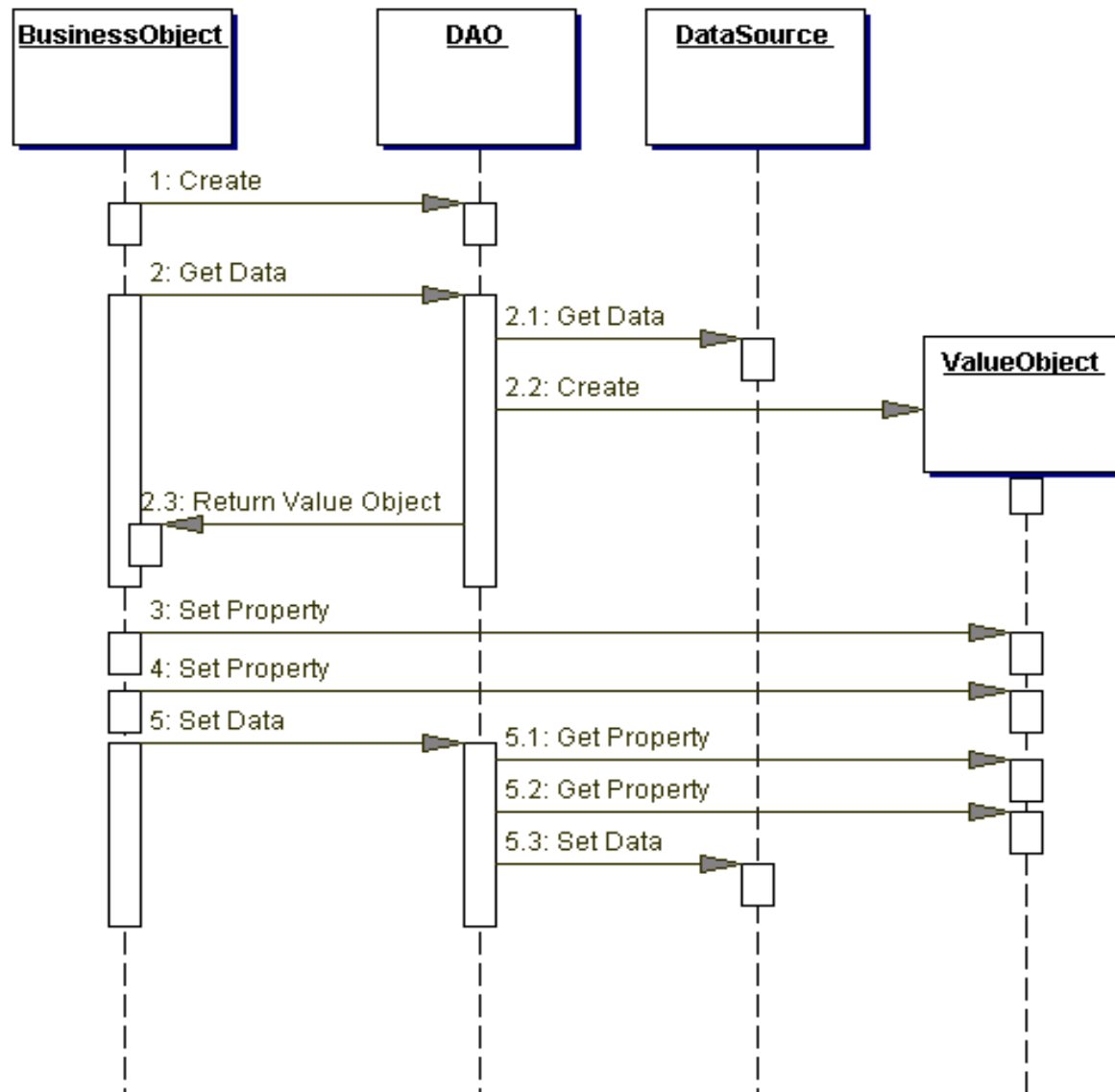


Data Access Object





Data Access Object





Data Access Object

■ Consecuencias

- Permite la Transparencia
- Permite una Migración más Fácil
- Reduce la Complejidad del Código de los Objetos de Negocio
- Centraliza Todos los Accesos a Datos en un Capa Indenpendinte
- No es útil para Persistencia Manejada por el Contenedor
- Añade una Capa Extra (capa de objetos)
- Necesita Diseñar un Árbol de Clases



Service Activator

- Se utiliza para recibir peticiones y mensajes asíncronos de los clientes. Cuando se recibe un mensaje, el Service Activator localiza e invoca a los métodos de los componentes de negocio necesarios para cumplir la petición de forma asíncrona.
- Contexto
 - Los beans enterprise y otros servicios de negocio necesitan una forma de activarse asíncronamente.



Service Activator

■ Problema

- Un cliente necesita acceder a un EJB, primero busca el objeto home del bean. Luego le pide a ese objeto home que le proporcione una referencia remota al bean enterprise requerido. Entonces el cliente invoca a los métodos de negocio sobre la referencia remota para acceder a los servicios del bean enterprise. Estas llamadas a métodos son síncronos. El cliente tiene que esperar hasta que esos metodos retornan.



Service Activator

■ Causas

- Los beans enterprise se exponen a sus clientes mediante sus interfaces remotos, que sólo permiten acceso síncrono.
- El contenedor maneja los EJB, permitiendo interacciones sólo mediante referencias remotas.





Service Activator

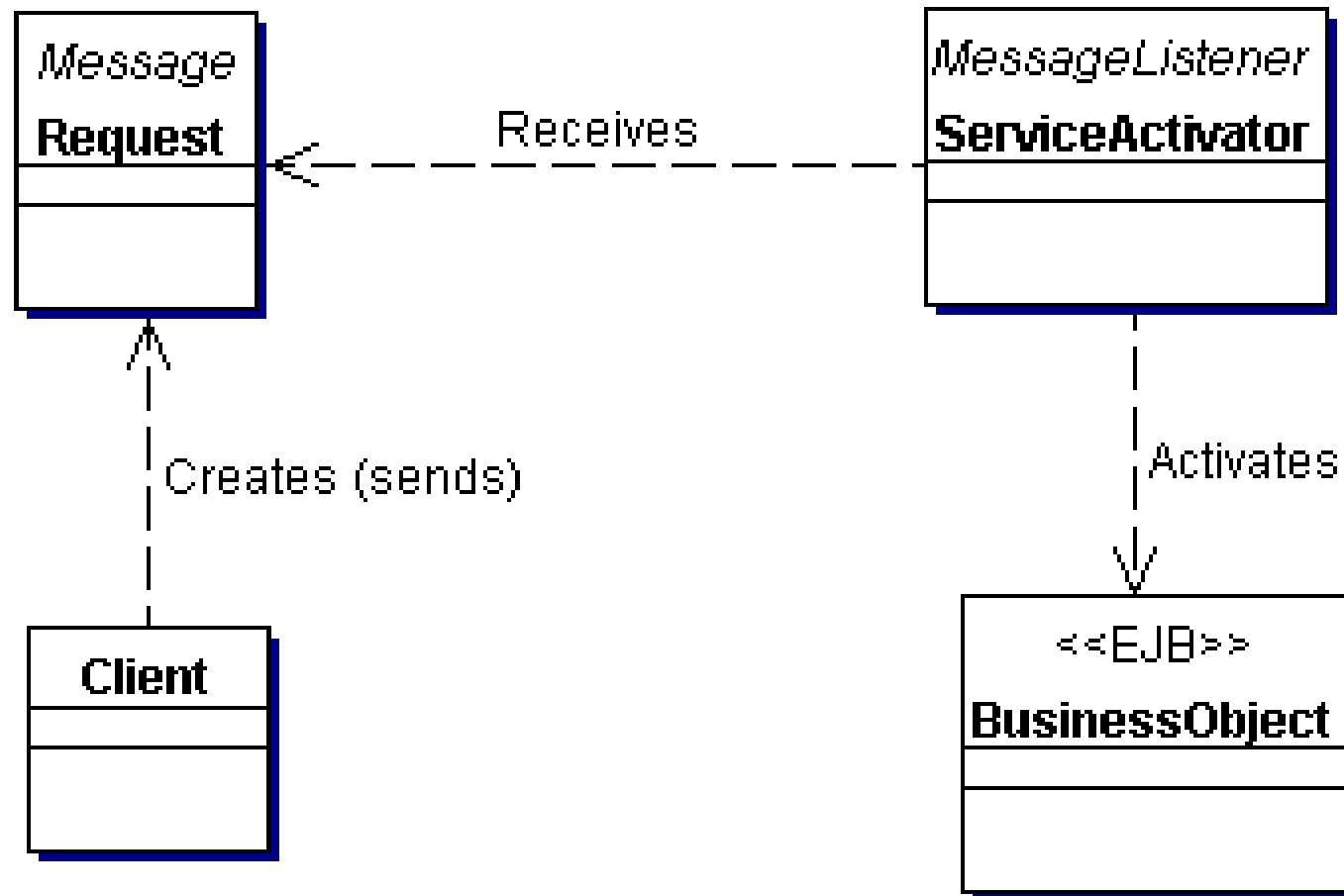
■ Solución

- Utilizar un Service Activator para recibir peticiones y mensajes asíncronos de los clientes. Cuando se recibe un mensaje, el Service Activator localiza e invoca a los métodos de los componentes de negocio necesarios para cumplir la petición de forma asíncrona.



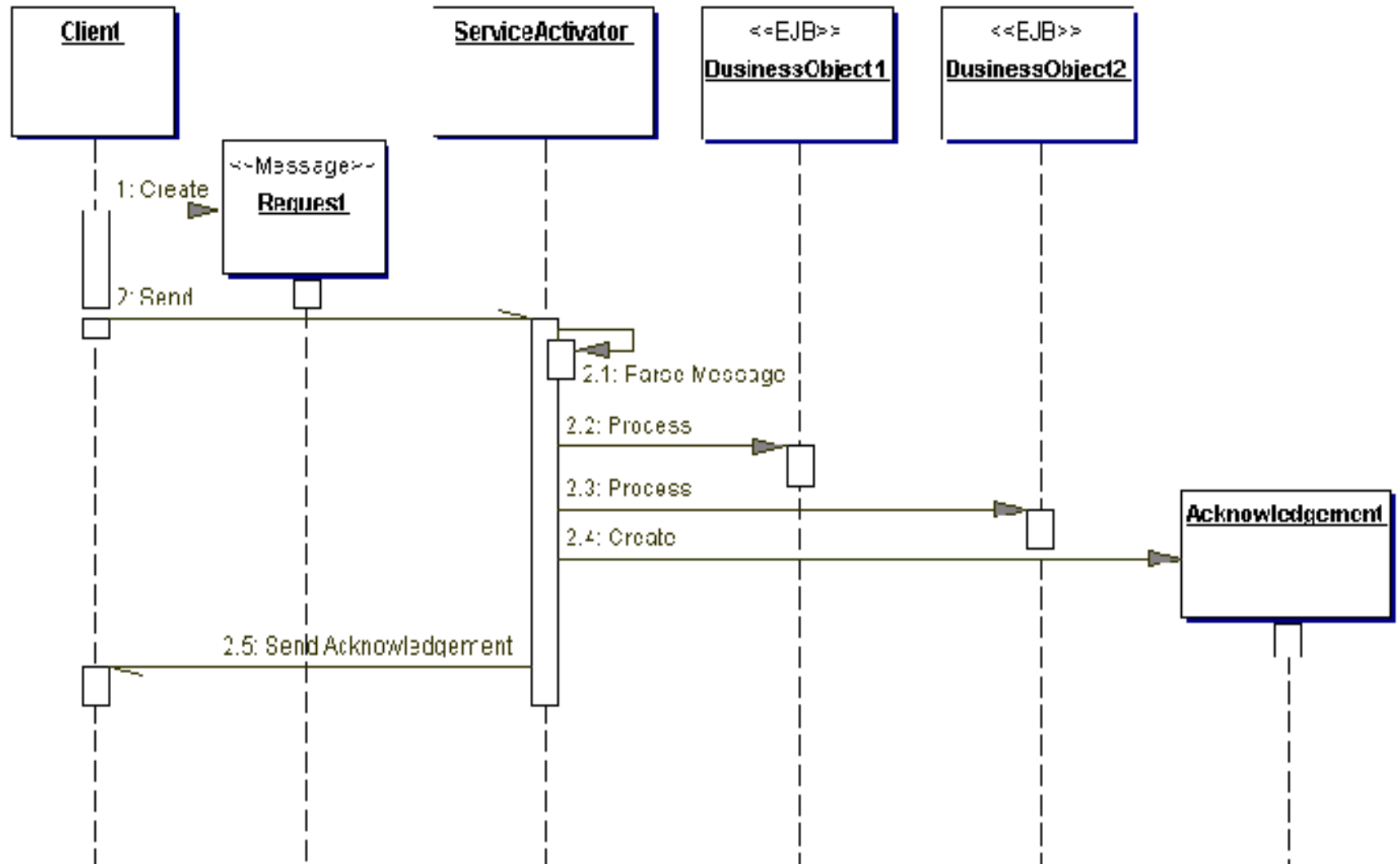


Service Activator





Service Activator





Service Activator

- Consecuencias
 - Integra JMS en implementaciones Pre-EJB 2.0
 - Proporciona Procesamiento Asíncrono para cualquier Bean Enterprise
 - Proceso Independiente

