Modelado de Software

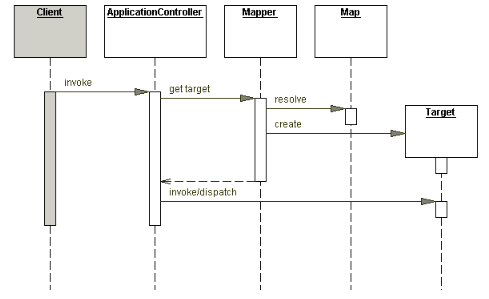
Hoja 1

1. **¿Cuál es la principal ventaja del controlador de aplicación?**

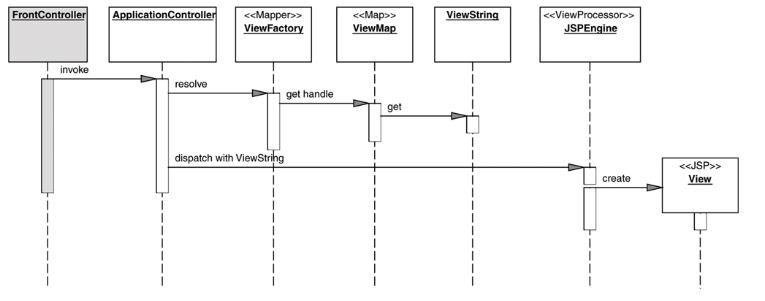
Que siempre es el mismo, no ha cambiado de la primera versión a la segunda del proyecto.

1. **Proporciona un diagrama de secuencia para alguno de los dos mappers del controlador de aplicación.**

**Factoria de comandos**



**Dispatcher**



1. **¿Cuál es la motivación del patrón contexto? ¿A qué se ha visto reducido en vuestros proyectos y por qué?**

Eliminar el acoplamiento que se genera al compartir los datos entre capas y que están fuera de su contexto. A un transfer abstracto ya que los contextos los genera el actionListener sin que haya una factoría de contextos.

1. **¿De todos los componentes del patrón service to worker cuál no utilizaste en vuestros proyectos y por qué?**

El FrontController porque no tenemos control de peticiones, no lo necesitamos sin haber acceso remoto por tanto solo damos respuesta a la petición. Porque no es una aplicación web.

El ViewHelper podríamos o no haberlo utilizado.

1. **En una arquitectura multicapa, ¿quién es el componente responsable de iniciar y acabar las transacciones (con éxito o no)? ¿Por qué?**

El servicio de aplicación porque es el único que sabe cuando empieza y acaba el caso de uso. Es el único que sabe cuáles son las reglas de negocio.

1. **¿Cuáles son las dos principales diferencias entre un modelo de dominio representado con transfers y con objetos del negocio?**

Que los transfers no aplican reglas de negocio ni tienen carga dinámica en cambio los objetos de negocio sí. Hay relaciones entre los objetos de negocio (asociaciones) y los transfers no.

1. **¿Cuál es la diferencia en la representación de un modelo del dominio con transfers según Alur, Crupi y Malks y según Martin Fowler?**

Alur, Crupi y Malks cuando un transfer necesita información de otro la mete en otro objeto TOA (Transfer Object Assembler), que es más sistemática, mientras que la de Martin Fowler crea un puntero en uno de los transfers al otro, es decir, los relaciona directamente pero es más complicado de programar.

1. **¿Cuál es la diferencia entre una transacción de negocio y una de sistema? Pon un ejemplo inspirado en tu proyecto.**

La transacción de negocio es la que conlleva un caso de uso.

La de sistema es contra el recurso transaccional.

Ejemplo: cuando voy y vuelvo varias veces de la interfaz a negocio, como en un modificar, lee el empleado a modificar y lo muestra, cambia los datos y ahora vuelve y lo modifica, esta segunda vez, el que la está usando piensa que solo existe una transacción, que es la de negocio total del caso de uso modificar pero esto se realiza con dos de sistema, una para buscar y otra para modificar. Las diferencias se ven cuando las transacciones son largas

1. **¿Cuál es la diferencia entre la gestión de la concurrencia pesimista y optimista? ¿Por un sencillo ejemplo (no hace falta utilizar UML, sino guiones del estilo utilizado en las transparencias de clase)?**

H1: leo C17, bloqueo,commit

H2: leo C17 no puedo

H1:

1. **¿Cuál es la diferencia entre la gestión de la concurrencia optimista normal y de incremento forzado (al estilo JPA)? ¿Por un sencillo ejemplo (no hace falta utilizar UML, sino guiones del estilo utilizado en las transparencias de clase)?**
2. **¿Qué cuatro problemas resuelve el patrón almacén del dominio?**

Concurrencia, carga dinámica, transaccionalidad y persistencia.

1. **En la adaptación de la gestión transaccional del almacén del dominio para usar con transfers, ¿cuál es la razón de ser del componente Transaction?**

Es para desacoplar (es un interfaz), ya que si le enchufamos la conexión directamente al Servicio de Aplicación estaríamos acoplando.

1. **¿Cuál es el tipo de retorno de Transaction::getResource()? ¿Por qué?**

Retorna Object, para desacoplar ya que si devuelves Connection estás acoplando.

1. **Describe los dos mecanismos fundamentales para la gestión de la concurrencia en la adaptación de la gestión transaccional del almacén del dominio para usar con transfers.**

Los transaction manager y usar SELECT FOR UPDATE (ya que para gestionar la concurrencia pesimista es necesario usarlos).

1. **¿En qué consiste, y que ventajas aporta, lo que coloquialmente hemos llamado en clase un DAO de Bilbao/empoderao? ¿Y uno de San Sebastián?**

Tener la capacidad de funcionar con o sin una transacción iniciada, si lo está coge su conexión y si no él mismo crea una conexión la abre y la cierra.

Cuando no encuentra una conexión la crea, porque en vez de crearla directamente lo hace a través del transaction manager.

1. **¿En qué se parecen y diferencian un intercepting filter de una cadena de responsabilidad?**

Son una red de objetos por donde va pasando una petición, se diferencia en que en la cadena de responsabilidad, solo un objeto de la cadena es responsable de tratarlo, mientras que en el intercepting filter todos los objetos de la cadena pueden tratarlo.

1. **¿Habéis utilizado vistas compuestas en vuestro proyecto? ¿Por qué?**

No, es algo muy propio de la web y con Swing no podemos hacerlo.

1. **¿Qué diferencia hay entre un service to worker y un dispatcher view?**

En el dispatcher view las vistas son capaces de invocar lógica de negocio y el controlador solo se queda para redirigir entre vistas, mientras que en el service to worker todo se centra en el controlador.

1. **¿Para qué sirve un manejador de lista de valores?**

El manejador de lista de valores permite resolver el problema de enviar a un cliente remoto una búsqueda con muchos valores, estos valores almacenados en capa de negocio y la lectura no requiere una transacción.

1. **¿Qué tienen en común una fachada de sesión, un agente de servicio web y un activador de servicio?**

Son mecanismos de exposición de lógica remota (Servicios de Aplicación remotos).

1. **¿Qué relación tienen un localizador de servicio y un delegado del negocio?**

El localizador de servicio es el mecanismo que usa el delegado de negocio para acceder a un componente de lógica remota.

1. **¿Qué relación tiene un delegado del negocio con una fachada de sesión/agente de servicio web/activador de servicio?**

El mecanismo que utiliza el cliente que invoca de forma remota a la lógica de negocio, ya sea bien expuesta por una fachada de sesión, un agente de servicio web o un activador de servicio.

1. Hay un entity manager por hebra, porque actúa como una caché.
2. **¿Cómo puede contribuir una unidad de persistencia JPA a la hora de evitar problemas de modificación de datos por parte de la capa de presentación en un dispatcher view?**

En presentación le doy un usuario de la base de datos de una unidad de persistencia que es de solo lectura.

**PREGUNTA SOBRE POLIMORFISMO EN EXAMEN.**

Hay un requisito que se llama función polimórfica y debe haber polimorfismo en el proyecto.

**Gestión concurrencia:**

En JPA es optimista, mirando los números de versión tras los commit.

Sencillo ejemplo

H1: leer C17, cambiar nombre, commit

H2: leer C17, cambiar teléfono, commit X, (esta no funciona porque la otra ha hecho commit antes).

H1: leer C17(bloqueo optimista), C17 no debería modificarse, commit X

H2: leer C17, cambiar teléfono, commit,

H1: leer C17, ¿está activo? Sí, …… (no se modifica), commit

H2: leer C17, da de baja, commit. Esto no falla porque la H1 no modifica el cliente solo lo lee, para que pete hay que bloquear (optimista) H1.

Si en la relación empleado departamento estoy calculando en una hebra la nómina de un departamento y en otra hebra doy de alta un empleado en ese departamento, tengo que bloquear en esta al departamento con bloqueo de incremento forzado, así lo marca como modificado. También hay que bloquear en la primera hebra (la que calcula la nómina) el departamento con bloqueo optimista.

En la pesimista puede haber interbloqueos, eso lo gestiona el SGBDR y echa para atrás las dos hebras o una de ellas.