UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Título: Una Solución para el Soporte de Conversaciones en Aplicaciones Web Java
Autores: Gabriel Alfredo Belingueres
Directores: Lic. Javier Díaz, Lic. Claudia Queiruga
Codirector: N/A
Asesor profesional: N/A
Carrera: Licenciatura en Informática Plan 90

Resumen

Se investiga la utilización de las Conversaciones como abstracción para simplificar el desarrollo de aplicaciones Web programadas en Java, así también para mejorar la calidad de las mismas. Se propone e implementa una solución utilizando el Framework Struts 2, así como una aplicación que demuestre su uso.

Palabras Claves

Java, Aplicaciones Web, Conversaciones, Struts 2

Conclusiones

Las conversaciones proporcionan una valiosa abstracción a la caja de herramientas del desarrollador de aplicaciones Web, la cual no solo mejora la calidad de los sistemas, sino que al mismo tiempo genera soluciones más sencillas y legibles, y por lo tanto más fáciles de mantener.

Trabajos Realizados

Desarrollo de un plugin de Struts 2 que soporta el desarrollo de aplicaciones Web con soporte de Conversaciones, así como otras facilidades como el uso de biyecciones.
Se implementó un sistema de reserva de habitaciones de hotel a efectos de demostrar el uso del plugin y evaluar tanto las mejoras de calidad como la sencillez de la solución obtenida.

Trabajos Futuros

Incorporar el uso de un modelo extendido de conversaciones, como pueden ser las conversaciones anidadas, o bien las continuations, y evaluar su impacto en la usabilidad del sistema. Estudiar rigurosamente las mejoras en la productividad y calidad generadas por el uso de conversaciones. Soportar el manejo de Persistencia de objetos utlizando otros frameworks de Mapeo Objeto Relacional como por ej. Hibernate. Incorporar la integración transparente de un motor de workflow como un nuevo scope, entre otras.

Fecha de la presentación: NOVIEMBRE 2012

Índice general

Índice de cuadros

Índice de figuras

Índice de listados

Resumen

El navegador Web es la herramienta de acceso universal para operar en la WWW, que históricamente ha ofrecido ciertas funcionalidades muy convenientes cuando el recurso a acceder es de naturaleza estática, pero poco conveniente cuando es de naturaleza dinámica como el contenido generado por una aplicación Web. Por ejemplo el acceso a un recurso desde un enlace almacenado en los Favoritos, presionar el botón de Refresco para recargar la página mostrada, Retroceder y Avanzar en el historial de navegación, realizar múltiples envíos de un formulario o abrir múltiples ventanas o pestañas de navegación. Sin una programación y pruebas cuidadosamente elaboradas, las aplicaciones web sufren de algunos problemas de calidad endémicos relacionados al uso que el usuario de la aplicación hace de ella, especialmente cuando utiliza alguna funcionalidad del navegador Web descrita previamente, ya que se desincroniza el estado de la aplicación con respecto al que el usuario está viendo en pantalla.

El estándar de Servlets provee un número limitado de *scopes* (en forma de una tabla de hashing, almacenada en la petición HTTP o en la sesión de usuario, entre otros) puestos a disposición del desarrollador para mantener el estado de la aplicación, sin embargo estos no reflejan fielmente las necesidades de alcance y tiempo de vida dictados por ciertos elementos que componen el estado de la aplicación, entonces el desarrollador debe escribir código especialmente para adaptarlo a sus necesidades. El problema endémico es que muchas veces este código es engorroso de escribir o bien muchas veces no es incluido. En consecuencia, es muy deseable que el desarrollador pueda acceder a un *scope* más adecuado, uno que pueda sobrevivir entre múltiples peticiones HTTP y aún así tener un alcance y tiempo de vida más efímero que la sesión de usuario: esta abstracción es lo que se conoce como una *conversación*.

Se estudian y comparan dos de los *frameworks* de desarrollo de aplicaciones Web con soporte nativo de conversaciones más utilizados: JBoss Seam y Spring Web Flow. Ambos ofrecen una solución de conversaciones que extienden el modelo básico y cubren un amplio espectro de necesidades; el primero implementa conversaciones anidadas mientras que el segundo un modelo basado en *continuations*.

Por otro lado, uno de los frameworks de desarrollo de aplicaciones Web más conocidos y utilizados, Struts 2, no posee soporte de conversaciones, un vacío que esta Tesina intenta llenar por medio de la implementación de un *plugin* que extiende las capacidades de Struts 2 para soportar conversaciones de una manera lo más transparente posible, y para ejemplificar su uso se desarrolla una aplicación Web de reserva de habitaciones de hoteles.

A causa del uso de conversaciones, esta aplicación presenta mayor robustez ante ciertas operaciones realizadas por el usuario, haciendo la aplicación más predecible en su comportamiento y redundando en una mejora en la calidad del producto. Asimismo, ya que se elimina código escrito por el desarrollador para tratar casos especiales, la programación se simplifica, lo que implica una mayor productividad y reducción de los tiempos de desarrollo.

Agradecimientos

Muchos eventos tuvieron que haber sucedido para poder completar este trabajo. No los he enumerado a todos aunque intento recordarlos, y de los que recuerdo algunos fueron más importantes que otros. Pero no son estos eventos lo que finalmente importan sino la gente que he conocido, con la que he trabajado, o con las que he hablado. Nunca podría haber terminado este trabajo sin ellos, la guía de mis directores, la ayuda de mis amigos y el apoyo de mi familia y mi esposa.

Quiero expresar mi más profunda y sincera gratitud a mi director y co-directora, Lic. Javier Díaz y Lic. Claudia Queiruga, respectivamente, por darme una segunda oportunidad al seleccionar este tema de investigación, y por la paciencia que han tenido en las entregas parciales que he realizado a lo largo de todo este tiempo. Luego de haber comenzado (y abandonado algunos años después) un tema de investigación (el desarrollo de juegos multi-jugador, co-dirigido por Lic. Miguel Luengo, a quien agradezco por las veces que me ha resuelto consultas del tema) se requiere mucha confianza en el alumno para apoyarlo en una nueva tesina. Sin duda no solo han tenido una gran confianza, sino que me la han sabido transmitir a mi (cosa que no es para nada menor).

A la gente que trabaja en la Facultad de Informática, especialmente la oficina de Alumnos, no solo por la calidad de atención sino por la atención personalizada, llamándome por teléfono varias veces para que vaya a la Facultad a completar algún trámite, o bien respondiendo mis consultas vía correo electrónico.

A la Facultad en general, a aquella época, una de las mejores de mi vida, y todos los amigos y compañeros que he cosechado durante todos estos años, tanto cuando cursaba en el C.U.R.J. (Centro Universitario Regional Junín) como cuando me mudé a La Plata a continuar los últimos años de la carrera.

A mis viejos, que me han bancado y apoyado (y todavía continúan haciéndolo) mis estudios, decisiones y por qué no mis locuras también. Mis hermanas Mara y Diana, por ser siempre ejemplos a seguir.

Finalmente, mi mayor agradecimiento y reconocimiento a mi amada esposa Daniela por su infinita paciencia, amor y comprensión por los largos días y horas encerrado frente a una computadora, y a mis hijas Martina y Marisel, las verdaderas razones por las que hoy estoy escribiendo estas líneas, por todos los momentos que hemos vivido, por los momentos que vendrán, y por todas las horas de juego robadas que a partir de ahora intentaré recuperar.

Capítulo 1

Introducción

Originalmente, la World Wide Web (o como comúnmente se la denomina WWW) fue pensada para compartir documentos científicos entre personas alrededor del mundo mediante *hipertexto*, es decir documentos enlazados unos con otros mediante *hipervínculos* en el formato HTML[1] de manera de que se pudiese acceder fácilmente a otro documento tan solo presionando un botón del ratón mediante un programa visualizador de documentos (el *navegador web*, o *web browser* en inglés). El protocolo de transporte subyacente que permitía este tipo de interacción era una versión preliminar de HTTP [10].

Poco tiempo después se comenzaron a utilizar los documentos HTML alojados en los servidores web para enviar información mediante formularios, lo que requería que el servidor ahora tuviese capacidades para generar código HTML en forma dinámica (en vez de leerlo desde un archivo guardado en el disco duro). Esto dio origen a las *aplicaciones web*, que en un principio eran muy sencillas, pero a medida que fueron madurando las tecnologías relacionadas a Internet las mismas fueron aumentando su complejidad.

Muchos años han pasado desde la invención de la WWW y del protocolo original HTTP, pero las características arquitecturales esenciales no han cambiado. La última versión de HTTP, la versión 1.1 [14] sigue siendo un protocolo *stateless*, por lo tanto, se han ido creado durante el transcurso del tiempo las técnicas más variadas para mantener el estado de la aplicación web entre página y página mostrada en el navegador, de las cuales las más conocidas se describirán en el Capítulo 2.

Sin embargo, a medida que las aplicaciones web van creciendo en su complejidad, también es necesario adaptar el modelo mental de pensamiento para desarrollarlas, ya que la arquitectura de la Web es stateless, mientras que la de las aplicaciones web son orientadas a procesos o tareas, esto es el usuario utiliza la aplicación para llevar a cabo algún trabajo o bien acciones que no son realizables con un único ciclo de request y response del servidor, sino que se requieren múltiples requests para realizar dicha tarea; en otras palabras las aplicaciones web necesitan gestionar su *estado conversacional* con el usuario.

Según el estándar que define la tecnología de Servlets [11] se define el objeto Sesión como abstracción para mantener el estado conversacional con el usuario. Efectivamente, cada request proveniente de la misma ventana de navegador será reconocida como del mismo usuario (o con más exactitud provenientes de la misma "sesión") y por lo tanto pueden mantenerse datos en el servidor sin necesidad de reenviarlos una y otra vez (a causa de la arquitectura stateless de HTTP).

Desafortunadamente la Sesión a veces resulta demasiado extensa en términos del alcance o tiempo de vida de los datos que se pueden almacenar en ella, ya que existen ocasiones en que se deben almacenar datos cuyo alcance o tiempo de vida no necesitan ser tan largos a los definidos por el objeto Sesión, de manera que hay una incongruencia semántica con la cual debe lidiar el desarrollador a la hora de programar la aplicación.

La presente Tesina se centra en el estudio de las *Conversaciones*, una abstracción muy útil para el desarrollador, que introduce un nuevo objeto en las aplicaciones web que permiten tener una granularidad más fina en el alcance y tiempo de vida de los objetos almacenados en Sesión, y aún así son mantenidos entre request y request. Esta abstracción es de gran importancia para simplificar el desarrollo de aplicaciones web a la vez que las hace más robustas y predecibles,

[1] Por supuesto, se trataba de una versión preliminar del conocido formato HTML. Vea [39] para conocer la historia de este popular formato de archivos.

redundando en mejor calidad del producto de software y reducción de tiempos de desarrollos.

El informe está organizado de la siguiente manera: El Capítulo 2 describe las motivaciones que originaron la presente Tesina e introduce al lector en los desafíos que presenta el desarrollador a la hora de programar aplicaciones web. El Capítulo 3 define la abstracción Conversación y se presentan ejemplos de su uso. En los Capítulo 4 y 5 se describen dos frameworks de desarrollo de aplicaciones web de amplio uso y que ya tienen incorporado el soporte de conversaciones: JBoss Seam [24] y Spring Web Flow [36], respectivamente. En el Capítulo 6 se propone una solución e implementación de conversaciones para Struts 2 [5], otro framework de desarrollo de aplicaciones web muy difundido. El Capítulo 7 describe una aplicación de ejemplo utilizando el soporte de conversaciones propuesto para Struts 2, y finalmente el Capítulo 8 presenta los resultados y conclusiones derivadas del presente trabajo de grado así como propuestas para posibles trabajos futuros que se pueden realizar continuando este tema de investigación.

Capítulo 2

Motivación

Como se describió en la Introducción, cuando la WWW comenzó a popularizarse, los diseñadores y desarrolladores se dieron cuenta de que sería una herramienta mucho más útil si pudiese ser interactiva, es decir, no solo utilizarla para mostrar contenido almacenado en servidores HTTP, sino que permitiese tener interacción con el usuario. Este nuevo requerimiento requirió varios cambios en los estándares HTML y HTTP:

- Se extendió HTTP. Además del comando `GET`, también se incorporó el comando `POST` para poder enviar información al servidor, mientras que los comandos `PUT` y `DELETE` permiten agregar y eliminar contenido al servidor, respectivamente.
- Se agregaron elementos interactivos a las páginas HTML con el tag `<FORM>` y otros tags de ingreso de datos como `INPUT`, `SELECT`, `BUTTON` o `TEXTAREA`, que permiten ingresar datos en un formulario, de forma similar a como se realiza en una aplicación de escritorio.
- Utilizando tecnologías como las *cookies* o re-escritura de URLs, se puede trabajar alrededor de la arquitectura *stateless* del servidor HTTP para seguir la pista a un usuario en particular a través de múltiples requests, de esta forma, las aplicaciones pueden ser personalizadas para cada usuario.

Todas éstas tecnologías combinadas dan nacimiento a lo que se llama una *aplicación Web* que liberan las tecnologías de la WWW. Las siguientes son algunas de las ventajas con respecto a las aplicaciones GUI de escritorio:

- No se requiere instalación en la PC del usuario, haciendo trivial el despliegue de la misma en múltiples estaciones de trabajo.
- No hay necesidad de actualizar la aplicación cuando ésta sufre alguna modificación: el usuario siempre utiliza la última versión de la misma en todo momento.
- Usuarios que ya están familiarizados con los navegadores web, páginas HTML e Internet no necesitan tanto entrenamiento en el uso de la aplicación.
- Admite nuevos modelos de venta de la aplicación, permitiendo generar canales de ventas alternativos a la venta directa del producto al usuario u organización.

por supuesto, también tienen algunas desventajas:

- Las aplicaciones Web a menudo no pueden ofrecer una experiencia interactiva tan fluida y rica como las aplicaciones GUI de escritorio[1], no obstante, de alguna forma también puede verse como una ventaja, ya que obliga a diseñar la aplicación de una forma simple.
- El soporte de sesiones HTTP hace que la escalabilidad de la aplicación sea más difícil de lograr, ya que es el servidor quien debe almacenar estas sesiones. Si la misma corre sobre un cluster de servidores, además deben establecerse mecanismos adicionales para asegurar un correcto balanceo de carga, failover y replicación de sesiones.

[1] Al menos hasta hace recientemente cuando comenzaron a popularizarse las RIA (*Rich Internet Applications*) basadas en tecnologías como Ajax, Flash, Flex o JavaFX.

2.1. Navegación controlada

La mayoría de las aplicaciones web no admiten la navegación libre por cualquier página por parte del usuario; deben permitirle llevar a cabo sus tareas, pero restringiendo las opciones de navegación para que la misma se produzca en una forma más controlada, de manera tal que el desarrollador pueda implementar una solución que sea predecible en su comportamiento, sea estable, segura y robusta.

Imaginemos que hoy en día se utilizan aplicaciones web para llevar a cabo tareas muy sensibles, como por ej:

Home Banking Usuarios desean ver sus estados de cuenta, realizar transferencias entre distintos bancos, pagar servicios e impuestos.

Reserva de pasajes aéreos La aplicación debe permitir al usuario seleccionar los vuelos de partida y destino, seleccionar preferencias de ubicación, utilizar sus millas de viajero frecuente, entrar información de pagos. Muchas agencias también permiten reservar un hotel y alquilar un automóvil.

Trámites municipales Muchos gobiernos y municipios utilizan aplicaciones para reserva de turnos, ej. para obtener el carnet de conducir, defenderse ante contravenciones (ej. multas por mal estacionamiento), emisión de libres deuda, etc.

Estas aplicaciones manejan datos muy sensibles del usuario, a veces información confidencial como bienes y dinero, por lo tanto es de suma importancia que la misma sea estable, segura y robusta. Los usuarios deberían sentirse cómodos interactuando con la aplicación, confiando en que ésta hará lo correcto aún en situaciones donde el usuario haya accidentalmente realizado una acción como presionar el botón de Recarga o Retroceso (conocidos en inglés como el *Refresh Button* y *Back Button*).

La mayoría de los frameworks de desarrollo web en Java, como Struts [4], Struts 2 [5] (y su predecesor WebWork [28]), Spring MVC [34] o JSF [30] permiten desarrollar aplicaciones web con cierta facilidad; utilizan la sesión HTTP para gestionar el estado asociado a un usuario en particular y hacen fácil trabajar con páginas que contienen formularios HTML utilizando técnicas como la ligadura automática de datos[2], sin embargo no proveen herramientas adicionales para satisfacer casos de uso con una navegación más complicada y por lo tanto los usuarios deben ser cuidadosamente guiados para completar ciertas tareas.

Esta es una deficiencia de estos frameworks, ya que implementar correctamente una navegación controlada en una aplicación web es un problema difícil de resolver. En principio hay dos temas claves que necesitan ser tenidos en cuenta para resolver este problema:

- Control navegacional.
- Gestión del estado.

2.1.1. Control navegacional

La WWW fue diseñada para permitir una navegación libre y sin restricciones, no obstante en una aplicación web: ¿cómo se controla la libertad de navegación del usuario? Los navegadores ofrecen listas de *bookmarks* de Favoritos, permitiendo a un usuario agendar una página en particular y saltar directamente a esa página de nuevo en el futuro. El listado de historial de páginas visitadas mantenidas en los navegadores, y los botones de Retroceso, Avance y Recarga hacen fácil para los usuarios ir hacia páginas que han visitado previamente. Un usuario también puede abrir múltiples ventanas o pestañas del navegador al mismo tiempo (para comparar información por ejemplo).

Uno podría pensar que esta funcionalidad es simplemente una conveniencia ofrecida por los navegadores modernos pero en realidad, el espíritu de la navegación libre está inmerso dentro de las especificaciones de las tecnologías esenciales de la WWW. El siguiente extracto está

[2] La técnica permite definir en una página HTML un campo de texto dentro de un formulario, por ej. `<input type="text"name="nombre"/>` y definir en una clase una variable que contenga al dato, por ej. `private String nombre` con sus métodos getter y setter; luego el framework automaticamente asigna el valor ingresado por el usuario en el form a una instancia de la clase con esa variable, quedando así disponible para su uso en la aplicación. Además, muchas veces el framework también hace la validación automática de los datos en el formulario.

copiado de la especificación de HTTP 1.1 [14] (sección 13.13 de la RFC 2616, que sugiere que los navegadores no deberían recargar una página desde el servidor cuando el usuario accede el historial de navegación, por ejemplo, usando el botón de Retroceso:

> User agents often have history mechanisms, such as "Back"buttons and history lists, which can be used to redisplay an entity retrieved earlier in a session.
>
> History mechanisms and caches are different. In particular history mechanisms SHOULD NOT try to show a semantically transparent view of the current state of a resource. Rather, a history mechanism is meant to show exactly what the user saw at the time when the resource was retrieved.
>
> By default, an expiration time does not apply to history mechanisms. If the entity is still in storage, a history mechanism SHOULD display it even if the entity has expired, unless the user has specifically configured the agent to refresh expired history documents.

Queda claro a partir de este extracto de la especificación que se sugiere el historial de navegación como una ayuda del lado del cliente, a modo de mejorar la experiencia del usuario al no tener que recargar el recurso nuevamente pero, si bien esto tiene mucho sentido para las páginas estáticas, para las aplicaciones web (donde una página puede cambiar su contenido dependiendo del momento en que se llame) resulta ser un problema.

Desafortunadamente no es generalmente posible inhabilitar el historial de un navegador, ni sus capacidades de salvar Favoritos[3], o su capacidad de abrir nuevas ventanas o pestañas. Por lo tanto, el desarrollador está forzado a lidiar con el problema: se deben manejar a priori las situaciones donde el usuario utiliza las conveniencias de una navegación libre cuando se mantiene una comunicación controlada con una aplicación web. Algunas de las situaciones a ser tenidas en cuenta son las siguientes:

- ¿Qué pasa cuando un usuario salva en sus Favoritos una página en la mitad de una secuencia de pasos para completar una tarea? No se puede detener al usuario de ingresarlo en sus Favoritos, pero cómo debería reaccionar la aplicación cuando el usuario utiliza esa URL favorita para saltar a esa página directamente?

- ¿Cómo maneja la aplicación los pedidos de refresco (botón Recargar)?

- ¿Cómo maneja la aplicación los pedidos de moverse hacia atrás o adelante en el historial de navegación?

- Un usuario puede accidentalmente enviar dos veces (o más) en forma reiterada los datos de un formulario que está completando ¿Cómo reacciona la aplicación ante la presencia de estos múltiples envíos del mismo formulario?

- Una situación cada vez más frecuente es que el usuario abra dos ventanas o pestañas del navegador sobre la misma aplicación, típicamente para comparar resultados, evaluar alternativas o trabajar sobre distintos conjuntos de datos. ¿Cómo trata la aplicación web con estas situaciones? Se necesita tener el cuidado de evitar interferencias o envíos duplicados.

Existe todo un abanico de soluciones para que estas situaciones no generen problemas, las cuales forman parte de la gestión de estado provisto por la aplicación.

2.1.2. Gestión del estado de la aplicación

La aplicación web debe incorporar controles y validaciones adicionales a las requeridas por la lógica de negocio para lidiar con los problemas generados por un uso inadecuado del navegador por parte del usuario. Las situaciones descritas arriba son causa común de errores, ya que cuando el usuario las realiza, esta introduciendo la posibilidad de que el estado actual del navegador y de la aplicación en el servidor quede desincronizado, es decir lo que ve el usuario en la pantalla no refleja el estado actual en el servidor.

[3] En inglés, *bookmarking*.

2.2. Problemas de gestión de estados y soluciones

A continuación se describen en detalle los problemas de gestión del estado de la aplicación junto a algunas de las soluciones más comunes.

2.2.1. Acceder a un proceso desde un enlace Favorito

Supongamos que la aplicación permite realizar alguna tarea mediante un *wizard*, es decir tiene implementada funcionalidad para completar cierta tarea mediante el llenado de formularios a través de una secuencia de páginas. Cada página contiene parte del estado de la aplicación, que va ingresando el usuario a medida que va recorriendo esa secuencia de páginas.

Supongamos (para dar un ejemplo concreto) que el wizard consta de tres páginas: La tarea comienza completando el formulario de la página 1, el usuario presiona un botón rotulado “Siguiente” y se muestra la página 2, la cual contiene un formulario, al presionar “Siguiente” se llega a la página 3 que también contiene un formulario con un botón rotulado “Salvar”. Al presionar este último botón sobre la página 3 se dispara un procesamiento en el servidor, que podría ser salvar en una base de datos la información recolectada en los 3 formularios.

El problema de acceder a un proceso desde un enlace Favorito se presenta cuando el usuario —mientras completa el formulario de la página 2— guarda la URL de la página actual en sus Favoritos del navegador. Acto seguido el usuario cierra el navegador y abre uno nuevo, comenzando así una nueva sesión de usuario con el servidor de aplicaciones, e inmediatamente selecciona esa URL favorita en el navegador, haciendo que el mismo envíe un GET request.

Lo que sucede en el servidor es que la aplicación presupone que el proceso debe comenzar en la página 1, no en la página 2, de modo que dicho request es claramente inválido desde el punto de vista funcional: El usuario no debería haber llegado a esa página sin antes haber pasado por la página 1.

Este problema de los Favoritos no solo se presenta al querer realizar tareas del tipo wizard. En efecto, un caso especial de este problema se produce cuando la aplicación requiere autenticar al usuario que la está operando. Para autenticar al usuario la aplicación típicamente solicita mediante una página de login que el mismo ingrese su usuario y contraseña, para luego chequear estas credenciales contra una base de datos y así poder determinar si se le da acceso. Si el usuario salva cualquier página de la aplicación en Favoritos, entonces en el futuro, cuando acceda a este Favorito, la aplicación debe poder detectar tal acceso y redirigir al usuario hacia la pantalla de login ya que el mismo no ha sido previamente autenticado.

Soluciones en el cliente

Se podría documentar o entrenar al usuario para que no grabe ni cargue Favoritos para utilizar este wizard, sin embargo no existiría ningún control por parte de la aplicación que valide que el usuario no lo haga, y en general no existen soluciones mágicas del lado del cliente, ya que no puede inhabilitarse la grabación y posterior carga del Favorito en el navegador web.

Soluciones en el servidor

Del lado del servidor la aplicación puede saber cuando recibe el request solicitando la segunda página del wizard si previamente el mismo cliente ha enviado la página anterior. Para esto comúnmente la aplicación crea y graba algún objeto cuando el cliente inicia el wizard en forma correcta, de modo que cuando el cliente solicita la siguiente página del wizard, el servidor puede chequear si ese objeto ya existe, y en tal caso responder correctamente con la página. Si el objeto no existe, entonces se ha detectado un request incorrecto y no debe ser tenido en cuenta: la acción más común en este caso es mostrar al usuario una página de error o enviarlo al menú principal de la aplicación. Cabe aclarar que esta solución podría pensarse como un caso especial del patrón Synchronizer token [1] que se verá en detalle en la Sección 2.2.4.

En el caso común de la autenticación del usuario mediante su usuario y contraseña, el objeto que se graba típicamente son los datos del usuario logueado, junto con los roles que tiene asignados, lo que lo autoriza a entrar (o no) en diferentes funcionalidades de la aplicación.

¿Dónde se graba el objeto? Típicamente las aplicaciones pueden grabarlo en la sesión del usuario, es decir en el objeto session (una instancia de una clase que implementa la interface

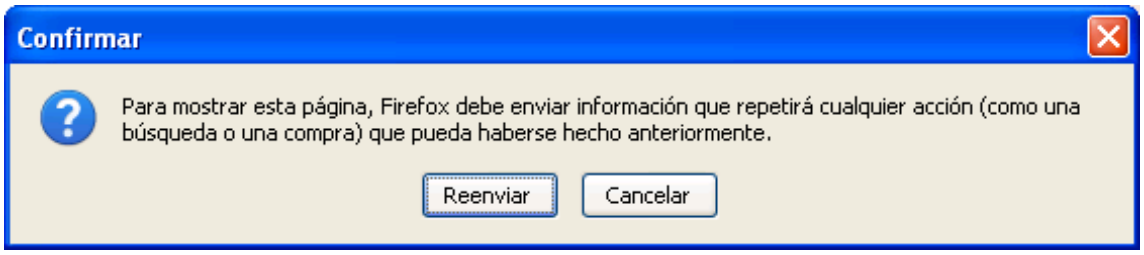

Figura 2.1: Ventana de advertencia de Refresco

`javax.servlet.http.HttpSession`, pero también podría ser en otro tipo de repositorio como una base de datos.

2.2.2. Pedido de refresco (botón Recargar)

Supongamos que estamos llenando un formulario para registrarnos como usuario de una aplicación web. Este formulario envía los datos suministrados usando el método `POST` de HTTP, ya que es la forma estándar de proveer datos a la aplicación. Entonces, habiendo completado el formulario y enviado los datos, la aplicación web recibe el request y salva los datos del nuevo usuario en una base de datos, y una vez salvados, muestra una página de confirmación con un mensaje de éxito que dice “Se le ha enviado a su casilla de correo un email de confirmación con el cual podrá activar su registro”.

Si por error el usuario presiona el botón Refresco, el navegador intentará repetir el último request generado, que fue el `POST` anterior con los datos del usuario. Como los datos son nuevamente enviados, el navegador *automáticamente* presenta una caja de dialogo al usuario informando que se reenviarán los datos anteriores (Figura 2.1). Si el usuario presiona “Cancelar” se cancela el refresco, y si presiona “Aceptar” se reenvía el mismo request anterior.

Aquí se pueden ver dos problemas:

- Por un lado está la confusa caja de dialogo confirmando el Refresco, la cual puede generar cierta desconfianza o duda al usuario que está operando el sistema, ya que puede suceder que no sepa interpretar lo que el navegador está informando.
- Al aceptar el refresco, se está ejecutando nuevamente la funcionalidad en el servidor que salva el nuevo usuario en la base de datos, es decir un **doble envío** de los datos, pudiéndose producir errores confusos como que el nombre de usuario ya existe en la base de datos (de hecho, se insertó en el request anterior, pero la segunda vez que se intenta crear al nuevo usuario ese nombre de usuario ya está reservado).

Soluciones en el cliente

Teniendo en cuenta el hecho de que el botón de Refresco de un navegador no se puede inhabilitar, podrían sugerirse soluciones del lado del cliente pero resultan ser muy ingenuas o demasiado costosas.

La opción menos costosa pero la más vulnerable a errores es sencillamente documentar e informar al usuario que no utilice las ayudas de navegación, ya que la aplicación no se comportará debidamente en el caso contrario. Obviamente en aplicaciones críticas como un home banking esto es inaceptable.

Una opción sencilla pero muy simplista es abrir la aplicación web en una ventana de navegador sin la barra de menú ni de direcciones, lo cual efectivamente oculta los botones de Refresco, Retroceder y Avanzar en el historial de navegación, sin embargo todavía se puede hacer un refresco presionando la tecla F5 o Ctrl-F5 que cumplen la misma función del botón de Refresco.[4]

En ambientes controlados, donde el usuario está obligado a ejecutar la aplicación desde un navegador en particular (por ej. en una Intranet corporativa) se podría instalar en cada puesto

[4] Existen dos variaciones de refresco provistos por los navegadores: con F5 se hace el request con la condición de que si no se ha modificado en el servidor, se utilizará la copia en el cache; con Ctrl-F5 siempre se recupera nuevamente el recurso desde el servidor.

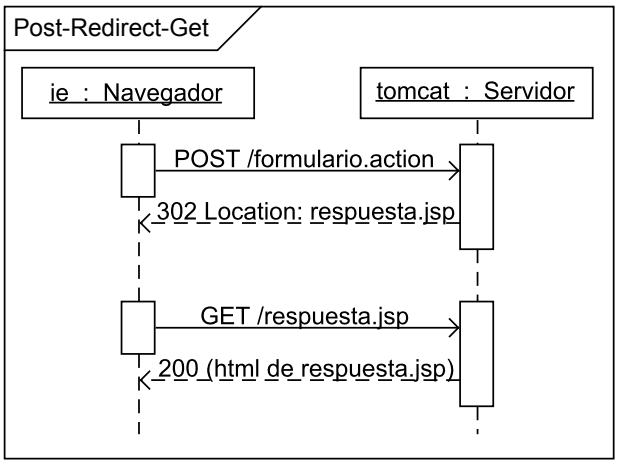

Figura 2.2: Patrón Post-Redirect-Get

de trabajo una versión personalizada de un navegador web donde los programadores del mismo inhabiliten todas las ayudas de navegación. No obstante, esta solución tiene un alto costo de mantenimiento. Por supuesto, ya no es una opción válida si el usuario puede acceder a la aplicación desde cualquier navegador web del mercado.

Soluciones en el servidor

Una solución elegante es utilizar el patrón de diseño web llamado POST-REDIRECT-GET [25], también conocido como “Redirect after post”.

Cuando una aplicación envía una página HTML en respuesta a un `POST` request, esta página se verá como cualquier otra, haciéndole pensar al usuario que puede hacer un Refresco de la misma sin consecuencias negativas. Lo que propone este patrón de diseño web es que, en vez de enviar la página directamente en respuesta a un `POST` request, la aplicación debería enviar un REDIRECT request[5] con la URL de la página HTML, de modo que una vez mostrada la página de resultados, al ser solicitada al servidor mediante un `GET` request, el Refresco de la misma no hará envío de los datos del formulario, y por lo tanto, se elimina el problema del doble envío descrito arriba (existen otras formas de realizar un envío duplicado, las cuales este patrón no soluciona). El diagrama de secuencia del patrón se muestra en la Figura 2.2.

2.2.3. Retroceder y Avanzar en el historial de navegación

Supongamos que hemos llenado un formulario para dar de alta un nuevo producto en un sistema de ventas, de modo que al enviar el formulario al servidor, se graba un nuevo producto en una base de datos y el servidor responde con una pantalla de confirmación indicando el éxito de la operación. Si se presiona el botón Retroceder, se verá la página con el formulario tal cual se veía antes de enviarla, lo cual puede hacer pensar al usuario distraído (incorrectamente) que el producto no se ha salvado todavía, haciendo que envíe nuevamente el formulario y por lo tanto se grabe nuevamente el mismo producto (o bien que el servidor retorne una pantalla indicando que el producto ya existe). En otras palabras, al presionar el botón de Retroceder, se pierde el sincronismo entre el estado actual del servidor con respecto a lo que ve el usuario en la pantalla.

Avanzar en el historial de navegación es un poco más benigno que retroceder, ya que el daño ya se ha hecho, es decir, sólo se puede avanzar en el historial de navegación si es que se ha retrocedido previamente: el daño ha sido causado por el uso del botón de Retroceso y por lo tanto se ha perdido el sincronismo con el estado del servidor. Decimos que es un poco más benigno porque presionar el botón de Avanzar en el historial hace que se muestre una pantalla por la que ya ha pasado la aplicación,*i.e.* se está más cerca de estar sincronizado. Si es la última página

[5] El REDIRECT enviado por un servidor hacia el navegador instruye al mismo a emitir inmediatamente un `GET` request sobre la URL asociada al redirect.

```
1 <%
2 response.setHeader("Cache-Control","no-cache"); // HTTP 1.1
3 response.setHeader("Pragma","no-cache"); // HTTP 1.0
4 response.setDateHeader("Expires", 0); //evita que se cachee en servidor proxy
5 %>
```

Listado 2.1: Encabezados para evitar que se almacene en cache

recibida entonces estaremos con el navegador sincronizado con el servidor; caso contrario estará desincronizado.

Soluciones en el cliente

Como se mencionó anteriormente los botones de Retroceder y Avanzar del navegador no pueden inhabilitarse, por lo cual las opciones para controlar el botón de Retroceso están basadas en el uso de Javascript o efectos producidos por el código HTML generado por el servidor, aunque no son 100 % eficaces ya que podrían no funcionar de igual manera sobre distintos navegadores, o bien cuando el usuario inhabilita la ejecución de código en Javascript. A continuación se enumeran algunas de las soluciones:

- Hacer que cada acción sobre la página actual (como presionar en un hipervínculo o un botón) abra una nueva ventana en el navegador, haciendo cerrar automáticamente la actual [19]. Algunos navegadores presentan una ventana de dialogo para que el usuario confirme si desea cerrar la ventana actual, lo cual hace que este efecto sea poco natural.
- Insertar en el <body> de la página HTML el siguiente código en JavaScript: `<script type="text/javascript">window.history.forward(1);</script>`. Esta sentencia se ejecutará cada vez que se cargue la página en el navegador; cuando es la última página cargada la instrucción `forward(1)` no tiene efecto, sin embargo cuando se presiona el botón de Retroceder se ejecuta nuevamente, haciendo que el navegador se redirija hacia la última página cargada en el historial.
- `<a href="javascript:location.replace('pagina.html');event.returnValue=false;">...</a>`: Hace que al presionar en el enlace se *reemplace* la página actual del historial por la referenciada en la función `replace` en vez de agregarse una nueva al final del historial de navegación. El efecto es que no hay página previa ya que es reemplazada por otra.

Soluciones en el servidor

Cuando se genera la página de respuesta que se enviará al navegador, el servidor puede solicitar al cliente que *no almacene* la misma en su cache. Esto hace que cuando se presione el botón Retroceder el navegador muestre una página de error indicando que la página que se desea buscar del historial no está disponible, o bien que se re-envíe el request al servidor para obtener nuevamente la página actualizada.

Para evitar que se almacene la página en el navegador se generan encabezados especiales del protocolo HTTP los cuales serán enviados antes del contenido de la página propiamente dicho. El Listado 2.1 muestra un ejemplo dentro de una página JSP. La línea 2 establece que la respuesta no debe ser usada nuevamente por el navegador a menos que se haya revalidado la misma con el servidor de origen. La línea 3 establece la misma directiva pero comúnmente se agrega para proveer compatibilidad con navegadores que interpretan HTTP 1.0. Finalmente la línea 4 establece la fecha y hora precisa en la que la respuesta se va a considerar expirada, donde el valor cero significa “ya expirado” lo que causa el efecto de forzar a que los servidores proxy no la almacenen en sus respectivos caches.

Otra forma en la que se puede controlar que el estado de la aplicación no quede desincronizado al presionar el botón de Retroceso es almacenando en el servidor algún dato que permita reconocer si el usuario ya ha realizado cierta acción o no. Supongamos que el usuario está completando un wizard que consta de 4 páginas consecutivas de manera que cuando se completa la última de ellas se almacenan los datos ingresados en una base de datos. Si se almacena en la sesión del usuario el número de página actual que está completando el usuario en ese momento de manera que vamos incrementando el valor de esta variable cada vez que avanzamos

```
1 <form name="miForm" action="/aplicacion/procesar.action" method="POST"
      onsubmit="this.btnEnvio.disabled=true; return true;">
2   ...
3   <input type="submit" name="btnEnvio" value="Enviar" />
4 </form>
```

Listado 2.2: Inhabilitando botón de envío con JavaScript

de página, y además guardamos ese valor como un campo oculto dentro del formulario de cada página en particular, entonces podrá detectarse el re-envío de una página previa comparando el valor de la variable almacenada en Session contra el enviado en el formulario actual, de manera tal de que si no coinciden estamos ante la presencia de un envío realizado desde alguna página anterior o posterior y por lo tanto podemos mostrar una página de error para informar al usuario, o bien podemos deshacer el estado actual del servidor y establecer el estado actual del mismo al informado en la página. Note también que esta es una muy buena forma de detectar doble envíos de formularios!

2.2.4. Múltiple envíos de datos

Es posible que en el servidor se reciban los datos del mismo formulario dos (o más veces) si el usuario presiona el botón de envío múltiples veces. Esto puede suceder por error del usuario al hacer doble-click sobre el botón de envío (en vez de una sola vez) o bien podría pasar si la operación tarda demasiado tiempo en ejecutarse, haciendo creer al usuario que los datos se han "perdido" en el camino de manera que luego presiona otra vez el botón de envío.

Otra forma en que pueden suceder múltiples envíos es cuando el usuario hace un primer envío exitoso y luego presiona el botón Retroceder, volviendo al mismo formulario y reenviando los mismos datos nuevamente, o bien presionando el botón de Refresco en la página de respuesta luego de hacer el envío (aunque se le presentará previamente al usuario la ventana de advertencia de la Figura 2.1).

Estos envíos múltiples son indeseables porque pueden generar problemas o confusiones, por ejemplo si el formulario de envío ejecuta una acción para registrar un nuevo usuario en un sitio web, el primer envío insertará al usuario dentro de una base de datos y enviará un email de confirmación, pero el segundo puede devolver un mensaje de error al usuario al intentar registrarse con un nombre de usuario que ya existe. Un ejemplo aún más crítico es el pago de un servicio a través de una aplicación de home banking, quizás haciendo debitar de la cuenta bancaria dos o más veces el mismo monto.

Soluciones en el cliente

Una forma simple de evitar el problema es inhabilitando el botón de envío del formulario mediante código en Javascript como muestra el Listado 2.2, sin embargo esta solución no tiene un comportamiento uniforme en distintos navegadores cuando se presiona el botón retroceder; por ejemplo sobre FF el botón de envío continua inhabilitado (inclusive continuará inhabilitado si presionamos el botón de Recarga), pero sobre IE aparece habilitado nuevamente.

Una variación de esta técnica es crear una variable global que mantenga el estado de si el formulario ya fue enviado o no [13] como muestra el Listado 2.3 pero también adolece del mismo problema ya que en IE no funciona (se resetea el estado de la variable global al presionar el botón Retroceder).

Finalmente, como con cualquier solución basada en JavaScript, sólo funcionará si el navegador lo tiene habilitado.

Soluciones en el servidor

El patrón POST-REDIRECT-GET protege contra los reenvíos resultantes de presionar el botón Refrescar, ya que una vez enviado el formulario la primera vez, la página de resultados se obtiene mediante un redirect, por lo tanto el botón Refrescar vuelve a traer del servidor la página de resultados solamente, ya que no se hace el POST realizado por el formulario.

```
var form_enviado = false;

function submit_form() {
  if (form_enviado) {
    alert("El formulario ya ha sido enviado!");
    return false;
  }
  else {
    form_enviado = true;
    return true;
  }
}
...
<form name="miForm" action="/aplicacion/procesar.action" method="POST"
    onsubmit="return submit_form()">
  ...
</form>
```

Listado 2.3: Inhabilitando botón de envío usando variable global

Para proteger contra los casos de que el usuario presione repetidamente el botón de envío o bien retrocediendo a la página del formulario con el botón Retroceder, se utiliza el patrón Synchronizer Token [1]. Este patrón funciona creando en el objeto `HttpSession` una variable con un valor aleatorio (el *token*) cuando el usuario quiere realizar una acción cuyo resultado es una página con un formulario para poder enviar. Dentro de este formulario se agrega una variable oculta con el mismo valor aleatorio almacenado en Sesión. Cuando el servidor recibe los datos del formulario, también recibirá el token, el cual procederá a comparar con el que tiene almacenado produciendo alguno de los siguientes resultados:

- Los tokens son iguales: El usuario ha enviado un formulario "válido", lo cual lleva a ejecutar el código asociado a la funcionalidad en forma normal, y antes de finalizar el procesamiento del request se graba en Sesión un nuevo token aleatorio (distinto al anterior).
- Los tokens son distintos: El usuario ha enviado un formulario "inválido", esto puede ser porque se ha enviado nuevamente el mismo formulario (o bien podría ser porque el request viene de una página almacenada en sus Favoritos). El request se interrumpe devolviendo alguna página de error.
- No se encuentra el token en Sesión: La probable causa de esto es que se haya entrado a la aplicación desde una página almacenada en Favoritos. El request se interrumpe devolviendo alguna página de error.

Como se puede ver, el patrón Synchonizer Token permite detectar cuando un formulario ha sido enviado múltiples veces, pero también permite detectar otros usos incorrectos de la aplicación, como hacer un bookmark de un formulario y utilizarlo directamente sin realizar la secuencia de pasos previa para que el estado de la aplicación no se corrompa (por ejemplo llenar un formulario sin loguearse previamente).

2.2.5. Múltiples ventanas de navegación

Durante la interacción con una aplicación, puede pasar que el usuario abra más de una ventana del navegador operando con la misma desde estos múltiples puntos. Puede hacer esto comúnmente cuando la aplicación tiene páginas con hipervínculos, ya que todos los navegadores modernos permiten mediante el menú contextual "Abrir el enlace en nueva ventana" y "Abrir el enlace en nueva pestaña". El problema proviene del hecho que estas ventanas y pestañas comparten la misma sesión de usuario que la ventana principal desde donde se abrieron, entonces las interacciones que realice el usuario desde estas ventanas pueden corromper el estado de la aplicación.

Por ejemplo supongamos que en una aplicación se puede obtener un listado de personas almacenadas en una base de datos, y que desde este listado se pueden modificar los datos personales de estas personas. Supongamos que el usuario hace clic con el mouse sobre una persona abriendo una nueva pestaña, lo que hace que se ejecute una acción en el servidor que carga los datos de esta persona P_1 desde la base de datos y los almacena en el objeto

`personaActual` en Sesión, y haciendo que la acción muestre un formulario donde se pueden editar estos datos; luego abre otra pestaña haciendo clic sobre otra persona P_2, con lo cual el objeto `personaActual` contendrá ahora los datos de P_2. Si el usuario modifica los datos sobre la ventana con los datos de P_1 y envía el formulario, en realidad (para sorpresa del usuario) la persona que será modificada será P_2, y no P_1 como intuitivamente él deseaba.

El ejemplo de arriba es un caso en el que puede pensarse una solución que no involucre almacenar los datos en Sesión, haciendo que la clave primaria de la persona sea un campo oculto del formulario, de manera que cuando se envíe el formulario, de manera inequívoca se obtenga la clave primaria de la verdadera persona a modificar, pero existen casos en que realmente es un requerimiento poder hacer procesamientos desde distintas ventanas y no resulta tan directo reescribir todo el estado del servidor en campos ocultos de formulario para luego poder procesarlo correctamente, de hecho, si en realidad reescribiésemos *todo el estado* del servidor como campos ocultos, no podríamos distinguir múltiples envíos del mismo formulario, como muestra el patrón Synchronizer Token de la sección anterior!

Soluciones en el cliente

Mediante código en JavaScript una página puede configurarse para que no aparezca el menú de contexto (el que aparece al hacer click sobre el botón derecho del ratón) de manera que no se tenga oportunidad de abrir un hipervínculo sobre una nueva ventana o pestaña, pero el código necesario para realizar esta funcionalidad es dependiente del navegador por lo que se suelen usar bibliotecas como OpenJS [27] o YUI [42] las cuales aumentan el nivel de abstracción para que el programador no necesite pensar en términos de cual navegador estará utilizando el usuario.

Esta solución no es muy efectiva para evitar que se abra un hipervínculo en una pestaña nueva porque los navegadores modernos como FF o IE8 permiten abrir una pestaña haciendo click sobre la rueda del ratón, entonces el usuario todavía tiene posibilidad de abrir otra ventana desde donde operar el sistema.

Soluciones en el servidor

Para poder hablar de las soluciones que se pueden implementar del lado del servidor, debemos primero decidir si queremos *prohibir* o *permitir* el uso de múltiples ventanas o pestañas por sesión de usuario.

Si lo que se desea es prohibir la posibilidad de abrir varias pestañas durante la misma sesión de usuario, no existen muchas opciones efectivas. El servidor podría evitar generar páginas que contengan hipervínculos reemplazándolos por formularios con botones para navegar por la aplicación: esto efectivamente anula la capacidad de abrir nuevas pestañas y ventanas con la página que contenga el resultado de haber enviado el formulario pero, por otro lado puede llegar a representar un problema de usabilidad de la aplicación ya que se pierde la forma natural que tiene un usuario de utilizar el navegador.

Si se desea permitir el uso de varias pestañas y ventanas durante la misma sesión de usuario, debe establecerse algún mecanismo de detección de manera de poder determinar a qué ventana pertenece un determinado request al servidor, y así poder evitar que los datos almacenados en la Sesión del usuario sean corrompidos. Una forma de implementarlo es que por cada hipervínculo que sea abierto, al llegar el request al servidor éste genere un número único identificando la nueva ventana o pestaña a abrir y haciendo que cada formulario o hipervínculo que proceda de esa ventana o pestaña incorpore ese número como un parámetro, de manera que cuando sea recibido por el servidor, éste pueda determinar a qué ventana pertenece ese request y tomar las acciones apropiadas. Ésta es la idea básica detrás del concepto de *Conversaciones*, una abstracción que ayuda a resolver este tipo de problemáticas de una manera muy elegante, y sobre la cual se profundizará en capítulos posteriores.

2.3. Alcance y tiempo de vida de objetos en el servidor

Ya hemos visto los problemas a los cuales están expuestas las aplicaciones web. Si bien las mejores soluciones (o al menos las más comunes) deben implementarse del lado del servidor, éstas hacen uso intensivo de la Sesión de usuario. El problema principal con el objeto Sesión es

Objeto en	Interfase o Clase	Alcance	Tiempo de vida	Múlt req
Web context	`ServletContext`	cualquier componente web dentro de la aplicación	hasta que se remueve el objeto, o se remueve la aplicación o se detiene el servidor	Si
Session	`HttpSession`	cualquier componente web sirviendo un request perteneciente a la misma sesión	hasta que se remueve el objeto, o se cierra la sesión, o bien expira por inactividad	Si
Request	`ServletRequest`	cualquier componente web sirviendo el request	hasta que se remueve el objeto o finaliza el request	No
Page	`JspContext`	la página JSP que creó el objeto	hasta que se remueve el objeto o finaliza el procesamiento de esa página	No

Cuadro 2.1: Objetos definidos por el servidor (adaptada de [29]).

que su granularidad es demasiado gruesa. Consideremos ahora el alcance y tiempo de vida de los objetos almacenados en Sesión, y en general de los objetos que pone a nuestra disposición la especificación del estándar de Servlets y JSP.

Las variables que contienen el estado de la aplicación dentro del servidor pueden ser guardadas en diferentes "espacios de almacenamiento" llamados *scopes*. El estándar define los siguientes cuatro scopes:

- page.
- request.
- session.
- application (web context).

donde cada uno define un alcance diferente para los datos que guardan, lo que implica que dependiendo de donde el programador decida guardar el dato, este tendrá un tiempo de vida diferente.

Dada una variable, por *alcance* entendemos el rango de sentencias que pueden acceder a la misma, mientras que el *tiempo de vida* de la variable está definido por el intervalo de tiempo en que la variable tiene un espacio de almacenamiento ligado a la misma.

Entonces, considerando las definiciones de arriba, en el Cuadro 2.1 se pueden comparar las características de cada uno de los objetos proporcionados por el servidor de aplicaciones para que el programador guarde variables del sistema. En particular la última columna de la tabla —que indica si el valor almacenado sobrevive en el servidor entre múltiples requests— muestra que los únicos scopes que preservan el valor entre requests son el contexto web y sesión, siendo sesión el de más granularidad entre ambos.

2.4. Características y problemas con los *scopes* estándar

Si bien los scopes estándar definidos por la especificación de Servlets y JSP son muy útiles para desarrollar diversos tipos de aplicaciones web, éstos no están libres de problemas, en particular los relativos al alcance y tiempo de vida *reales* de los objetos que una aplicación debe manejar, ya que cuando éstos difieren de los definidos por los scopes estándar, se produce una incongruencia semántica que debe ser salvada por el desarrollador.

2.4.1. Web Context

El Web Context es un scope el cual tiene un alcance global a la aplicación esto es, todos los servlets y páginas JSP ubicados bajo el mismo nombre de contexto web pueden tener acceso a los objetos almacenados allí. Este espacio de almacenamiento es creado cuando se inicializa la aplicación en el servidor de aplicaciones, y se destruye cuando se borra la aplicación del servidor (o bien cuando el servidor de aplicaciones es detenido por un administrador). En caso de que el servidor de aplicaciones esté distribuído entre múltiples máquinas virtuales (cluster), habrá un Web Context por cada JVM (es decir no actúa como un repositorio distribuido de objetos).

El almacenar un objeto en el web context hará que éste sea visible a todas las sesiones de usuarios de la aplicación, de modo que tiene sentido usarlo para almacenar datos globales a la misma, como por ej. datos de configuración o bien objetos inmutables. No es conveniente

almacenar objetos mutables en este scope ya que puede muy fácilmente terminar produciendo un *memory leak* (por ej. si se van agregando datos a una lista dinámicamente). También las modificaciones a los objetos deben estar debidamente protegidas para evitar interferencias entre modificaciones concurrentes y *race conditions*, lo cual puede llegar a terminar siendo un problema de performance, ya que habría mucha contención por adquirir el mismo recurso entre varios componentes web ejecutando en forma concurrente.

2.4.2. Session

Consideremos el caso de que una aplicación deba preservar los datos de un objeto entre varios requests y que sólo deba ser accesible por un usuario en particular, la opción más natural es almacenarlo en sesión debido a que la opción de almacenarlo en Request implicaría que este dato deba ser reescrito en las subsiguientes páginas de la aplicación ya sea como campo oculto de un formulario, o como parámetro adicional en un hipervínculo, lo cual se traduce en un trabajo adicional para el programador y puede llegar a ser muy propenso a errores. Ahora bien, si el dato a preservar entre distintos requests son las credenciales de un usuario con sus datos de autorización y roles que permiten al mismo operar sobre distintas pantallas de la aplicación, se produce un **encaje perfecto** entre el alcance y tiempo de vida de sesión y el de los datos de autorización, ya que los mismos tienen el mismo tiempo de vida que la sesión del usuario, es decir por un lado estos datos no pueden ser removidos de la sesión antes de terminar la misma, y por otro lado si se cierra la sesión o bien expira por inactividad es correcto que estos datos no deban ser mantenidos en sesión ya que ésta ausencia activará los controles necesarios en la aplicación para que el usuario vuelva a autenticarse nuevamente. Pero supongamos que el dato que se necesita guardar consiste en un objeto cuyos datos se van solicitando en diferentes pantallas de un wizard de modo que cada pantalla actualiza una parte de los datos de este objeto, y al llegar a la última pantalla se realiza una transacción en la base de datos de la aplicación salvando los datos solicitados; asimismo este objeto debe ser visible solamente por el usuario actual. Nuevamente es natural elegir incorporar este objeto en sesión debido a que su estado debe ser mantenido a través de diferentes requests, sin embargo el tiempo de vida de este objeto **no coincide** con el de sesión, puesto que al terminar el procesamiento este objeto debe ser eliminado de sesión en forma manual por el programador! Si bien esto ha sido (y sigue siendo) una solución muy aceptada para tratar este tipo de incongruencias semánticas, no es lo ideal ya que introduce problemas de desincronización del estado del servidor, a saber:

- Al presionar el botón Retroceso: El objeto desaparece luego de completar por primera vez el wizard.
- Al abrir múltiples ventanas o pestañas: El objeto es compartido por todas ellas ya que se hace referencia a la misma sesión del navegador lo cual lleva a problemas de interferencia de datos al ejecutar requests desde distintas ventanas o pestañas.

En general, no puede haber problemas de desincronización de datos almacenados en sesión cuando:

- El alcance y tiempo de vida de los datos a almacenar concuerdan con el de sesión, y
- Los datos son almacenados en sesión tan pronto como se cree la misma en el servidor, y
- Los datos a almacenar son de solo lectura (no pueden ser modificados por el usuario ni por el sistema).

Cualquiera de estas condiciones que no se cumpla y estaremos lidiando con las situaciones descritas en la Sección 2.2. Estas situaciones son en realidad *bugs* o errores de las aplicaciones pero son tan comunes que muy poca gente los considera como tal.

2.4.3. Request

El objeto que representa el scope Request en un servlet o página JSP es un subtipo de la clase `javax.servlet.ServletRequest`, el cual tiene un tiempo de vida de sólo un request, esto es, todos los objetos de la aplicación almacenados en este scope, serán eliminados tan pronto como se termine de enviar la respuesta del servidor al cliente. Esto implica que múltiples clientes

accediendo al mismo componente web operan sobre sus propias copias de objetos almacenados en Request: no habrá nunca interferencia entre los requests de diferentes clientes.

Este scope se utiliza principalmente para almacenar objetos que tienen un tiempo de vida muy corto, y que no hace falta que sobrevivan entre dos o más requests ya que se consigue similar funcionalidad creando nuevas copias de los mismos para cada request que llegue al servidor.

Es posible simular que un objeto sobreviva múltiples request haciendo una re-escritura del mismo en cada envío de una respuesta por parte del servidor, y haciendo que el próximo request contenga este objeto como parámetro o campo de formulario, haciendo de esta forma que el valor "viaje" por cada request realizado por el usuario, creando la ilusión de que el objeto preserva el estado en el servidor. Esta técnica hace que la funcionalidad de la aplicación sea *stateless* es decir, que no requiere que el servidor guarde información de estado, lo que hace que la aplicación sea más escalable comparándola a una solución donde se guarda el mismo objeto en Session (lo que produciría una funcionalidad *stateful*), sin embargo es una solución antinatural y mucho más engorrosa ya que puede ser propensa a errores, de manera que sólo debe utilizarse cuando las demás opciones no son convenientes.

2.4.4. Page

Este es el scope más pequeño, y tiene lugar solamente dentro de una misma página JSP. Los objetos almacenados allí son visibles solamente dentro de esa página particular, y son destruidos ni bien termina el procesamiento de esa página, ya sea porque ya se ha enviado la respuesta al cliente, o bien el request es forwardeado hacia otro destino [12].

Evidentemente este scope tiene sentido utilizarlo sólo para variables temporales dentro de una misma página JSP, comúnmente con el objeto de ayudar a trazar el diseño de la misma, por ej. para el armado de una tabla complicada o bien iterar sobre una colección de objetos.

2.5. Conclusiones

Como hemos descrito a lo largo de este capítulo, los navegadores están históricamente diseñados para permitir una navegación libre por la WWW, mientras que las aplicaciones web son particularmente sensibles a esta navegación sin control, ya que por las diversas formas de interacción con el usuario el estado de la aplicación puede corromperse o desincronizarse rápidamente dando lugar a errores, problemas de estabilidad o de confiabilidad.

Las formas de interacción del usuario con el navegador son muy variadas, así como los efectos que pueden causar: por ejemplo el acceso a una página desde un bookmark, un pedido de Refresco, Retroceder o Avanzar en el historial de navegación, hacer múltiples envíos de los mismos datos o abrir varias ventanas o pestañas desde donde se continuará operando la aplicación, cada una con un abanico de posibles soluciones que se pueden aplicar del lado del cliente (navegador) o del servidor (Servlet o páginas JSP).

De todo lo expuesto se puede sacar una conclusión:

> Un control de navegación incompleto junto con la presencia de usuarios que accidentalmente usan las ayudas navegacionales causan problemas en la aplicación. Ya que muchos de estos problemas son identificables, podemos definir a priori controles adecuados para evitar que la aplicación entre en estados no deseados, de manera tal que se comporte en forma predecible, robusta y confiable.

Capítulo 3

Conversaciones en aplicaciones Web desarrolladas en Java

Como hemos visto en el Capítulo 2, existe un gap semántico entre los scopes estándares y las variables que representan el estado de una aplicación web. En este Capítulo definiremos una nueva abstracción para ayudar a solucionar este problema: las *conversaciones*; se presentarán ejemplos de uso de las mismas y se analizarán sus ventajas y desventajas.

3.1. Definición informal

Las conversaciones son una abstracción que nos permite gestionar el estado de la aplicación web en forma más efectiva que si lo hiciéramos almacenando el estado en Session. Según el estándar de servlets y páginas JSP, la sesión de usuario es el único scope (visible sólo para un usuario en particular) disponible al programador para poder gestionar el estado de la aplicación que puede sobrevivir múltiples requests, pero este scope típicamente resulta ser poco efectivo.

Escrito en forma sencilla, **una conversación es un espacio de almacenamiento de estado**, similar a una `HttpSession`, pero con el beneficio de **que permite tener múltiples espacios de almacenamiento concurrentes para una misma sesión de usuario**, como se muestra en la Figura 3.1.

En definitiva, una conversación es cualquier acción o tarea llevada a cabo por el usuario que toma varias páginas para poder completarse: un wizard o un carrito de compras son ejemplos obvios de conversaciones. Sin embargo, cada ciclo de request/response también es una conversación ya que involucra dos páginas: la página con el formulario enviado en el request, y la página con la respuesta. Una sesión HTTP de usuario puede contener múltiples conversaciones, y cada una de ellas puede estar limitada a una ventana o pestaña del navegador web.

3.2. Propagando el contexto de la conversación

Como ya sabemos, las conversaciones pueden abarcar múltiples ciclos de request/response entre el navegador web y el servidor de aplicaciones y, dado que el protocolo HTTP es stateless, para poder retomar una misma conversación ante un nuevo request, se necesita una forma de saber **cual** de todas ellas en la sesión de usuario es la que hace referencia ese request en particular. En otras palabras, cada ciclo de request/response esta inmerso dentro del contexto de una conversación en particular, y éste contexto debe ser propagado en cada request para poder seguir ejecutando las acciones en el servidor dentro de ese mismo contexto.

La opción más común es propagarlo agregando un dato extra en el request, o sea en un GET request se agregará un parámetro extra, y si es un POST request se incluirá como un campo adicional (por ejemplo un campo `hidden` dentro de un `form`), y como **el contexto debe ser identificado unívocamente** dentro de la sesión del usuario, este parámetro es un **identificador de conversación** dentro de la sesión de usuario, o en inglés *conversation id*. En el Listado 3.1 se muestra la propagación del identificador de la conversación, mediante el parámetro `cid`.

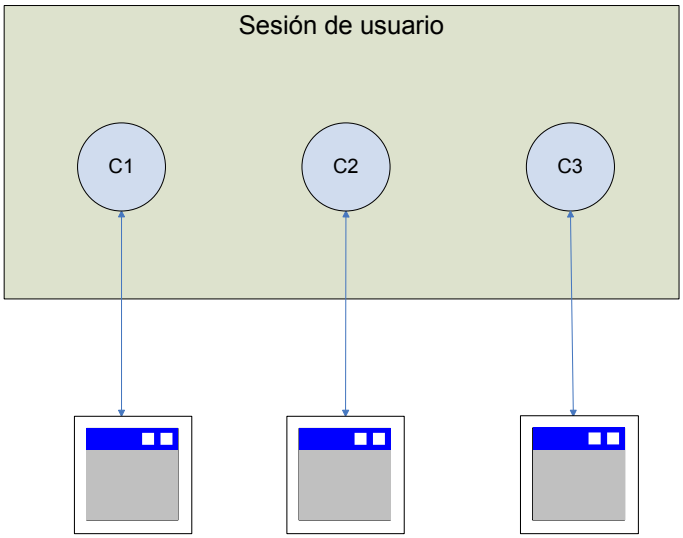

Figura 3.1: Sesión de usuario con tres conversaciones

```
request mediante GET:
http://www.dominio.com/aplicacion/agendarHotel.action?cid=123

request mediante POST:
<form action="/aplicacion/agendarHotel.action" method="POST">
  <input type="hidden" name="cid" value="123"/>
  ...
</form>
```

Listado 3.1: Propagando la conversación mediante su identificador

3.3. Identificación de conversaciones

Como vimos en la Sección anterior, se necesita de un identificador de conversación para poder propagar el contexto en el que está ejecutando la tarea del usuario, y que identifique unívocamente a la conversación. En principio, cualquier cadena de caracteres única dentro de la sesión de usuario será útil para utilizarla como identificador, sin embargo no es necesario que ésta sea creada en forma aleatoria. En particular, dos formas del identificador son comunes dentro de los frameworks, una es utilizada por su simplicidad, y la otra por su practicidad.

En efecto, en forma análoga al concepto de *clave primaria* de una tabla dentro de una base de datos relacional, existen dos formas comunes de identificar a las conversaciones, una mediante una clave *sintética* y otra mediante una clave *natural*.

3.3.1. Identificación mediante clave sintética

Utilizar una clave sintética significa que la identificación de la conversación es independiente del contenido de la misma. La implementación más común es utilizar un valor numérico entero en forma secuencial, que la aplicación generará automáticamente cuando se cree una nueva conversación. Un ejemplo es el `cid` del Listado 3.1. Por supuesto, a cada conversación debe ser asignado un número único.

La ventaja de utilizar este tipo de identificador es que es sencilla de implementar, sin embargo no hay forma de que el usuario cree una regla mnemotécnica para relacionar este identificador artificial con un objeto particular sobre el cual está trabajando.

3.3.2. Identificación mediante clave natural

Consideremos como ejemplo una aplicación genérica de reserva de hoteles, donde la aplicación le ofrece al usuario un listado de hoteles con habitaciones disponibles para el día y hora

seleccionados. Para reservar una habitación, se presiona el botón de reserva sobre un hotel en particular, y esto genera una nueva conversación, la cual es identificada por el parámetro `cid=437` si se hubiese utilizado el esquema de clave sintética. Pero tal vez la aplicación fuese más usable si la conversación pudiese ser identificada por el parámetro `cid=Hilton`, o bien por `cid=Sheraton`; de esta forma el usuario puede crear una técnica mnemotécnica para relacionar la reserva de un hotel en particular y poder ejecutar (desde un Favorito) la misma directamente en caso de ser necesario.

Más aún, si se utiliza un identificador natural para la conversación, es posible retomarla luego en el estado donde había quedado por última vez, en vez de crear *otra conversación* si se presiona otra vez el botón de reserva de un hotel particular. Esto sucede porque la aplicación puede detectar que la conversación que se intenta crear ya existe, a causa de que se está utilizando la misma clave natural para identificarla. Este caso no podría haberse detectado con un esquema de clave sintética.

3.4. Niveles de abstracción

Un framework MVC que tenga soporte de conversaciones puede proporcionar distintos niveles de abstracción a disposición de los programadores para poder operar con ellas. El espectro de opciones puede ir desde proporcionar un aislamiento total de si se utilizan conversaciones o no, como sólo proporcionar el soporte de conversaciones a través de una API. En las subsecciones siguientes se discutirán distintas opciones posibles de manipulación de las conversaciones; distintos frameworks en el mercado pueden soportar uno o más de estos mecanismos.

3.4.1. Lenguaje de Expresión

En inglés, *Expression Language*, o abreviado EL, es un lenguaje de scripts que permite el acceso a componentes (que respetan la nomenclatura de los JavaBeans) dentro de las páginas JSP, así como la posibilidad de llamar métodos arbitrarios. Existen en el mercado varios EL que son utilizados en los frameworks, a saber, Unified EL estándar utilizados en JSP y JSF [12], OGNL [26] y MVEL [9].

La sintaxis es muy simple, por ejemplo para acceder a la propiedad `nombre` del objeto `persona` almacenado en la sesión de usuario en una página JSP, escribimos:

- `${sessionScope.persona.nombre}` ó
- `${persona.nombre}`

En el primer caso, se especifica el scope donde está almacenado, mientras que en el segundo caso, el framework buscará ese objeto a través de todos los scopes del sistema, comenzando por los de menor alcance, hasta llegar a los de mayor alcance: page, request, session, application. Para un framework que soporte conversaciones es inmediato el hecho de que el scope de la conversación actual debe buscarse después de request, pero antes que session, de manera que el orden de búsqueda queda: page, request, conversation, session, application.

3.4.2. Archivo de flujo de navegación

Se trata de un archivo que define cual será el flujo o secuencia de navegación de un caso de uso, en particular de una conversación. Este archivo define una abstracción similar a una máquina de estados finito donde se tiene:

- Estados: son las configuraciones en las que se puede encontrar la aplicación en un instante de tiempo particular. Cada estado tiene asociada una acción ejecutable y una vista (o página) de la aplicación.
- Estados iniciales: estados en los cuales se crea una nueva conversación.
- Estados finales: estados en las cuales termina (o se cancela) una conversación.
- Transiciones: definen acciones o eventos realizados por el usuario, que producen cambios en el estado de navegación de la aplicación.

```
1 public class PagoReservaHotelAction extends ActionSupport {
2   @In
3   private Hotel hotel;
4   @In @Out
5   private ReservaHotel reserva;
6   ...
7   public String execute() { ... }
8 }
```

Listado 3.2: Inyección de código

Dentro de este archivo —que por lo general tiene formato XML— se definen las acciones que la aplicación ejecuta al recibir un request, y por lo general se utiliza un EL para definir las variables involucradas en la conversación, o bien para definir los métodos a llamar cuando se produce un request. El framework Spring Web Flow [36] utiliza este tipo de abstracción.

3.4.3. Dependency Injection, Outjection y Bijection

Los contenedores de inversión de control (IoC) como Spring [34] y Guice [20] simplifican el desarrollo de aplicaciones ya que se encargan de instanciar, configurar y establecer en tiempo de ejecución los objetos que usaremos, así como todas sus dependencias en forma transitiva. Esta funcionalidad comúnmente llamada "inyección de dependencias" reduce la cantidad de código necesario para crear e inicializar objetos, lo cual es gestionado por el contenedor, entonces el código de la aplicación queda más legible. Otra ventaja es que la facilidad de cambiar la configuración de objetos hace que sea más sencillo hacer pruebas unitarias, ya que fácilmente se puede cambiar el archivo de configuración del contenedor por uno donde se haga referencia a *mock objects* (objetos que tienen la misma interface que el original, pero que tiene un comportamiento muy básico para "simular" el comportamiento del objeto real que se quiere probar).

En los objetos que componen la capa de la lógica de presentación (como los `Action` de Struts), además de inyectar los servicios necesarios para ejecutar la funcionalidad de la acción, también se pueden inyectar datos que están almacenados en alguno de los scopes del servidor de aplicaciones, así también como los parámetros del request. En particular, es muy conveniente inyectar datos que están almacenados en la conversación, de manera que el código del `Action` solo haga referencia a estos datos, y no a un `Map<String,Object>` genérico donde se pueden recuperar y salvar los datos en la conversación: todo esto aporta a la legibilidad del código. Por ejemplo, en el Listado 3.2 se puede ver un `Action` al cual se le inyecta el `Hotel` y `ReservaHotel` asociados a la conversación actual.

Por otro lado, la ejecución del `PagoReservaHotelAction` va a generar una reserva del hotel, la cual debe ser guardada en la conversación para luego mostrar una pantalla de confirmación al usuario. Esta extracción del valor de una variable dentro de un scope en particular luego de terminada la ejecución del `Action` es llamado "outjection", y junto con la inyección de dependencias, a esta funcionalidad se la conoce como "biyección". En realidad, la biyección es crear un alias de una variable almacenada en algún contexto o scope, dentro de un atributo de un componente. El framework JBoss Seam [24] soporta biyecciones.

3.4.4. API

Es tal vez la forma de más bajo nivel para programar con conversaciones, en el que el framework expone una API a través de la cual el desarrollador pueda operar sobre éstas. Las funcionalidades básicas que debería suministrar esta API son las siguientes:

- Crear una nueva conversación.
- Terminar una conversación existente.
- Almacenar, obtener y eliminar un atributo de la conversación.
- Establecer el tiempo de expiración (timeout) de la conversación.
- Obtener el identificador de conversación.

- Establecer el identificador de conversación (en caso de que soporte conversaciones naturales).

Naturalmente, la API debería proporcionar la capacidad de que el programador explote al máximo el framework, de manera que este debería estar desarrollado para que valga la pena utilizar la API solamente en el caso de que se requiera funcionalidad muy específica del mismo, y para las operaciones básicas con conversaciones se pueda utilizar algún otro método de más alto nivel de abstracción.

3.4.5. Listeners

De manera semejante a como el estándar de servlets define listeners para ser notificados de eventos que ocurren con la sesión de usuario, también pueden existir listeners análogos para notificar eventos que ocurren sobre las conversaciones, como por ejemplo cuando:

- Se crea una nueva conversación.
- Se destruye una conversación.
- Se activa o pasiva una conversación (en realidad es la sesión la que se activa o pasiva).
- Se agrega, modifica o elimina un atributo de la conversación.
- Se agrega o elimina un objeto particular de una conversación.

Si bien estos listeners no son necesarios para desarrollar usando conversaciones, complementan de forma natural el API para que el programador tenga control total sobre todos los aspectos del ciclo de vida de la conversación.

3.5. Terminación de conversaciones

Al momento de que el sistema haya culminado su procesamiento en forma exitosa, es momento de dar por terminada la conversación. Los frameworks por lo general le permiten al programador especificar en qué momento se destruye la conversación: inmediatamente o luego de hacer un redirect HTTP, o bien cual será el destino de la conversación al ocurrir una excepción.

3.5.1. Terminación normal

Si la conversación termina inmediatamente, al hacer un redirect HTTP hacia una página de confirmación, ésta ya habrá sido destruida y no podrá ser encontrada, mientras que si se especifica que se destruirá *después de hacer un redirect*, entonces la conversación permanecerá viva hasta luego de haber completado la recepción del redirect (esta última opción hace que sea muy sencillo de implementar el patrón POST-REDIRECT-GET).

3.5.2. Comportamiento ante la presencia de excepciones

En líneas generales, la decisión de si se termina una conversación ante la presencia de una excepción es muy subjetiva y debe ser evaluada caso por caso. Depende del programador si la excepción permite hacer un nuevo reintento ya que se trata de una condición temporal, o bien se da por terminada dicha conversación.

Por ejemplo, si sucede una `java.sql.SQLException`, lo que indica una condición de error al acceder a una base de datos, tal vez no debería terminarse la conversación dado que puede ser un error momentaneo y por lo tanto el usuario puede intentar realizar la operación nuevamente. En cambio si se trata de una excepción propia de la aplicación como por ej. `FondosInsuficientesException` al hacer una transferencia en una aplicación de home banking, dicha excepción no es recuperable ya que (a menos que se depositen fondos desde otro lugar) haciendo un reintento para hacer la transferencia, la aplicación no arrojará un resultado distinto.

3.5.3. Acceso a conversaciones ya terminadas

Puede pasar que, luego de terminar una conversación el usuario presione el botón Retroceder en el navegador e intente seguir procesando sobre la misma, o puede suceder que el usuario haga un request sobre una conversación que ya ha expirado, o bien que el usuario fabrique una URL con un identificador inválido de conversación. Estas situaciones son detectables dado que se está intentando acceder a una conversación que no existe; en tal caso el framework debería proveer al programador de algún evento para detectar tal situación y poder tomar la medidas necesarias para el caso.

Una solución común es elevar alguna excepción particular del framework como `ConversationNotFoundException` para indicar el error, de manera de poder mostrar una pantalla de error al usuario. Además, cuando ocurre esto, lo más sensato sería cerrar la ventana o pestaña donde aparece el error, ya que siendo la estrategia que cada conversación opere en una ventana particular, el mismo indica que esta ventana ha quedado inutilizada.

3.6. Políticas de expiración de conversaciones

De forma similar a la sesión de usuario, a las conversaciones se les puede definir un *tiempo de expiración*, que es el tiempo máximo que puede esperar la conversación sin tener actividad; pasado este período de tiempo cualquier intento de acceso a los datos en esa conversación resultará en un error. Más aún, como las conversaciones existen dentro de la sesión de usuario, este período de expiración está limitado al de la sesión de usuario.

Existen distintas políticas de expiración que se podrían adoptar: por tiempo fijo, o por tiempo variable, ya sea dependiendo de la conversación que se esté retomando o por la página que se esté viendo. Existen además otras políticas de expiración que no dependen del tiempo de inactividad, sino en el número de conversaciones activas concurrentes.

3.6.1. Expiración por tiempo fijo

Es la política más simple, y se trata de definir a nivel global un tiempo de timeout (usualmente en término de minutos o segundos) único para todas las conversaciones. Por supuesto este tiempo de expiración debe ser menor al definido para el timeout de sesión (caso contrario no tendría sentido definirlo). Este valor de timeout típicamente se define en algún archivo XML de configuración del framework.

También se podría definir un valor de timeout por cada conversación en particular, por ej. en una aplicación que tenga funcionalidad de reserva de hotel y reserva de autos para alquilar, tal vez los usuarios tardan más tiempo reservando un auto que un hotel, entonces se podría definir un valor distinto para cada tipo de conversación.

La ventaja de esta política es que es muy simple de implementar y de entender. Por ejemplo es posible definir un valor de timeout de sesión de 15 minutos, y un timeout de conversación de 10 minutos.

Una desventaja notable de esta política es que al usuario no se le permite realizar un nuevo request sobre una conversación después de expirado el timeout de conversación, pero sin haber expirado el tiempo de sesión. Por ej. en el caso de hacer un request a los 12 minutos, la conversación estará expirada pero no la sesión, entonces de alguna forma se está restringiendo el uso de la aplicación al usuario cuando en realidad aún puede hacerlo ya que su sesión sigue vigente.

3.6.2. Conversaciones en primer y segundo plano

Esta política de expiración implica clasificar las conversaciones en dos tipos: Existe una conversación ejecutando en primer plano (*foreground*), y el resto se dice que están en segundo plano (*background*).

La conversación ejecutando en primer plano es la realizada en el último request, es decir, cuando la aplicación está procesando un request retomando una conversación, ésta queda en primer plano, y el resto de las conversaciones abiertas por el usuario quedan en segundo plano.

La idea es que *solo expiren las conversaciones en segundo plano*, en otras palabras el tiempo de expiración para la conversación en primer plano no se toma en cuenta, con lo cual ésta seguirá vigente mientras la sesión de usuario no expire. La expiración de las conversaciones en segundo

plano se realiza cada vez que la aplicación recibe un request, pudiendo de esta forma comparar los tiempos de timeout con el tiempo actual y eliminar las que corresponda.

Esta política permite que el usuario siga operando la aplicación normalmente de acuerdo al tiempo de expiración de la sesión preestablecido, pero como desventaja se distingue el hecho de que la aplicación necesariamente debe realizar la limpieza de las conversaciones expiradas al momento de recibir un request, y por lo tanto puede suceder que una conversación no se elimine exactamente en el período configurado, es decir, si el timeout está configurado a 10 minutos, y el próximo request se recibe a los 14 minutos, recién en ese momento se eliminarán físicamente las conversaciones expiradas del servidor.

3.6.3. Expiración selectiva por vista

Esta política permite especificar un valor de timeout diferente de acuerdo a la vista (página) que se esté mostrando en el navegador, dependiendo de los patrones de usabilidad de la aplicación.

Por ej. en una aplicación de reserva de hoteles, donde hay una conversación que abarca 3 páginas:

1. verHotel: muestra los detalles del hotel seleccionado para reserva.
2. reservar: presenta un formulario con los datos de ingreso de tarjeta de crédito, días que abarca la reserva, etc.
3. confirmar: presenta una pantalla de confirmación de los datos ingresados en la página anterior.

En este caso sería recomendable —a efectos de salvar recursos del servidor— que la primer vista tuviese un tiempo de expiración menor que el resto, digamos 5 minutos, mientras que las páginas 2 y 3 tengan 10 minutos. La razón es que en la primer página de la conversación el usuario ve los detalles del hotel a reservar, y es posible que seleccione varios de ellos antes de continuar con la vista de reserva o confirmación, las cuales ya forman parte del proceso de la reserva propiamente dicha, y es probable que el usuario pase más tiempo en estas páginas. En otras palabras, el patrón de usabilidad de esta aplicación nos indica que es probable que el usuario cree muchas conversaciones que son abandonadas en la primera página, mientras que se crean muy pocas que llegan al final de la misma, de modo que es más eficiente desde el punto de vista de la gestión de recursos en el servidor que las conversaciones que lleguen a la primer página expiren lo más pronto posible.

Esta política en realidad podría ser implementada usando conversaciones en primer y segundo plano, o bien haciendo *pooling* de las mismas cada cierto intervalo de tiempo; lo importante de esta política es que se puede definir un timeout diferente por *cada vista* por la que va pasando la conversación.

La ventaja de usar esta política es una mejor gestión de los recursos del servidor, ya que cada vista puede ser “tuneada” para alcanzar óptimos niveles de rendimiento. Como desventaja, el encontrar valores de expiración adecuados para cada vista requiere de un análisis previo que contemple los patrones de interacción y usabilidad de la aplicación, así como un análisis del consumo de recursos.

3.6.4. Expiración por cantidad de conversaciones activas

Una política de expiración no necesariamente debe depender del tiempo de inactividad que ha tenido una conversación, sino que puede depender de otros factores. Un ejemplo es contar la cantidad de conversaciones ejecutando concurrentemente en la misma sesión de usuario: si este número alcanza un nivel considerable, es posible que se trate de alguna tarea que el usuario ha comenzado pero que (tal vez) nunca termine, y estas conversaciones pueden producir un consumo de memoria considerable dentro del servidor de aplicaciones.

Si se define una *cantidad máxima de conversaciones ejecutandose por sesión de usuario* este consumo de memoria se puede limitar de manera que se obliga al usuario a no abrir muchas pestañas en el navegador creando nuevas conversaciones.

Cuando se supere el número máximo definido de conversaciones activas, el framework debe seleccionar una para “desalojar” del sistema (de manera similar a una memoria caché) utilizando

algún algoritmo de desalojo como por ej: eliminar la más antigua, o eliminar la menos recientemente usada (LRU).

La implementación de esta política puede realizarse sencillamente manteniendo una lista de las conversaciones activas dentro de la sesión de usuario, contando la longitud de la misma. Cuando llega una petición solicitando la creación de una nueva conversación, se selecciona (con el criterio de desalojo elegido) una de ellas, la cual será destruida y eliminada de la lista.

La ventaja principal es una mejor gestión de los recursos del servidor, ya que un usuario estará limitado en la cantidad de conversaciones que puede mantener activas en un momento dado. Como desventaja, encontrar un valor adecuado del número máximo de conversaciones activas permitidas, así como la selección del algoritmo de desalojo de conversaciones invitan a hacer un análisis de la forma de interacción del usuario con la aplicación.

3.7. Acceso concurrente

De manera similar al acceso a los datos almacenados en la sesión del usuario —representado en Java por la clase `HttpSession`— el acceso concurrente debe hacerse de forma controlada para no corromper los datos, generar *race conditions* ni ocasionar *deadlocks*.

En general, los frameworks pueden adoptar distintas políticas al implementar la forma de acceso concurrente sobre una misma conversación: pueden hacerse responsable de garantizar el acceso seguro a la misma, o pueden dejar esa responsabilidad al desarrollador.

3.7.1. Con control de concurrencia

En esta política, el framework es el encargado de serializar los requests concurrentes sobre la misma conversación y por lo tanto el desarrollador no debe preocuparse en implementar medidas para garantizar el acceso correcto a los datos almacenados en ella.

La ventaja aquí es que el desarrollo se simplifica y el código será más legible debido a que no será necesario usar las bibliotecas con primitivas de programación concurrente.

La desventaja es que el framework hace que los requests concurrentes en el contexto de la misma conversación (o sesión) sean encolados y ejecutados en forma secuencial. Esto permite que cada request sea ejecutado en forma determinística. Sin embargo esta serialización de requests hace que —si éstos tienen un tiempo de procesamiento grande para completarse— comiencen a encolarse demasiado con lo cual cada uno de ellos no tendrá oportunidad de ejecutarse hasta que los que estén adelante en la cola no hayan terminado (con los riesgos de una ataque de negación de servicio que esto implica), y por otro lado, aplicaciones que utilizan AJAX y que requieren mostrar mensajes de estado en forma frecuente, simplemente que éste request tarde demasiado tiempo ya torna inútil este tipo de funcionalidad.

En los frameworks que fuerzan el procesamiento secuencial de requests en el contexto de la misma conversación, se puede configurar un *tiempo de expiración de request*, que es un timeout que indica la cantidad de tiempo máxima que un request puede estar esperando en la cola de ejecución; pasado este tiempo, el request no se procesa en forma normal, sino que se genera algún tipo de mensaje de error que pueda mostrarse en el navegador.

3.7.2. Sin control de concurrencia

Bajo esta política, el desarrollador es el responsable de asegurar que el acceso concurrente a la conversación sea correcto, ya que forzar una exclusión mutua sobre una misma conversación desde el framework MVC implica que puede haber mucho tiempo de espera entre múltiples requests accediendo a la misma conversación (a causa del acceso secuencial a la misma). En el espectro de soluciones, también podemos encontrar que el framework MVC tome la responsabilidad de garantizar que la lectura y escritura de la conversación per se (dentro de la sesión) sea realizada en forma correcta, o ni siquiera esta opción, aunque éste último caso probablemente estaría evidenciando un framework con soporte de conversaciones no muy maduro.

Pero a diferencia del acceso concurrente a la sesión del usuario, el acceso concurrente a la misma conversación es menos frecuente ya que ésta tiene un alcance y tiempo de vida mucho más limitado. Por ejemplo, asociando una conversación a cada ventana o pestaña del navegador, ya hemos protegido a la aplicación de requests concurrentes producidos desde distintas ventanas o pestañas puesto que se trata de conversaciones distintas.

Sin embargo, es común que en aplicaciones con tecnologías de request asincrónicas (AJAX) se observen requests concurrentes en una misma conversación, debido a que el usuario normalmente no está consciente de que (tal vez cada vez que presiona una letra en el teclado) se estén haciendo requests en forma frecuente al servidor de aplicaciones.

En general este problema existe en todas las aplicaciones AJAX, ya que éstas normalmente producen una gran cantidad de requests, entonces, si se presenta el caso de que los clientes inunden de requests al servidor, sería adecuado incorporar algún control del lado del cliente que permita desacelerar la taza de requests.

3.8. Integración con Mapeos Objeto-Relacional

Las soluciones de Mapeos Objeto-Relacional (en inglés *Object-Relational Mapping*, abreviado ORM) como Hibernate [21] y JPA [32] simplifican el desarrollo de aplicaciones por medio del agregado de una capa de abstracción que permite mapear registros, tablas y relaciones entre tablas de una base de datos relacional en objetos. Java, al ser un lenguaje que sigue el paradigma de Programación Orientado a Objetos, provee en forma nativa las construcciones apropiadas para gestionar clases y el ciclo de vida de objetos, mientras que las bases de datos relacionales son muy buenas para almacenar y recuperar información estructurada usando el lenguaje SQL (basado en el álgebra relacional). Ambas tecnologías son tan fundamentalmente diferentes, que herramientas como las de ORM son muy deseables ya que tanto el lenguaje como la base de datos pueden programarse “hablando” cada uno en su idioma nativo.

3.8.1. Transacciones en aplicaciones Web

Un tema siempre importante en las aplicaciones Web son las *transacciones*. Debido a que el usuario que opera una aplicación a través del navegador tiene un tiempo para pensar entre cada una de sus acciones, entonces es imposible gestionar las transacciones de la misma forma en que se haría para una aplicación cliente/servidor funcionando en una red corporativa. Por un lado, normalmente se necesita que las aplicaciones Web sean lo más escalables posible, lo que requiere que se minimicen tanto los accesos, la cantidad de conexiones abiertas, y la cantidad de *locks* mantenidos en tablas/tuplas de la base de datos.

En general, un motor de base de datos relacional debe garantizar que las transacciones cumplan con las propiedades *ACID: Atomicity, Consistency, Isolation, Durability*:

- Atomicity (Atomicidad): Una transacción debe completarse exitosamente en forma completa, o bien no debe tener efecto ninguna operación de la misma (es decir no puede quedar a medias).
- Consistency (Consistencia): Cada transacción debe cambiar el sistema desde un estado consistente a otro.
- Isolation (Aislamiento): Ninguna transacción debe interferir con otra transacción.
- Durability (Durabilidad): Una vez que una transacción ha sido comprometida (commit), sus efectos deben ser persistidos en forma permanente, aún en presencia de fallas del sistema.

Al desarrollar aplicaciones Web, el programador debe manejar con cuidado estos cuatro criterios deseables de las transacciones, debido a que algunas cuestiones no es posible manejarlas desde la aplicación y, recíprocamente, otras sólo pueden manejarse desde la misma. Es más, **es deseable que cuando se desarrolla usando conversaciones, en cierta medida, las mismas cumplan con las propiedades ACID, lo que le otorga previsibilidad y facilidad de lectura al código fuente de la aplicación**. A continuación se verá cómo se pueden conseguir estas propiedades cuando se utilizan conversaciones en aplicaciones transaccionales.

Atomicidad

Que la conversación sea atómica, implica que el usuario de la aplicación pueda abortar la misma sin consecuencias permanentes, y se pueda realizar un *rollback* de los cambios realizados. Cuando la conversación salva todos los datos de la misma en su último paso, esto no es

problemático ya que este último evento realiza la transacción de todos los cambios contra la base de datos; los pasos previos solo cargan datos para mostrarlos. Sin embargo, conversaciones más complejas son posibles, donde se realicen transacciones contra la base de datos en un paso intermedio, de modo que si el usuario decide abandonar la conversación, deben gestionarse acciones compensatorias para restaurar la consistencia en la base de datos. Por supuesto, este tipo de situaciones muchas veces pueden evitarse, especialmente utilizando un esquema de gestión del contexto de persistencia extendido (Sección 3.8.5, "Conversaciones usando Contexto de Persistencia Extendido").

Consistencia

La propiedad de Consistencia debe gestionarse con cuidado ya que, a menos que el motor de base de datos ya tenga creadas las reglas de integridad de los datos (claves externas, chequeos de claves únicas por tablas, triggers, etc.) las mismas deben realizarse programáticamente desde el código fuente de la aplicación, y este código básicamente es independiente de la tecnología de manipulación de los datos, o sea que el uso de un ORM o de JDBC es indistinto desde el punto de vista del desarrollo. Por ej. antes de hacer un `INSERT` dentro de una tabla, debe chequearse mediante la ejecución de un `SELECT` SQL que la nueva tupla no duplique una clave primaria ya existente en la tabla, o alguna otra clave alterna que deba ser única.

Aislamiento

El Aislamiento debe gestionarse desde el código fuente de la aplicación, ya que comúnmente los motores de bases de datos no están configurados por defecto para operar en su máximo nivel de aislamiento.[1] Ya que la aplicación Web no interactúa con la base de datos mientras el usuario está pensando o escribiendo datos en un formulario, no es una buena práctica usar *pessimistic locking* porque no es escalable. Lo que se hace es *optimistic locking*, que significa que no se previene que dos usuarios puedan operar sobre los mismos datos al mismo tiempo, pero se espera que esto no ocurra frecuentemente. Cuando ocurre, se debe tener una estrategia de resolución de conflictos, de modo que cuando se realiza un `UPDATE` o `DELETE` de una tupla en una tabla, se chequea contra un número de versión[2] que permite determinar si el item a actualizar ha sido modificado por otro usuario y, en tal caso, poder emitir un mensaje de alerta, cancelando así la transacción.

Durabilidad

La propiedad de Durabilidad de las transacciones es exclusivamente dependiente del motor de base de datos: Si la aplicación recibe una respuesta exitosa de que el motor ha realizado el COMMIT de la transacción, debe interpretarse como tal sin dudar.

3.8.2. Soporte de ORM en frameworks MVC

Muchos frameworks de desarrollo web no están específicamente diseñados para utilizar ORM, ya que los mismos gestionan solamente el ciclo de una petición HTTP y su respuesta, dejando la gestión de interacciones que abarcan múltiples peticiones a cargo del desarrollador.

Por ejemplo, frameworks populares como Struts o Spring MVC no gestionan el contexto de persistencia durante todo el ciclo de vida de la petición HTTP, ya que es el desarrollador quien debe gestionar la persistencia y por lo tanto, comúnmente el contexto de persistencia se abre dentro de la ejecución de un Action y se cierra antes de finalizar el mismo, de modo que la etapa de renderizado de la respuesta en una página JSP se realiza con el contexto de persistencia ya cerrado, lo que produce muchas veces la tan común `LazyInitializationException`.[3]

[1] De menor nivel de aislamiento a mayor, los distintos niveles provistos son: *Read uncommitted*, *Read committed*, *Repeatable reads* y *Serializable*.

[2] Normalmente, es un atributo adicional de la tabla en cuestión que se va incrementando a medida que se van realizando modificaciones, pero bien podría ser otro atributo del dominio de la aplicación que sirva al mismo propósito (*i.e.* poder determinar si la tupla ha sido modificada o no).

[3] Es una excepción producida por Hibernate y JPA (con Hibernate como proveedor de persistencia), que indica que se ha solicitado acceso a un objeto que no ha sido cargado en el contexto de persistencia, y como el mismo ya ha sido cerrado, es imposible hacerlo en este momento.

3.8.3. Patrón de diseño *Open Session In View*

Una de las formas de evitar la aparición de las `LazyInitializationException` es utilizar el patrón de diseño *Open Session In View* [23] (abreviado OSIV). Este patrón de diseño se implementa comúnmente con un filtro web declarado en el archivo `web.xml` para que se ejecute *antes* del servlet principal del framework MVC que se esté utilizando. Al ejecutarse antes, el filtro intercepta todas las peticiones HTTP dirigidas hacia el servlet de la aplicación web, y por lo tanto puede abrir el contexto de persistencia para que esté disponible durante todo el ciclo de vida de la petición, inclusive la etapa de renderizado de la respuesta. Al terminar el renderizado de la respuesta, el servlet principal del framework retorna el control al filtro web, el cual hace el commit si es necesario de las modificaciones realizadas al contexto de persistencia y luego lo cierra. Así, la excepción `LazyInitializationException` nunca aparece.

Un punto a tener en cuenta con el filtro OSIV es que se debe tener cuidado con los errores al acceder a la base de datos. Si el commit de la transacción (que se realiza luego de renderizar la página de resultados) produce una violación de una *constraint* de la base de datos, la transacción no podrá completarse, pero la página de respuesta posiblemente ya habrá sido enviada al navegador del usuario operando la aplicación. Una forma de resolver este problema es hacer el `flush` del contexto de persistencia antes de renderizar la página, o bien agrandar el tamaño del buffer de envío de respuestas en el contenedor web para que sea menos probable que se envíe una respuesta no deseada. Muchos frameworks MVC renderizan la respuesta en un buffer interno y luego se envía el mismo como respuesta en el contenedor web, lo que evita el problema completamente.

La principal limitación del filtro OSIV es que el mismo actúa en forma independiente entre una petición HTTP y la siguiente, lo que implica que dos peticiones consecutivas utilizarán **diferentes contextos de persistencia**. En otras palabras, no se preserva el estado conversacional entre una petición y la siguiente, sin embargo es posible hacerlo con alguno de los dos siguientes esquemas: el primero es almacenando en la conversación objetos en estado *detached*, y el otro extendiendo el alcance del contexto de persistencia para que abarque la conversación entera.

3.8.4. Conversaciones con objetos en estado *detached*

Consideremos un caso de uso típico donde una petición carga los datos que son mostrados en un formulario para su modificación, y otra petición donde se graban los mismos a la base de datos. En esta situación, el objeto de dominio en cuestión queda en estado *detached* luego de que se haya cerrado el contexto de persistencia de la primera petición; al enviar el formulario, se debe hacer un *merge* del objeto dentro del nuevo contexto de persistencia, lo que podría provocar una consulta a la base de datos para determinar si el objeto en cuestión sigue existiendo o si algún otro atributo del mismo a sufrido modificaciones en la base de datos (a efectos de implementar *optimistic locking* dentro de la aplicación).

Recordemos que en JPA e Hibernate, el estado *detached* significa que el objeto en cuestión es persistente dentro de la base de datos, pero el contexto de persistencia (objeto `EntityManager` en JPA, o `Session` en Hibernate) que lo ha cargado en memoria ya se ha cerrado, con lo cual ya no está asociado al mismo y por lo tanto ya no se puede garantizar que el objeto esté sincronizado con la base de datos. El objeto puede ser modificado, pero para guardar esos cambios a la base de datos nuevamente, debe hacerse una operación de *reattachment* o *merge* dentro de un nuevo contexto de persistencia. Estas operaciones chequean los valores actuales contra los almacenados en la base de datos (mediante alguna consulta SQL) para así obtener un nuevo objeto sincronizado con la base de datos (y los datos modificados previamente, preservados).

En la Figura 3.2 se muestra un esquema denominado *session-per-request-with-detached-objects* [7], de cómo serían las interacciones que suceden cuando se tiene una interacción como la descrita arriba, es decir una conversación de dos peticiones. En la primer petición, el objeto que se desea modificar es cargado desde la base de datos (el círculo con la letra P) que se encuentra en estado Persistente. Como se puede apreciar, en la primer petición HTTP se crea una `Session` (de Hibernate) asociada a una Transacción de base de datos y a su contexto de persistencia, las cuales tienen la misma duración que la petición: al terminar de renderizar la respuesta, se hace un commit de la transacción, y se cierran la Session y su contexto de persistencia, por lo tanto el objeto cargado (guardado en una conversación o en la Sesión HTTP del usuario) queda en estado *detached*.

En la segunda petición, se vuelve a crear una nueva `Session`, transacción y contexto de

Let's create the conversation with native Hibernate interfaces and a detached object strategy. The conversation has two steps: The first step loads an object, and the second step makes changes to the loaded object persistent. The two steps are shown in Figure 11.2:

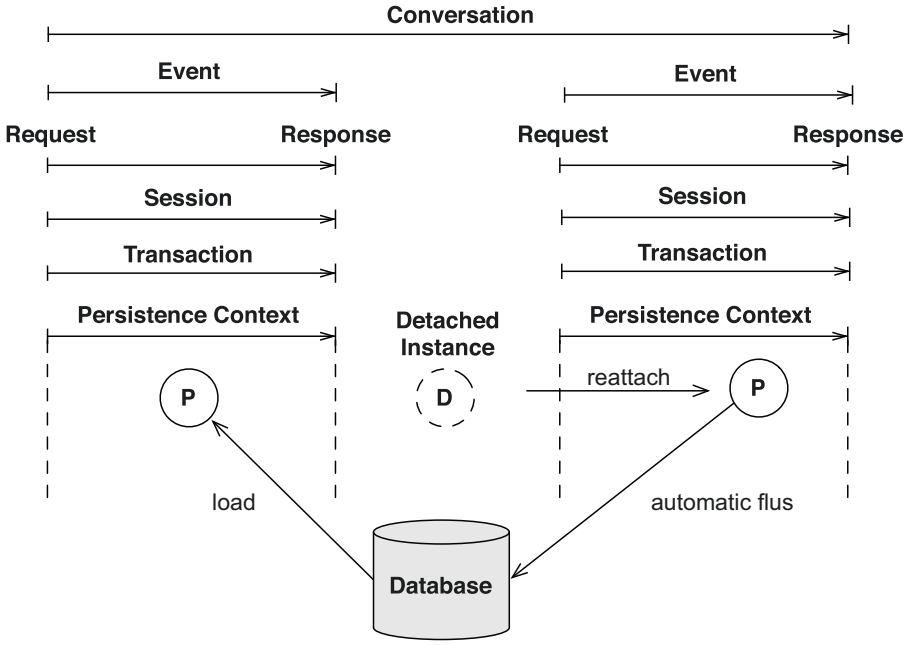

Figure 11.2 A two-step conversation implemented with detached objects

Figura 3.2: Una conversación de dos pasos implementada con objetos en estado *detached*. (Extraída de [7].)

persistencia, con lo cual el desarrollador debe llamar a alguno de los métodos `merge(Object)`, `update(Object)` o `saveOrUpdate(Object)` de la `Session` de Hibernate (los métodos varían en algunos detalles) para adjuntarlo en estado Persistente al nuevo contexto de persistencia, planificando una sentencia `SELECT` y luego un `UPDATE` de SQL. Cuando la petición termina, se hace el COMMIT de la transacción y el objeto es persistido en la base de datos con las modificaciones realizadas por el usuario.

No es complicado implementar la conversación con transacciones utilizando objetos en estado *detached*, pero el usuario espera que la transacción que intenta representar la conversación esté aislada de las demás, y que sea atómica. Como se observó antes, el aislamiento se consigue utilizando *optimistic locking*, de modo que si dos conversaciones intentan modificar la misma pieza de información, ocurrirá una excepción y alguna de ellas se cancelará. No obstante, la atomicidad se verá afectada toda vez que el contexto de persistencia se cierre y haga un *commit* de la transacción en un paso que no sea el último de la conversación; en otras palabras, la conversación producirá transacciones largas en forma atómica siempre y cuando las modificaciones a la base de datos se produzcan durante la última petición HTTP e inmediatamente se cierre la conversación para evitar posibles modificaciones futuras indeseadas.

3.8.5. Conversaciones usando Contexto de Persistencia Extendido

Una estrategia diferente, que no involucra el uso de objetos en estado *detached* es **extendiendo** el contexto de persistencia para que abarque la conversación completa. Este esquema es conocido como *session-per-conversation*, que se muestra en la Figura 3.3, pero depende de que el contexto de persistencia tenga inhabilitado el *flush* automático de los objetos asociados al contexto de persistencia.

Tanto en JPA como en Hibernate, las operaciones de creación, actualización y borrado de objetos no son inmediatamente ejecutados contra la base de datos (usando sentencias SQL `INSERT`, `UPDATE` o `DELETE`) cuando se ejecuta una instrucción del framework de ORM, sino que las mismas son encoladas dentro de una cola que gestiona el contexto de persistencia. Por defecto, esta cola es sincronizada con la base de datos (es decir, las sentencias SQL son ejecutadas) cuando ocurre un evento importante: se hace el *commit* de la transacción, o bien antes de hacer una consulta a la base de datos (entonces la consulta va a retornar los datos actualizados). Sin embargo, existe una forma de inhabilitar este comportamiento, para que este *flush* a la base de datos se realice solo a pedido del desarrollador, cuando se ejecuta el método `flush()` del ORM,

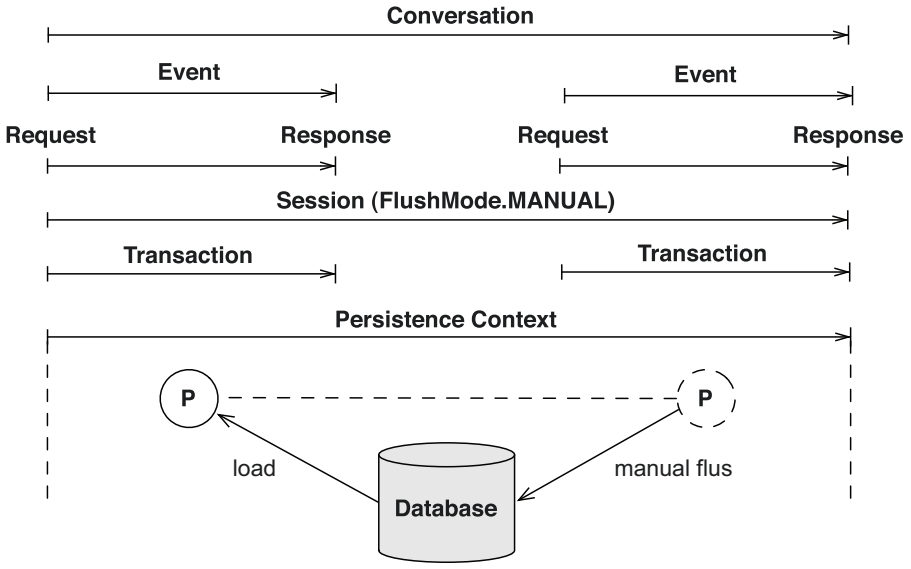

Figure 11.3 A disconnected persistence context extended to span a conversation

Figura 3.3: Contexto de persistencia extendido para abarcar una conversación. (Extraída de [7].)

setting a `FlushMode.MANUAL`—you should do this when the conversation begins and the `Session` is opened.

Modifications made in concurrent conversations are isolated, thanks to optimistic locking and Hibernate's automatic version checking during flushing. Atomicity of the conversation is guaranteed if you don't flush the `Session` until the last step, the end of the conversation—if you close the unflushed `Session`, you effectively abort the conversation.

We need to elaborate on one more point in this discussion: the topic of insertion of new entity instances. Note that this isn't a problem in this example, but it's something you'll have to deal with in more complex conversations.

Delaying insertion until flush-time

To understand the problem, think about the way objects are saved and how their identifier value is assigned. Because you don't save any new objects in the CompleteAuction conversation, you haven't seen this issue. But any conversation in which you save objects in an intermediate step may not be atomic.

The `save()` method on the `Session` requires that the new database identifier of the saved instance be returned. So an identifier value has to be generated when the `save()` method is called. This is no problem with most identifier generator strategies; for example, Hibernate can call a `sequence`, do the in-memory `increment`, or ask the `hilo` generator for a new value. Hibernate doesn't have

y es lo que se usará para hacer funcionar este esquema de *session-per-conversation*.

Retomando la Figura 3.3, cuando se crea una nueva conversación, se abre un nuevo contexto de persistencia (un objeto del tipo `EntityManager` si usamos JPA, o bien uno de tipo `Session` de Hibernate), inhabilitado el *flush* automático del mismo (`FlushMode.MANUAL`). Durante el primer paso, el objeto persistente se carga en una primer transacción de base de datos. Cuando se cierra la conexión de base de datos subyacente (una `Connection` de JDBC), este `EntityManager` (o `Session`) queda desconectado de la misma, pero el contexto de persistencia asociado puede seguir siendo usado para otros propósitos, en particular, al tratar con un dato de la misma conversación recién creada. Cuando el usuario realiza la próxima petición HTTP —retomando la conversación— se obtiene el contexto de persistencia guardado en la misma, y se asocia a una nueva conexión a la base de datos, comenzando una nueva transacción. Cualquier objeto que se haya cargado en el contexto de persistencia está en estado persistente: nunca pasa a estado *detached*. Luego, cualquier modificación realizada a los objetos persistentes son sincronizadas con la base de datos en el momento en que se llame al método `flush()` del `EntityManager` (o `Session`).

Al igual que con las conversaciones con objetos en estado *detached*, las modificaciones realizadas en conversaciones concurrentes están aisladas, usando *optimistic locking* durante el *flush*. La atomicidad (que no siempre se podía garantizar con el esquema anterior) ahora si está garantizada, siempre que no se realice un `flush()` del `EntityManager` de JPA (o `Session` en Hibernate) hasta el último paso (el fin de la conversación). Si se cierra el contexto de persistencia sin hacer el `flush()` (por ej. si se aborta o expira la conversación), entonces no se realizarán los cambios en la base de datos.

Ya que no se necesitan hacer `merges` de objetos en diferentes métodos que manipulan los objetos persistentes (en definitiva, los objetos del dominio de la aplicación), este esquema de extender el contexto de persistencia es más fácil de entender que las aplicaciones que utilizan el esquema con objetos en estado *detached*: el código fuente es más simple y por lo tanto más mantenible, y ofrece de por sí mejores garantías en cuanto a la atomicidad de las conversaciones.

3.9. Modelos extendidos de conversaciones

Existen dos modelos que extienden las funcionalidades de las conversaciones que son de uso común, por un lado están los frameworks que soportan el modelo de **conversaciones anidadas**, y por otro lado están los que soportan **continuations**.

3.9.1. Conversaciones anidadas

Las conversaciones hacen simple el mantener la consistencia de estado en una aplicación web, aún frente a la operación de la misma en múltiples ventanas o presionando el botón de Retroceso del navegador. Desafortunadamente, simplemente comenzar y terminar una conversación no es suficiente; dependiendo de los requerimientos de la aplicación, pueden producirse inconsistencias entre lo que el usuario espera y el estado real de la aplicación.

Consideremos nuevamente el ejemplo de la reserva de hotel. Ahora queremos extender la aplicación para que el usuario que desea reservar un hotel, pueda seleccionar en cual habitación del hotel desea alojarse, entonces la secuencia de pasos que debe hacer el usuario queda así:

1. El usuario busca un hotel donde alojarse en una ciudad y en cierta fecha.
2. El sistema muestra un listado de posibles hoteles que cumplen con ese criterio.
3. El usuario selecciona un hotel del listado, posiblemente en otra pestaña del navegador, creando una nueva conversación.
4. El sistema presenta los datos del hotel seleccionado, junto con un listado de habitaciones disponibles, y su precio por día.
5. El usuario selecciona una habitación de ese listado, posiblemente en otra pestaña del navegador.
6. El sistema muestra una pantalla con información detallada de esa habitación, con un botón de Confirmación de Reserva.
7. El usuario presiona el botón de Confirmación de reserva.
8. El sistema muestra un formulario donde el usuario debe ingresar los datos de tarjeta de crédito para concretar la reserva.
9. El usuario completa el formulario, y presiona el botón de Pago.
10. El sistema muestra una pantalla de confirmación de pago, terminando así la conversación.

El problema se presenta cuando el usuario ejecuta el paso 5. Aquí, el usuario selecciona una de las habitaciones posibles, haciendo que al recibir el request, el sistema cargue los detalles de esa habitación desde una base de datos, para luego almacenarlos en la misma conversación. Sin embargo, si el usuario abre varias habitaciones en diferentes pestañas del navegador, la habitación que realmente quedará almacenada en la conversación es la última seleccionada, ya que las anteriores se perderán por estar salvadas bajo la misma clave en la misma conversación. En otras palabras, **se tiene nuevamente el problema que originalmente se tenía con los datos almacenados en la sesión de usuario, sólo que ahora se presenta en el contexto de una conversación**.

La manera de solucionar este problema es **anidar** una nueva conversación dentro de otra, de manera que la conversación hija (*i.e.* la que quede anidada) sea la que referencia a la habitación, y la conversación padre (*i.e.* la que anida a otras conversaciones) sea la que referencia el hotel. Entonces, de manera análoga a cuando seleccionamos un hotel, cuando se selecciona una habitación se *crea una nueva conversación* especificando en el framework MVC que estará anidada a la conversación actual.

El anidamiento de conversaciones tiene reglas de visibilidad bastante intuitivas: Supongamos en forma general que la conversación H está anidada dentro de la conversación P, entonces si el `Action` se ejecuta en el contexto de la conversación H:

1. Desde el `Action` son visibles tanto los datos almacenados en H como en P.
2. Los datos almacenados en H son de lectura y escritura.
3. Los datos almacenados en P son de *sólo lectura*.

La Regla Nro. 3 se debe a que de poder modificar los datos de la conversación padre, estaríamos ante la posibilidad de modificar el mismo dato desde distintas ventanas o pestañas, precisamente la situación que se intenta evitar.

Pila de conversaciones

Como las conversaciones pueden crearse con múltiples niveles de anidamiento, las reglas de visibilidad se pueden generalizar viendo a las conversaciones *apiladas* unas sobre otras, dando lugar a lo que se conoce como una **pila de conversaciones**. Cuando al framework MVC se le solicita el valor de una variable guardada en una conversación, éste comienza por buscar en la conversación actual (la que está en el tope de la pila) y sigue buscando en las subsiguientes conversaciones apiladas en caso de que no se encuentre, hasta llegar a la **conversación raíz**, que es la última conversación de la pila. Luego, en caso de no encontrar el dato en ese scope, prosigue buscando en la sesión de usuario.

Además, otras conversaciones pueden existir concurrentemente, apiladas sobre las mismas conversaciones padre, lo que permite guardar un estado independiente sobre cada una de ellas. Por ej. en la aplicación de reserva de hoteles seleccionando una habitación en particular, si el usuario (en el paso 5) abre varias pestañas con distintas habitaciones sobre el mismo hotel, cada una de ellas será una conversación anidada, teniendo ambas como padre a la conversación con el mismo hotel.

Terminación de conversaciones

La terminación de las conversaciones anidadas también terminan las conversaciones que están antes en la pila de conversaciones, entonces si por ej. tenemos una pila con las conversaciones $\{C_1, C_2, C_3\}$, donde C_1 es la conversación raíz y C_3 es la conversación corriente, entonces terminando C_2 también hace que se termine la conversación C_3 en forma automática, y haciendo que el sistema retome la conversación C_1.

De la misma forma, todas las conversaciones anidadas concurrentes se verán afectadas por la terminación de una conversación compartida en sus pilas: Si un contexto de ejecución tiene una pila $\{C_1, C_2, C_3\}$, y otro contexto una pila $\{C_1, C_2, C_4\}$, terminando la conversación C_2 hace que tanto la conversación C_3 y C_4 sean terminadas también.

Finalmente, terminando la conversación raíz termina todas las conversaciones anidadas en forma recursiva que se han creado a partir de ella. Esto es lo que haríamos en el ejemplo de reserva de habitaciones cuando se ha terminado de procesar el pago de la reserva.

Ventajas y desventajas de las conversaciones anidadas

Como ventaja principal con respecto a un framework con soporte de conversaciones de un único nivel es la posibilidad de representar casos de uso más complejos, donde existen dependencias entre las conversaciones (como en el ejemplo de la reserva de hoteles seleccionando un tipo de habitación). Este poder intrínseco proporcionado por las conversaciones anidadas de alguna forma queda opacado por el hecho de que estas funcionalidades combinadas son complicadas para usuarios comunes: Imaginemos un sistema de reserva de pasajes de aviones, donde el usuario puede buscar y seleccionar un vuelo, luego puede buscar y seleccionar un hotel en la ciudad de destino del vuelo, luego pueda seleccionar el tipo de habitación, y luego buscar, seleccionar y alquilar un automóvil: si bien el modelo de conversaciones anidadas soporta tal funcionalidad en una forma muy elegante, difícilmente se le pueda pedir al usuario de tal sistema que concluya una transacción de este tipo pasando por todas las etapas.

Como desventaja se puede notar que en una implementación sencilla (donde una conversación almacena una referencia a su conversación padre) es muy difícil controlar que el acceso a la conversación padre sea de solo lectura, ya que es posible acceder y modificar los datos utilizando distintos caminos, por ej. accediendo al objeto que representa la conversación padre, obtener una referencia a un objeto almacenado en esa conversación, y llamar a métodos que modifiquen el estado interno del mismo. Entonces, básicamente este control queda en manos del desarrollador, y por lo tanto se debe testear muy bien el código que haga uso de conversaciones anidadas.

3.9.2. Continuations

Una *continuation* [38] es un concepto que proviene originalmente de los lenguajes de programación funcionales, y representa una instancia de un proceso de computación en un punto particular en la ejecución de ese proceso. Una continuation contiene información tal como la pila

```
1 public class Prueba {
2   public Continuation execute() {
3     int i = 0;
4     while(i < 5) {
5       print(i);
6       return new Continuation(this); // interrumpe ejecucion
7       i++;
8     }
9   }
10 }
11
12 Prueba p = new Prueba();
13 Continuation c1 = p.execute(); // imprime 0
14 c1 = c1.continue(); // imprime 1
15 c1 = c1.continue(); // imprime 2
16 Continuation c2 = p.execute(); // imprime 0
17 c1 = c1.continue(): // imprime 3
18 Continuation c3 = c2.continue(); // imprime 1
19 c2 = c2.continue(); // imprime 2;
20 c2 = c2.continue(); // imprime 3;
21 c2 = c2.continue(); // imprime 4;
22 c2 = c2.continue(); // termina el procesamiento
23 c3 = c3.continue(); // imprime 2
24 c3 = c3.continue(); // imprime 3
```

Listado 3.3: Continuations: ejemplo ilustrativo

de llamadas actual del proceso (incluyendo todos los datos al alcance del proceso, por ej. variables locales) así como el punto en el que se interrumpe la ejecución. Esta continuation luego puede ser retomada cuando se invoca, "continuando" de esta manera la ejecución del proceso.

Como analogía para ayudar a entender esta abstracción, podemos nombrar los Sistemas Operativos multi-tarea (como Unix, Linux o Windows) donde múltiples procesos se ejecutan utilizando un solo CPU: estos procesos ejecutan durante un breve momento hasta que se termina su *slice* de tiempo o bien quedan bloqueados esperando algún evento: cuando sucede esto el Sistema Operativo salva el estado del proceso en una estructura de datos llamada *Process Control Block* (PCB) [40] con los registros del CPU, el puntero a la pila, el puntero de próxima instrucción (*Program Counter*) y otra información[4], para que en el futuro pueda retomar el mismo proceso en el estado en que estaba cuando se interrumpió su ejecución.

Para aclarar un poco más el concepto de continuation, imaginemos que tenemos un método como el del Listado 3.3, en un lenguaje con una sintaxis *similar* a Java.[5] Cuando algún objeto llama al método `execute()`, éste se ejecuta hasta llagar a la línea 6, donde se retorna del método[6] creando una nueva continuation, lo que hace que la ejecución de ese método sea interrumpida y haciendo que dentro de la continuation quede grabada la pila de ejecución actual, junto con las variables al alcance del método —entre ellas `i`— que serán salvadas para posterior uso. En algún momento posterior, algún objeto puede retomar esa continuation, lo que hará que el método `execute()` retome su ejecución desde la línea 7, restaurando previamente la pila de ejecución y variables, tal cual estaba en el momento de crear esa continuation. El efecto es que el método continua su ejecución como si nada hubiese pasado, dentro del mismo contexto en el que estaba ejecutando antes de crear la continuation. De manera análoga, cada ciclo del loop dentro del método crea una nueva continuation, haciendo que algún cliente pueda retomar *cualquiera de ellas*. Las líneas 12 a 24 del Listado 3.3 ilustran estos casos de ejemplo de ejecución.

En aplicaciones web las continuations pueden ser usadas porque abstraen muy bien la naturaleza stateless de HTTP, ya que cada request realiza una operación independiente de otro, y cada uno de estos requests modifica el estado de la aplicación dentro del servidor, por lo tanto puede utilizarse para abstraerse del ciclo de request/response.

La idea es que cada vez que el servidor de aplicaciones va a renderizar una respuesta, previamente salva el estado actual de la aplicación en una *nueva* continuation, y cuando el servidor recibe un request desde el navegador web, el servidor restaura la continuation reanudando la ejecución de la misma utilizando el estado previo del servidor. Como cada continuation es inde-

[4] Otra información salvada: el identificador de proceso, el espacio de direcciones utilizada por el proceso, la prioridad, lista de archivos abiertos, dispositivos de E/S en uso, etc.

[5] Java como lenguaje no posee soporte nativo de continuations.

[6] Nuevamente, esto no es código Java válido!

pendiente una de otra, el usuario puede presionar el botón de Retroceso en el navegador sin efectos laterales en la aplicación, ya que al hacer un request desde una página ya procesada, el servidor web restaurará la misma continuation que ha sido generada para esa respuesta en particular.

Identificación de continuations

De manera similar a las conversaciones, se requiere que la respuesta generada por el servidor de aplicaciones contenga un *identificador de continuation*, que deberá ser incorporado en la respuesta como un campo oculto o bien como parámetro de una URL. De esta manera cuando la aplicación recibe un request puede deducir a cual continuation pertenece a efectos de restaurarla para que la aplicación quede en el estado en que estaba previamente.

A diferencia de las conversaciones, que en ocasiones conviene identificarlas con una clave natural como se describió en la Sección 3.3.2, no tiene mucho sentido darle un identificador natural a una continuation.

Identificación de conversaciones *lógicas*

Como cada continuation guarda el estado de la aplicación en forma independiente una de otra, es imposible distinguir un doble envío de un formulario hacia el servidor, dado que la misma continuation será reanudada dos veces sin poder determinar si ya ha sido enviado o no un request previo. En otras palabras, no hay ninguna relación entre las continuations y el caso de uso que el usuario está ejecutando, o sea las continuations no conocen nada con respecto a la conversación lógica que está llevando a cabo el usuario con la aplicación.

Para detectar múltiples envíos del mismo formulario al servidor, se requiere de una conversación bajo la cual se engloben todas las continuations generadas para esa conversación lógica, hasta que el usuario termine de ejecutar el caso de uso y le dé fin al mismo, terminando la conversación y con ella todas las continuations generadas. Con esta conversación el múltiple envío de datos es detectable ya que cuando se procesa por segunda vez, la conversación no será encontrada.

El identificador en este caso es una clave compuesta, integrada tanto por el identificador de conversación como por el identificador de la continuation.

Representación de la lógica de navegación

Si pensamos que cada vez que un usuario hace un request dentro de un caso de uso que abarque múltiples pantallas (por ej. en un wizard) la aplicación cambia de estado, podemos establecer una correspondencia uno a uno entre los estados y transiciones dentro de una máquina de estados finitos, con las pantallas y acciones generadas por el usuario. Cada request genera un posible cambio de estado en la aplicación y la misma genera una continuation para salvarlo hasta que llegue el próximo request, donde se reanudará su ejecución. Esto último es análogo a la ejecución de una máquina de estados, donde la máquina produce una respuesta distinta dependiendo del estado en que se encuentre y los eventos de entradas (datos proporcionados en el request).

Existen frameworks como SWF [36] que permiten directamente operar sobre tal abstracción, la cual contiene el flujo de navegación de la capa de presentación de la aplicación (Sección 3.4.2).

Ventajas y desventajas

La principal ventaja es que un framework que soporte el uso de continuations permite tener una navegación completamente controlada, mientras que le permite al usuario utilizar todas las ayudas de navegación proporcionadas por el navegador web. Por otro lado las continuations permiten al programador expresar la lógica de navegación de un caso de uso en forma sencilla, ya que las mismas permiten definir una máquina de estados por los cuales va pasando el caso de uso sin tener que pensar en los detalles de implementación de la misma.

La principal desventaja de las continuations es una gran utilización de memoria, la cual es causada por el almacenamiento de las múltiples continuations que se van creando por cada request del usuario. Los frameworks que soportan continuations normalmente permiten definir la cantidad máxima de continuations que se pueden almacenar por conversación a efectos de

limitar el consumo de memoria: cuando una conversación recibe un nuevo request que crea una nueva continuation (y ya se ha alcanzado el límite máximo) entonces debe descartar una de ellas para poder almacenar la última continuation creada (típicamente se descartan en orden FIFO). El descarte de una continuation implica que el usuario ya no puede ir hacia atrás en el historial de navegación para reanudar el procesamiento desde allí.

Asimismo, el límite máximo de continuations creadas tiene utilidad para evitar posibles ataques de negación de servicios.

3.10. Problemas de gestión de estado y soluciones usando conversaciones

Aquí retomaremos los problemas de gestión del estado de una aplicación web, los cuales fueron introducidos en la Capítulo 2, Sección 2.2 pero con el enfoque puesto en cómo las conversaciones pueden ayudar al desarrollador a solucionar estos problemas en forma más sencilla que las planteadas previamente.

3.10.1. Acceder a un proceso desde un enlace Favorito

El problema se presenta cuando se accede a una URL previamente almacenada en los Favoritos del navegador, y este enlace se conecta con cierta parte de un caso de uso que no ha sido iniciado en forma correcta. Para detectar si una URL que hace un request está accediendo a cierta funcionalidad de la aplicación a la cual no está permitida en ese momento, se puede grabar un objeto dentro de la conversación que indique si la misma ha comenzado correctamente o no: Si no se encuentra previamente almacenado ese objeto dentro de la conversación significa que el request es ilegal y por lo tanto corresponde redirigir al usuario hacia una pantalla de error.

En efecto, esta solución sería un Synchronizer token [1] en caso de que el objeto almacenado en la conversación sea ficticio, pero más comúnmente se podrá utilizar algún objeto del dominio del problema en su lugar, el cual contendrá la información suministrada por el usuario a medida que se va desenvolviendo el caso de uso en las distintas pantallas.

Por otro lado, aunque el Synchronizer token sea ficticio, el mismo es mejor almacenarlo en el contexto de una conversación que en la sesión de usuario, ya que el alcance y tiempo de vida del mismo será más restringido, y además permite tener tokens independientes por ventana, en caso de que el usuario abra más de una ventana con el enlace Favorito.

Por último, si la aplicación inicia la secuencia de pantallas del caso de uso creando una nueva conversación, entonces no será necesario guardar ningún dato en la misma para poder detectar un request inapropiado, ya que el objeto `Action` que procesa el request puede chequear directamente si ya hay una conversación creada previamente, o bien si ha creado durante el procesamiento del request actual, con lo cual no hace falta realmente guardar ningún dato: una conversación recién creada (o aún no creada) implica un request inválido.

3.10.2. Pedido de refresco (botón Recargar)

Aquí el problema es que al presionar el botón de Refresco del navegador luego de haber completado el caso de uso y de haber recibido una pantalla de confirmación, se hará un doble envío de la misma información al servidor, lo que puede causar errores inesperados por el usuario, como problemas de integridad referencial dentro de una base de datos relacional.

La culminación de una secuencia de pantallas de la aplicación instruye a la misma a realizar un procesamiento final como una transacción en una base de datos, enviar un email, etc. En este caso el uso de las conversaciones son muy útiles por dos factores: en primer lugar, es muy fácil implementar un patrón POST-REDIRECT-GET [25] ya que la mayoría de los frameworks con soporte nativo de conversaciones también soportan este patrón en forma nativa, de modo que usando este patrón de diseño se evita que el navegador muestre la confusa pantalla de advertencia de Refresco de datos (Figura 2.1) en forma sencilla. En segundo lugar, al completar el procesamiento la conversación es automáticamente destruida (o marcada para su destrucción luego del próximo request en caso de que se desee hacer un POST-REDIRECT-GET) con lo cual un segundo request sobre la misma acción es detectable como inválido dado que la conversación ya no existe).

3.10.3. Retroceder y Avanzar en el historial de navegación

Cuando un usuario presiona el botón de Retroceso en el navegador, el estado de la aplicación que ve la persona no es el que corresponde con el que tiene el servidor, sin embargo las conversaciones ayudan enormemente a preservar un estado coherente de la aplicación web del lado del servidor. Desde el punto de la realización de una transacción existen dos posibilidades: La primera se presenta cuando la conversación no ha finalizado; aquí no se ha grabado ningún objeto dentro de la base de datos, entonces un re envío de un formulario solo sobre-escribirá (en memoria) los datos anteriores lo cual es comúnmente benigno para la aplicación. La segunda se presenta cuando la conversación ya ha terminado y los datos han sido salvados en la base de datos, aquí el re envío de un formulario será descartado por el framework ya que detectará que esa conversación ya no existe.

Si bien cuando el operador sólo tiene abierta una única ventana para usar la aplicación, hacer un reenvío de una página ya navegada puede no generar mayores problemas, cuando se introduce la posibilidad de ejecutar la misma desde múltiples ventanas puede generar una desincronización importante ya que el último estado almacenado en la sesión de usuario se verá comprometido por las múltiples fuentes de requests operando sobre la misma funcionalidad.

Es por eso que es mucho más conveniente almacenar el estado de la aplicación a nivel de conversación que en la sesión de usuario, ya que cada una de ellas puede mantener un estado diferente en forma independiente una de otra.

Asimismo, los modelos extendidos de conversaciones proveen una protección mayor ya que representan una forma aún más granular de almacenar el estado de la aplicación. Los frameworks que soportan conversaciones anidadas toleran diferentes niveles de aislamiento para el almacenamiento de los datos, mientras que los que soportan continuations proveen una independencia total entre un estado de la aplicación y otro, al punto de que (en un modelo puro de continuations) no sería posible detectar un doble envío de un formulario, ya que cada continuation contiene todo el estado del servidor.

3.10.4. Múltiple envíos de datos

Para detectar los múltiples envíos de formularios que puede llegar a hacer el usuario es necesario almacenar algún dato de manera que se pueda comparar el primer envío de otros posteriores. Para ello el patrón Synchronizer Token implementado en el contexto de la conversación ofrece el alcance ideal para resolver el problema.

Si la funcionalidad que se quiere proteger de múltiples envíos es la que se ejecuta en el servidor y cuyo éxito dispara la destrucción de la conversación, ésta información por si misma ya alcanza para detectar el múltiple envío, ya que la segunda vez que se recibe el request la conversación no existirá más. Esta solución no sería posible implementarla utilizando datos almacenados en la sesión de usuario (al menos, no sin eliminarlos manualmente al terminar el request).

Por último, el patrón POST-REDIRECT-GET ayuda enormemente a que no se produzcan múltiples envíos, ya que el formulario que da origen al request queda eliminado del historial de navegación.

3.10.5. Múltiples ventanas de navegación

Ante la presencia de funcionalidad corriendo desde múltiples ventanas, no es posible proteger los datos de la aplicación en el servidor cuando los mismos están almacenados en la sesión de usuario, ya que todas las ventanas comparten la misma sesión.

Además, la opción de reescribir el estado de la aplicación en cada una de las páginas generadas como respuesta a un request exige un trabajo mayor por parte del programador ya que debe de alguna forma *serializar* los mismos para poder luego recuperarlos al recibir el próximo request de la misma ventana: dicha tarea manual es muy susceptible a errores, o bien podría instalarse un procedimiento que serialize y deserialize el estado de la aplicación en forma automática, pero esto hace que se requieran especificaciones más rigurosas para los objetos a mantener (por ej. que implementen la interface `java.io.Serializable` o `java.io.Externalizable`).

La opción más sencilla y confiable es mantener el estado de la aplicación en distintas conversaciones, cada una asociada a una ventana en particular. Este estado es propagado de página en página reescribiendo solamente el identificador de la conversación que los contiene. Si se

compara la reescritura del identificador con el estado entero del servidor en forma serializada, vemos que la primera opción no impone restricciones adicionales a los objetos mantenidos en ellas.

3.11. Resumen

En este Capítulo hemos abordado de lleno en las conversaciones, definiendolo como un espacio de almacenamiento en el cual la aplicación puede guardar su estado, o más específicamente como un nuevo scope. Las conversaciones pueden ser restablecidas entre request y request debido a que su contexto es propagado utilizando un identificador de conversación. Los frameworks pueden exponer distintos niveles de abstracción para que el programador pueda trabajar con conversaciones: desde un lenguaje de expresión utilizable desde distintos medios de presentación, una definición formal del flujo de información y estados por los que puede pasar el caso de uso, hacer inyección de dependencias, o la grabación de las mismas haciendo outjection o bijection, hasta permitir al programador aprovechar toda la potencia del framework utilizando APIs y Listeners específicos del mismo. El framework además debería poder asegurar un funcionamiento estable y robusto ante el acceso concurrente a las conversaciones, de modo que el control de concurrencia pueda ser transparente para el programador. Se definieron distintas semánticas de terminación de conversaciones, así como las políticas de expiración en caso de que el usuario permanezca demasiado tiempo sin interactuar con la aplicación. El potencial de las conversaciones realmente se puede apreciar cuando consideramos los modelos extendidos, como los introducidos por las conversaciones anidadas o las continuations. Por último repasamos los problemas de gestión de estado más comunes en las aplicaciones web, desde la perspectiva de brindar una solución utilizando conversaciones.

Lo que se deduce de este Capítulo es que desarrollar una aplicación con un framework MVC que soporte conversaciones hace posible construir una aplicación más sencilla de programar, más legible y por lo tanto más fácil de mantener, más robusta y más predecible.

Capítulo 4

JBoss Seam

En este Capítulo se estudiará en detalle cómo es el soporte de conversaciones proporcionados por el framework JBoss Seam versión 2.2. Para ello nos basaremos en las definiciones introducidas en el Capítulo 3. Sin embargo, nos limitaremos exclusivamente a describir las características relacionadas a las conversaciones que nos provee este framework: una descripción completa de todas las funcionalidades que ofrece está fuera del alcance de este documento, y se refiere al lector interesado a visitar el sitio web del mismo [24].

4.1. Introducción

Seam es un framework para el desarrollo de aplicaciones web con soporte nativo de conversaciones, pensado para ser integrado muy fácilmente con componentes del estándar Java EE 5 [29]. En particular ofrece la posibilidad de poder integrar transparentemente EJBs y JSF, que son dos tecnologías del mismo estándar pero con un modelo de programación diferente. En efecto, los EJBs se utilizan para proveer servicios (Session Beans) o para modelar los objetos del negocio (entity classes) del lado del servidor, mientras que JSF (siendo el framework de desarrollo web estándar) utiliza backing beans para gestionar la lógica de presentación, que es la que llama a la capa de servicios (EJBs) y operan sobre el modelo de objetos.

Sin embargo, para utilizar JSF y EJBs el estándar requiere que se configuren archivos XML para declarar los backing beans, obtener referencias a EJBs a través de JNDI, y a menudo usar los backing beans sólo como un facade de la capa de servicios. Seam simplifica esta integración haciendo transparente el uso de los backing beans de JSF y los EJBs, permitiendo que un EJB sea utilizado como un backing bean.

Seam también es bastante versátil en cuanto a su modelo de programación, ya que no solo permite utilizar EJBs como backing beans de JSF, sino que también permite utilizar cualquier POJO[1] en vez de un EJB, de manera que una aplicación desarrollada con Seam no necesariamente debe correr en un servidor de aplicaciones compatible completamente con Java EE 5, sino que puede correr en un contenedor web como Tomcat [15] o Jetty [16].

En las siguientes Secciones se describirá cómo Seam gestiona las conversaciones, en una estructura similar a la brindada en el Capítulo 3.

4.2. Conversaciones en Seam

En Seam las conversaciones pueden pasar por dos estados:

- Temporaria: Son las conversaciones creadas por defecto. Se crea una por cada request (a menos que una conversación larga sea retomada) y terminan cuando la respuesta está totalmente renderizada, inclusive sobreviven a un redirect, de modo que se pueden renderizar hasta dos páginas con este tipo de conversaciones.
- Larga (de duración prolongada): Las conversaciones temporarias son promovidas a Largas si se le dice a Seam que las convierta en conversaciones de duración prolongada, lo cual significa que la misma será mantenida durante múltiples requests.

[1] Plain Old Java Object.

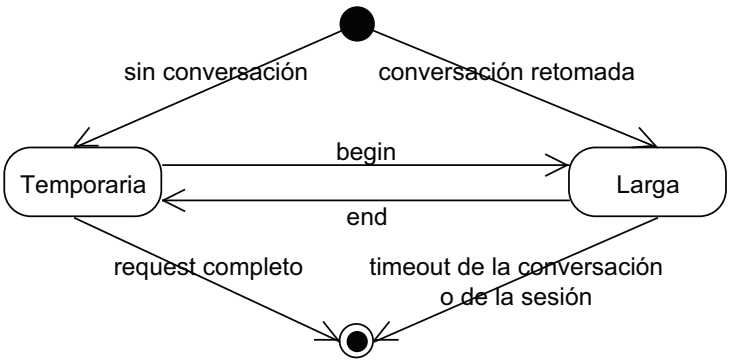

Figura 4.1: Estados por los que pasa una conversación

En la Figura 4.1 se muestra el diagrama de estados por los que pasa una conversación. En cada request, o bien se puede crear una nueva conversación Temporaria (estado Temporaria), o bien el request contiene el identificador de conversación (el parámetro `conversationId`) que indica que se debe retormar una conversación Larga (estado Larga). Anotando un método de la clase Action con la anotación `@Begin`, una conversación Temporaria es promovida a Larga, con lo cual puede durar múltiples requests. Si un método de la clase Action es anotado con `@End` se instruye a Seam que "degrade" la conversación Larga a una Temporaria, de manera que la misma (a partir de ese momento) solo sobreviva ese request (y un redirect opcionalmente) antes de ser destruida. Una conversación Larga también es destruida si el tiempo de inactividad entre un request y otro supera un timeout, o bien la sesión misma del usuario supera su timeout configurado.

En Seam el scope más importante es el de Conversación, tal es así que es el scope por defecto asignado a un SFSB (Stateful Session Bean) cuando no se ha especificado uno explícitamente.

4.3. Identificación de conversaciones

JBoss Seam identifica las conversaciones por medio de un parámetro de petición HTTP llamado `conversationId`[2] y soporta tanto la identificación de conversaciones por clave sintética como por clave natural. En ambos casos, el identificador puede viajar al servidor tanto como un parámetro de petición GET de HTTP, o bien como un campo adicional dentro de una petición POST (comúnmente como un campo oculto de un formulario HTML).

La identificación por clave sintética consta de un número entero que se va incrementando a medida que se van creando nuevas conversaciones. Por supuesto, este número es único, es decir que solo existe una conversación con un identificador determinado dentro de una misma sesión de usuario.

Las conversaciones con claves naturales, o "conversaciones naturales" son definidas dentro de un archivo de configuración llamado `pages.xml`, donde se declara el nombre del parámetro de la petición HTTP y una expresión que indica desde donde se obtendrá el valor de la misma. Por ej. en el Listado 4.1 se declara que la conversación utilizará el parámetro `hotel` en la petición HTTP para referirse al identificador, mientras que el valor del mismo estará almacenado dentro de la variable `nombre` dentro del objeto `hotel` (el cual estará almacenado en el contexto de la conversación). También se ve en el Listado que la conversación tiene un *nombre* el cual será usado dentro del archivo `pages.xml` asociando las vistas que participarán de esa conversación.

Desde una página JSF, se puede especificar el nombre de la conversación para crear un hipervínculo a la acción que participa de esa conversación, como muestra el Listado 4.2. El

[2]El nombre de este parámetro HTTP puede cambiarse por otro nombre desde un archivo de configuración si se desea, cambiando la propiedad `org.jboss.seam.core.manager.conversationIdParameter` en el archivo `components.xml`.

```
<conversation name="agendarHotel"
    parameter-name="hotel"
    parameter-value="#{hotel.nombre}"/>

<page view-id="/agendarHotel.xhtml" conversation="agendarHotel" login-required
    ="true">
    <navigation from-action="#{agendarAction.confirmar}">
        <rule if-outcome="success">
            <redirect view-id="/reserva.xhtml">
                <param name="id" value="#{agendarAction.reserva.reservaId}"/>
            </redirect>
        </rule>
    </navigation>
</page>
```

Listado 4.1: Declarando una conversación natural en Seam

```
  <s:link value="Reservar" action="#{agendarAction.reservarHotel}"
      conversationName="agendarHotel"/>
```

Listado 4.2: Hipervínculo a una acción retomando una conversación natural en Seam

tag `<s:link>` es uno de tantos proporcionados por la biblioteca de *taglibs* de Seam: este tag genera el código HTML necesario para llamar a una acción que participa de una conversación (en el caso de una conversación con identificador sintético no hace falta especificar el atributo `conversationName`).

4.4. Abstracciones para manipular conversaciones

Seam soporta diversas abstracciones para poder manejar el estado de las conversaciones, como accederlas desde el *Expression Language* (EL) dentro de páginas JSP, desde archivos donde se define el flujo de navegación, desde las clases Action mediante *Dependency Injection*, *outjection* y *bijection*, así también como por medio de una API.

4.4.1. Acceso mediante EL

Seam utiliza JBoss EL que es una extensión del estándard *Unified Expression Language* proveyendo varias mejoras que incrementan la expresividad de las expresiones del EL. JBoss EL permite escribir expresiones de métodos con parámetros lo que no existe en el EL estándar, por ej. la expresión `#{reservarAccion.reservarHotel(hotel, usuario)}` pasa como parámetro el `hotel` y `usuario` al método `reservarHotel` sobre el objeto `reservarAction`.

Otra característica de JBoss EL son las proyecciones, que permiten mapear una subexpresión a través de una expresión multivaluada (como una lista, conjunto, etc.) Por ejemplo, la expresión `#{compania.dptos}` podría retornar una lista de departamentos, pero si lo que se necesita es solamente el nombre del departamento, en vez de iterar sobre la lista se puede escribir la expresión como `#{compania.dptos.{d|d.nombre}}`

4.4.2. Archivos de flujo de navegación

Seam soporta la descripción del flujo de navegación entre las distintas Acciones y páginas utilizando dos tipos de archivos de configuración en formato XML. Por un lado, para las aplicaciones que tienen una forma de navegación relativamente libres de restricciones se pueden utilizar las reglas de navegación de JSF y Seam definidas dentro del archivo `pages.xml` (que es un archivo específico de Seam que mejora el control de flujo estándar de JSF), mientras que para aplicaciones más restringidas en su estilo de navegación (por ej. aplicaciones donde el usuario debe seguir las reglas de un *workflow*) se puede definir el flujo usando el lenguaje *jPDL* (que es el lenguaje usado por *jBPM*[22], un motor de gestión de procesos de negocio); esto permite a Seam utilizar un proceso de negocio para definir el flujo de páginas.

Usando reglas de navegación en el archivo `pages.xml`

El archivo `pages.xml` unifica la lógica de navegación con la lógica de "orquestración" de Seam. Esta lógica incluye parámetros de las páginas, gestión de conversaciones, seguridad, manejo de excepciones, y más. En el Listado 4.3 (abajo) se muestra un caso extraído de uno de los ejemplos que vienen en Seam, mientras que en el Cuadro 4.1 se describen las etiquetas disponibles.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <pages xmlns="http://jboss.com/products/seam/pages"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://jboss.com/products/seam/pages http://jboss.com/
        products/seam/pages-2.2.xsd"
5   no-conversation-view-id="/main.xhtml" login-view-id="/home.xhtml">
6
7   <page view-id="/register.xhtml">
8     <action if="#{validation.failed}" execute="#{register.invalid}" />
9     <navigation>
10      <rule if="#{register.registered}">
11        <redirect view-id="/home.xhtml" />
12      </rule>
13    </navigation>
14  </page>
15
16  <page view-id="/home.xhtml" action="#{identity.isLoggedIn}">
17    <navigation from-action="#{identity.isLoggedIn}">
18      <rule if-outcome="true">
19        <redirect view-id="/main.xhtml" />
20      </rule>
21      <rule if-outcome="false">
22        <render view-id="/home.xhtml" />
23      </rule>
24    </navigation>
25  </page>
26
27  <page view-id="/password.xhtml" login-required="true">
28    <navigation>
29      <rule if="#{identity.loggedIn and changePassword.changed}">
30        <redirect view-id="/main.xhtml" />
31      </rule>
32    </navigation>
33  </page>
34
35  <page view-id="/main.xhtml" login-required="true">
36    <navigation from-action="#{hotelBooking.selectHotel(hot)}">
37      <redirect view-id="/hotel.xhtml" />
38    </navigation>
39    <navigation from-action="#{bookingList.cancel}">
40      <redirect />
41    </navigation>
42  </page>
43
44  <page view-id="/hotel.xhtml" conversation-required="true" login-required="
        true">
45    <description>View hotel: #{hotel.name}</description>
46    <navigation from-action="#{hotelBooking.bookHotel}">
47      <redirect view-id="/book.xhtml" />
48    </navigation>
49  </page>
50
51  <page view-id="/book.xhtml" conversation-required="true" login-required="
        true">
52    <description>Book hotel: #{hotel.name}</description>
53    <navigation from-action="#{hotelBooking.setBookingDetails}">
54      <rule if="#{hotelBooking.bookingValid}">
55        <redirect view-id="/confirm.xhtml" />
56      </rule>
57    </navigation>
58  </page>
59
60  <page view-id="/confirm.xhtml" conversation-required="true" login-required="
        true">
61    <description>Confirm booking: #{booking.description}</description>
62    <navigation from-action="#{hotelBooking.confirm}">
```

```
63        <redirect view-id="/main.xhtml" />
64      </navigation>
65    </page>
66
67    <page view-id="*">
68      <navigation from-action="#{identity.logout}">
69        <redirect view-id="/home.xhtml" />
70      </navigation>
71      <navigation from-action="#{hotelBooking.cancel}">
72        <redirect view-id="/main.xhtml" />
73      </navigation>
74    </page>
75
76    <exception class="org.jboss.seam.security.NotLoggedInException">
77      <redirect view-id="/home.xhtml">
78        <message severity="warn">You must be logged in to use this feature</
              message>
79      </redirect>
80    </exception>
81    <exception class="javax.faces.application.ViewExpiredException">
82      <redirect view-id="/home.xhtml">
83        <message severity="warn">Session expired, please log in again</message>
84      </redirect>
85    </exception>
86  </pages>
```

Listado 4.3: Declarando la lógica de navegación usando el archivo `pages.xml`

Usando un archivo jPDL de definición de proceso

El proceso de ejemplo en el Listado 4.4 (que se puede encontrar entre los ejemplos ofrecidos por Seam) modela el flujo de una aplicación donde se intenta que el usuario adivine qué número entre 1 y 100 ha seleccionado el servidor. La semántica del XML que define el proceso es bastante directa: El atributo `name` en el elemento `<pageflow-definition>` define el nombre del objeto Action que estará almacenado en la conversación. El elemento `<start-page>` define el comienzo del proceso de negocio (la conversación) donde el atributo `view-id` especifica cual es la página a mostrar, la cual tiene un campo de texto para ingresar el número y 3 opciones posibles (botones para oprimir): *guess* (adivinar), *giveup* (rendirse) y *cheat* (hacer trampa) las cuales son representadas por el elemento `transition` y su atributo `to` que define a cual estado pasará el proceso en caso de que ocurra ese evento. La transición puede tener elementos `<action>` que evalúan expresiones en el objeto Action cuando se produce la transición (en este caso chequea si el usuario ha acertado el número o no). El elemento `<decision>` representa un punto dentro del proceso donde se puede chequear el estado del mismo y tomar algún camino alternativo; cuando se presiona el botón [guess] se evalúa si el usuario ha acertado el número evaluando el método `isCorrectGuess()` y, dependiendo del resultado, se tomará una transición u otra. Una construcción muy potente es la composición de procesos, lograda por el tag `<process-state>`, el cual pausa la ejecución de un flujo para comenzar la ejecución de otro (tag `<sub-process>`): cuando el flujo interno termina, se retoma el control al flujo exterior con la transición definida en el tag `<process-state>`. El tag `<end-conversation>` termina la conversación, mientras que en los tags `<start-page>` y `<page>` se puede definir un atributo `no-conversation-view-id` que define a cual vista ir en caso de que no se encuentre la conversación en la cual debe participar el proceso.

Manejo del botón de Retroceso

Como se puede observar, los dos modelos de navegación tienen características que lo hacen adecuado para una u otra situación. En principio, dos cosas pueden notarse:

- Las reglas de navegación de Seam (usando `pages.xml`) son mucho más simples (aunque oscurece el hecho de que el código en Java que lo acompaña debe ser más complejo).
- El jPDL hace que la interacción del usuario sea fácil de visualizar rápidamente, sin necesidad de observar las páginas JSP o código en Java.

Elemento	Descripción
<pages>	Elemento raíz que define la configuración general para todas las páginas. El atributo no-conversation-view-id especifica la página a mostrar cuando no se encuentra una conversación vigente. Análogamente, login-view-id define la página a mostrar cuando no hay un usuario logueado en el sistema.
<page>	Define la navegación así como la lógica de "orquestración" asociada con la view-id. Algunos atributos posibles son: view-id: La página a mostrar. action: Una expresión a ser ejecutada antes de renderizar la página. no-conversation-view-id: Página a mostrar si no hay una conversación vigente. conversation-required: Indica si debe haber una conversación de larga duración vigente o no. login-required: Si el usuario debe estar logueado o no. concurrent-request-timeout: Tiempo en ms. que se espera antes de descartar la petición HTTP si no puede ser procesada. conversation: Nombre de la conversación natural a utilizar. timeout: Valor de timeout en milisegundos para que expire la conversación.
<conversation>	Configura una conversación natural por su nombre. Los atributos posibles son: name: El nombre de la conversación natural. parameter-name: El nombre del parámetro de la petición HTTP que contiene el identificador de la conversación. paramater-value: Expresión EL que evalúa la clave natural del negocio para utilizar como identificador de conversación.
<description>	Provee una descripción para una página que será mostrada en el ConversationSwitcher.
<action>	Ejecuta una acción en respuesta a una petición HTTP. Los atributos posibles son: execute: Expresión EL que indica la acción a ejecutar. if: Expresión booleana EL que indica si debe ejecutarse la acción o no. on-postback: true si se admite ejecutar la acción en un *post-back* de JSF; false si no se permite (*i.e.* solo en peticiones GET).
<navigation>	Define las reglas de navegación asociadas con una página.
<rule>	Define un resultado de acción específico o una expresión booleana bajo la cual debería ocurrir la navegación. Atributos: if-outcome: Define lo que retorna la acción ejecutada. if: Expresión booleana EL que indica si debe seguirse la regla o no.
<redirect>	Hace un redirect HTTP hacia la view-id especificada. Los atributos disponibles son: view-id: La página hacia la cual hacer el redirect. url: Una URL donde hacer el redirect. include-page-params: True si se desea incluir los parámetros de la petición HTTP.
<render>	Renderiza la vista especificada por el atributo view-id.
<begin-conversation>	Comienza una conversación de larga duración. Atributos: join: True si se desea unirse a la conversación si es que ya está activa. False si se quiere crear una nueva. nested: True si deseamos crear una conversación anidada cuando ya hay una activa. False si no se desea que sea anidada. flush-mode: Especifica el modo de operación del *Persistence Context* para la transferencia de datos a una base de datos. Los valores posibles son: ▪ MANUAL: Hay que llamar al método flush() explícitamente. ▪ AUTO: La transferencia ocurre automáticamente cuando se hace el commit o antes de una consulta a la base de datos. ▪ COMMIT: La transferencia ocurre automáticamente cuando se hace el commit a la base de datos. if: Comienza la conversación si se cumple la condición booleana especificada.
<end-conversation>	Finaliza una conversación de larga duración cuando se accede a una página, en la navegación o ante la ocurrencia de una excepción. Atributos: before-redirect: Indica si se debe terminar antes (true) o después de hacer un redirect a la vista. root: En el contexto de ejecución de una conversación anidada, estableciendo el atributo a true indica que se desea que se destruya la pila de conversaciones completa. Si la conversación no está anidada simplemente se destruye la conversación actual. if: Termina la conversación si se cumple la condición booleana especificada.

Cuadro 4.1: Algunos elementos utilizados dentro del archivo pages.xml.

```
<pageflow-definition
  xmlns="http://jboss.com/products/seam/pageflow"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
    "http://jboss.com/products/seam/pageflow
     http://jboss.com/products/seam/pageflow-2.2.xsd"
  name="numberGuess">

   <start-page name="displayGuess" view-id="/numberGuess.jspx">
      <redirect/>
      <transition name="guess" to="evaluateGuess">
         <action expression="#{numberGuess.guess}"/>
      </transition>
      <transition name="giveup" to="giveup"/>
      <transition name="cheat" to="cheat"/>
   </start-page>

   <decision name="evaluateGuess" expression="#{numberGuess.correctGuess}">
      <transition name="true" to="win"/>
      <transition name="false" to="evaluateRemainingGuesses"/>
   </decision>

   <decision name="evaluateRemainingGuesses" expression="#{numberGuess.
       lastGuess}">
      <transition name="true" to="lose"/>
      <transition name="false" to="displayGuess"/>
   </decision>

   <page name="giveup" view-id="/giveup.jspx">
      <redirect/>
      <transition name="yes" to="lose"/>
      <transition name="no" to="displayGuess"/>
   </page>

   <process-state name="cheat">
      <sub-process name="cheat"/>
      <transition to="displayGuess"/>
   </process-state>

   <page name="win" view-id="/win.jspx">
      <end-conversation/>
      <redirect/>
   </page>

   <page name="lose" view-id="/lose.jspx">
      <end-conversation/>
      <redirect/>
   </page>

</pageflow-definition>
```

Listado 4.4: Declarando un proceso jPDL que modela el flujo de páginas

Además, el modelo basado en jPDL es más *restringido*: por cada estado lógico dentro del flujo, hay un conjunto restringido de transiciones posibles hacia otros estados.

Cuando se usan las reglas de navegación en el archivo `pages.xml`, Seam le permite al usuario navegar libremente con los botones de Retroceso, Avance y Refresco. Es responsabilidad de la aplicación el asegurar que el estado conversacional se mantenga internamente consistente cuando esto ocurre. Se sabe que cuando se usa un framework sin soporte de conversaciones (como Struts) esto es casi imposible de lograr! En el caso de Seam, se pueden lograr muy buenos resultados utilizando el atributo `no-conversation-view-id`, el cual instruye a Seam que redirija el flujo de la aplicación si la conversación ya no existe.

Por otro lado, usando jPDL el botón de Retroceso es interpretado como una transición indefinida hacia un estado previo. Ya que este modelo "stateful" fuerza un conjunto definido de transiciones desde el estado corriente, el botón de Retroceso no es permitido por defecto! Seam detecta transparentemente el uso del botón de Retroceso, y bloquea cualquier intento de ejecutar una acción desde una página previa (posiblemente desactualizada) y simplemente redirije al usuario a la página "corriente" (y muestra un mensaje). Puede anularse este comportamiento incorporando el atributo `back="enabled"` a los tags `<page>`; esto permite la navegación vía el botón de Retroceso hacia un estado previo. También puede usarse el atributo `no-conversation-view-id` para redirijir al usuario a otra página en caso de que la conversación ya no exista.

4.4.3. Dependency bijection

Como se describió en la Sección 3.4.3, Seam soporta el uso de *dependency injection*, *outjection*, y en general de *bijections*. Esta funcionalidad aporta a la legibilidad del código, y se materializa a través de dos anotaciones: `@In` para hacer *injection* y `@Out` para hacer *outjection*. Las mismas pueden aparecer asociadas a un atributo de la clase action o bien a uno de sus métodos, y si en el mismo lugar se especifica ambas, estamos hablando de una *bijection*.

En Seam, los componentes son *contextuales*, lo cual significa que cada vez que un método en un componente es llamado, se hace la injection, y cada vez que termina de ejecutar el método se hace una outjection; esto es debido a que en Seam los componentes pueden ser *stateful* ya que se pueden almacenar en (por ej.) la sesión de usuario, y cada vez que necesita ejecutarse algún método sobre el componente, las variables de instancia deben ser establecidas nuevamente. Compare este mecanismo con frameworks tradicionales de Dependency Injection, como Spring, cuyos componentes son de una naturaleza *stateless* y alcanza con establecer sus dependencia una sola vez cuando se instancia el componente en cuestión.

Anotación `@Name`

En el Listado 4.5 vemos un ejemplo de esta anotación, y si bien no está directamente relacionado con el concepto de bijection, es una de las anotaciones básicas de Seam. La anotación `@Name` indica que el componente tendrá el nombre específico indicado como parámetro; este componente luego puede referenciarse desde otros componentes que utilicen el objeto utilizando ese nombre, o desde una página JSP como una expresión del EL, por ej. para obtener el usuario: `#{loginAction.user}`.

Anotación `@In`

Esta anotación especifica que un valor debe ser inyectado, ya sea sobre una variable de instancia o sobre un método *setter*, como se muestra en el Listado 4.5. En este ejemplo vemos también una anotación extra: `@Stateless` indica que la clase es un *Stateless Session Bean* (SLSB) y es parte del estándar EJB, y muestra la facilidad de integrar ambas tecnologías.

Por defecto, Seam realizará una búsqueda por prioridad en todos los contextos, usando el nombre de la propiedad o variable de instancia siendo inyectada, pero también se puede especificar el nombre de la variable de contexto explícitamente, por ejemplo: `@In("usuarioLogueado")`.

Si se desea que Seam cree una instancia del componente cuando no haya ningún componente instanciado ligado al nombre especificado, se debe especificar `@In(create=true)`. Si el componente es opcional (*i.e.* puede ser `null`), se puede escribir `@In(create=false)`.

```
1 @Name("loginAction")
2 @Stateless
3 public class LoginAction implements Login {
4   @In User user;
5    ...
6 }
7
8 @Name("loginAction")
9 @Stateless
10 public class LoginAction implements Login {
11   User user;
12
13   @In
14   public void setUser(User user) {
15     this.user = user;
16   }
17 }
```

Listado 4.5: Inyección de valores en variable de instancia o método setter

```
1 @Name("loginAction")
2 @Stateless
3 public class LoginAction implements Login {
4   @Out User user;
5    ...
6 }
7
8 @Name("loginAction")
9 @Stateless
10 public class LoginAction implements Login {
11   User user;
12
13   @Out
14   public User getUser() {
15     return user;
16   }
17 }
```

Listado 4.6: Outjection de valores desde variable de instancia o método getter

Por defecto, Seam supone que el componente inyectado debe estar previamente instanciado. Si se desea que se pueda inyectar el valor `null` cuando el componente no está instanciado, se puede especificar `@In(required=false)`.

Se puede utilizar el parámetro `scope` de la anotación `@In` para especificar un único scope donde buscar el componente a inyectar, en vez de que Seam busque entre todos los contextos. Por ej. `@In(scope=ScopeType.SESSION)`.

Por último, todos los componentes inyectados son "desinyectados" (*i.e.* seteados a `null`) inmediatamente después de que el método termine su ejecución y se haga la outjection.

Anotación `@Out`

La anotación`@Out` indica que debe hacerse la outjection, *i.e.* el atributo debe ser salvado a su correspondiente contexto, ya sea desde una variable de instancia o desde un método getter. como muestra el Listado 4.6.

El parámetro `value`, por ej. en `@Out("usuario")` define el nombre de la variable de contexto que tendrá el objeto, y si se omite, el valor tomado es el de la variable de instancia anotada o bien del método getter anotado.

Si se desea controlar que el valor a salvar no sea `null`, se puede especificar `@Out(required = true)`, que es el valor por defecto. Si se desea que se salve un valor que posiblemente sea `null`, se debe especificar el parámetro en `false`.

Se puede utilizar el parámetro `scope` de la anotación `@Out` para especificar el scope donde almacenar el componente. Si no se especifica ningún scope explicitamente, el scope por defecto depende de si el valor es una instancia de un componente de Seam. Si lo es, se usa el scope del componente. De lo contrario se usa el scope del componente que contiene el atributo con la

anotación `@Out` (pero si el scope del componente es `STATELESS`, el scope `EVENT` es el usado).

Bijección

La biyección se produce cuando una misma variable es anotada con `@In` y `@Out`, ya sea colocando ambas anotaciones sobre la variable de instancia, o bien colocandolas sobre los correspondientes métodos setter y getter, respectivamente.

4.4.4. API

Seam provee la clase `org.jboss.seam.core.Conversation` para la gestión de conversaciones desde una clase en Java (frecuentemente una clase Action representando lógica de presentación de la aplicación). Utilizando el método estático `Conversation.instance()` se obtiene acceso a una instancia, que contendrá los datos de la conversación corriente (si es que el request está participando de alguna). La siguiente es una lista de la funcionalidad provista por la clase `Conversation`:

- `begin()`: Comienza una conversación de larga duración, si no hay ninguna activa.
- `beginNested()`: Comienza una nueva conversación anidada.
- `begin(boolean join, boolean nested)`: Empieza una nueva conversación. Si `join` es `true` y ya hay una conversación activa, se une a ella. Si nested es true y hay una conversación activa, se crea una nueva conversación anidada.
- `changeFlushMode(FlushModeType flushMode)`: Cambia el modo de hacer flush de todos los contextos de persistencia manejados por Seam para la conversación.
- `end()`: Termina una conversación de larga duración.
- `end(boolean beforeRedirect)`: Termina una conversación de larga duración. Establecer `beforeRedirect` en `true` si se desea que la conversación sea destruida antes de hacer un redirect.
- `endAndRedirect()`: Termina una conversación anidada y hace un redirect hacia el último `view-id` definido para la conversación padre.
- `endAndRedirect(boolean endBeforeRedirect)`: Idem anterior, pero destruye la conversación antes de hacer un redirect si `endBeforeRedirect=true`.
- `endBeforeRedirect()`: Termina una conversación de larga duración y la destruye antes de hacer un redirect.
- `Integer getConcurrentRequestTimeout()`: Retorna el timeout configurado para peticiones concurrentes (para aplicaciones AJAX).
- `setConcurrentRequestTimeout(Integer concurrentRequestTimeout)`: Este método establece el timeout configurado para peticiones concurrentes (para aplicaciones AJAX).
- `String getDescription()`: Retorna el texto descriptivo de la conversación.
- `setDescription(String descriptio)`: Establece el texto descriptivo de la conversación para su uso en la lista de conversación, *breadcrumbs* o *conversation switcher*.
- `String getId()`: Retorna el identificador de la conversación.
- `String getParentId()`: Retorna el identificador de la conversación padre de una conversación anidada.
- `String getRootId()`: Retorna el identificador de la conversación raíz de una conversación anidada.
- `Integer getTimeout()`: Retorna el timeout de esta conversación.
- `setTimeout(Integer timeout)`: Establece el timeout de esta conversación.

- `String getViewId()`: Retorna el `view-id` a ser usado cuando se cambia nuevamente a esta conversación desde la lista de conversaciones, *breadcrumbs* o el *conversation switcher*. Si no se ha especificado ninguno, retorna el `view-id` corriente.
- `setViewId(String outcome)`: Establece el `view-id` a ser usado cuando se cambia nuevamente a esta conversación desde la lista de conversaciones, *breadcrumbs* o el *conversation switcher*.
- `isLongRunning()`: Retorna `true` si la conversación es de larga duración. Note que este método retorna `false` aún cuando la conversación ha sido promovida temporalmente a larga duración por el curso de un redirect, de manera que hace lo que el usuario realmente espera.
- `isNested()`: Retorna `true` si la conversación es anidada.
- `killAllOthers()`: Destruye todas las conversaciones excepto esta.
- `leave()`: Dejar el scope de la conversación corriente.
- pop(): Hace un "pop" de la pila de conversación, cambiando a la conversación padre.
- `redirect()`: Cambia nuevamente al último `view-id` definido para la conversación corriente.
- `redirectToParent()`: Hace un "pop" de la pila de conversación, y hace un redirect hacia el último `view-id` definido para la conversación padre.
- `redirectToRoot()`: Cambia hacie la conversación raíz y hace un redirect hacia el último `view-id` definido para la conversación raíz.
- `root()`: Cambia hacia la conversación raíz.

4.4.5. Eventos

Los componentes en Seam pueden interactuar implementando el patrón de diseño *Observer* [18] mediante `eventos manejados por componentes`: Existen componentes que pueden disparar eventos y componentes que pueden escuchar un evento en particular. Seam permite definirlos de dos formas diferentes:

- Definiendo el evento y sus *listeners* en el archivo `components.xml`: Por ejemplo en el Listado 4.7 se define el evento `hola` y los componentes y métodos que son llamados cuando éste evento es disparado (`holaListener.dijoHola()` y `logger.logHola()`):

```
<components>
  <event type="hola">
    <action execute="#{holaListener.dijoHola}"/>
    <action execute="#{logger.logHola}"/>
  </event>
</components>
```

Listado 4.7: Declarando eventos en el archivo `components.xml`

- Definiendo el evento y sus *listeners* con la anotación `@Observer`: Usando mismo ejemplo anterior, los *listeners* se definen de la siguiente manera:

```
@Name("holaListener")
public class HolaListener {
  @Observer("hola")
  public void dijoHola() {
    FacesMessages.instance().add("Estoy diciendo Hola!");
  }
}
```

Listado 4.8: Declarando los listeners con la anotación `@Observer`

El disparo de un evento puede realizarse de dos formas distintas, ya sea usando la anotación `@RaiseEvent` o programáticamente mediante una instancia de la clase `Events`, como se muestra en el Listado 4.9:

```
// usando anotacion:
@RaiseEvent("hola")
public void disparaHola() {
  FacesMessages.instance().add("Hola Mundo!");
}

// usando la API de Seam:
public void disparaHola2() {
  FacesMessages.instance().add("Hola Mundo!");
  Events.instance().raiseEvent("hola");
}
```

Listado 4.9: Disparando eventos

Además, Seam provee un número de eventos predefinidos que una aplicación puede usar para poder integrarse con el framework en distintos puntos dentro de la ejecución del mismo. Los eventos predefinidos llevan como prefijo un nombre de paquete Java para evitar confusiones con sus nombres. Los eventos relacionados con el manejo de conversaciones son los siguientes (para una lista completa de eventos, ver la documentación de Seam):

- `org.jboss.seam.noConversation`: Llamado cuando no existe ninguna conversación de larga duración y se requiere que haya una.
- `org.jboss.seam.preDestroyContext.CONVERSATION`: Llamado justo antes de que el contexto CONVERSATION sea destruido.
- `org.jboss.seam.postDestroyContext.CONVERSATION`: Es llamado después de que el contexto CONVERSATION fue destruido.
- `org.jboss.seam.beginConversation`: Llamado cuando comienza una conversación de larga duración.
- `org.jboss.seam.endConversation`: Llamado cuando termina una conversación de larga duración.
- `org.jboss.seam.conversationTimeout`: Llamado cuando ocurre un timeout de la conversación. El identificador de la conversación es pasado como un parámetro.

4.5. Terminación de conversaciones

Como se vio en la Sección 4.2, en Seam hay dos tipos de conversaciones: las temporarias y las de larga duración. Toda petición HTTP participa de una conversación, ya sea creando una nueva conversación temporaria, o bien uniéndose a una conversación de larga duración ya creada (ver Figura 4.1).

4.5.1. Terminación normal

Las conversaciones temporarias terminan automáticamente cuando se completa la petición HTTP que la creó, sobreviviendo únicamente un redirect HTTP.

Cuando termina una conversación de larga duración (mediante el procesamiento de un método anotado con `@End` o definiendo el elemento `<end-conversation>` en un archivo de flujo de navegación), ésta no es destruida sino que se degrada a una conversación temporaria, la cual puede sobrevivir un redirect HTTP. Opcionalmente (si se lo declara explícitamente) se puede especificar que la conversación de larga duración sea destruida al finalizar la petición HTTP actual, y que el redirect sea procesado en una nueva conversación temporaria.

4.5.2. Comportamiento ante la presencia de excepciones

Seam permite definir la semántica de las excepciones de dos formas: anotando la clase Exception o bien declarandola en el archivo `pages.xml`. Estos mecanismos están diseñados para ser combinados con la anotación estándar `@ApplicationException` de EJB 3.0, la cual especifica si la excepción debe causar un rollback en la transacción corriente.

EJB especifica reglas bien definidas que nos permite controlar si la transacción es marcada para hacer un rollback cuando una excepción es elevada desde un método de un EJB: Las *excepciones de sistema* siempre causan un rollback de la transacción, mientras que las *excepciones de aplicación* no causan rollback por defecto, pero sí si dicha excepción está anotada con `@ApplicationException(rollback=true)`. Seam aplica las mismas reglas de rollback que EJB 3.0 para los componentes de Seam.

Pero estas reglas aplican solo en la capa de componentes de Seam. Si la excepción no es atrapada y se propaga fuera de la capa de componentes de Seam, o bien fuera de la capa de JSF, entonces Seam hace un rollback de cualquier transacción que esté activa cuando ocurre la excepción y no sea atrapada en la capa de componentes de Seam.

Con respecto al manejo de las conversaciones, Seam permite que el programador declare si la excepción debería terminar o no una conversación de larga duración.

4.5.3. Acceso a conversaciones ya terminadas

Ciertas páginas son relevantes solamente en el contexto de ejecución de una conversación de larga duración. Seam "proteje" dichas páginas chequeando que exista una conversación de larga duración como pre-requisito para poder ser renderizadas.

En el archivo `pages.xml` se puede indicar que la conversación corriente se encuentre ya iniciada (de larga duración, anidada o no) para que una página pueda renderizarse, utilizando el atributo `conversation-required` de esta manera:

```
<page view-id="/hotel.xhtml" conversation-required="true" />
```

Como nota al margen, el único defecto de este mecanismo es que no existe una forma predefinida de indicar *cual* conversación de larga duración es la requerida. El programador puede construir sobre esta autorización básica chequeando si un valor específico está presente en la conversación dentro del Action de la página.

Cuando Seam determina que la página es requerida fuera de una conversación de larga duración, las siguientes acciones son llevadas a cabo:

- Se eleva un evento contextual llamado `org.jboss.seam.noConversation`.
- Se registra un mensaje de estado usando la clave `org.jboss.seam.NoConversation` de un `ResourceBundle`.
- El usuario es redirigido hacia una página alternativa, si ésta fue definida.

La página alternativa es definida en el atributo `no-conversation-view-id` sobre el elemento <pages> en el archivo `pages.xml`:

```
<pages no-conversation-view-id="/index.xhtml" />
```

Por el momento, solo se puede definir una sola página para la aplicación entera.

4.6. Políticas de expiración de conversaciones

De acuerdo a la clasificación realizada en la Sección 3.6, se puede decir que Seam soporta el modelo de expiración de conversaciones en primer y segundo plano (Sección 3.6.2) y expiración selectiva por vista (Sección 3.6.3).

4.6.1. Expiración en primer y segundo plano

Una conversación de larga duración es marcada para expirar cuando se torna la conversación *background* (en segundo plano). En la Figura 4.2 se muestra los dos estados en los que puede estar la conversación de larga duración. Cada vez que una conversación es retomada, es cambiada hacia el estado *foreground* (en primer plano). Recuerde que una conversación es retomada cuando una petición del usuario envía un identificador de conversación existente. Una conversación en primer plano solo expira cuando la sesión de usuario expira. Cualquier otra conversación en la sesión de usuario está en segundo plano cuando es retomada. Una conversación en segundo plano es susceptible de ser expirada tanto por el valor de timeout de conversación como el de la sesión de usuario.

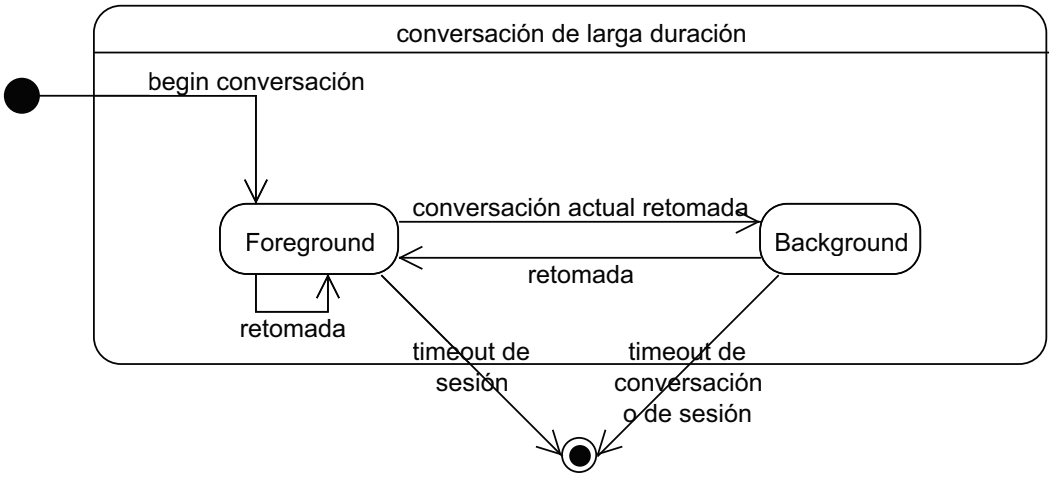

Figura 4.2: Los dos subestados de las conversaciones de larga duración: en *foreground* y *background*.

Seam define un valor global para el timeout de conversaciones (`conversation-timeout`), configurable dentro del archivo `components.xml`, en unidades de milisegundos:

```
<components>
  <!-- defino el timeout de conversacion en 2 minutos -->
  <core:manager conversation-timeout="120000"/>
</components>
```

En cada petición HTTP se determina la conversación en primer plano (la cual podría ser una conversación temporaria), y todas las conversaciones en segundo plano se chequean por el timeout de acuerdo a la última vez que han sido retomadas. Si este tiempo es mayor al valor establecido de `conversation-timeout`, la conversación es destruida y se dispara el evento `org.jboss.seam.conversationTimeout`.

Del párrafo anterior se puede determinar que el valor de timeout configurado no es exacto, sino que depende de la frecuencia con que la aplicación reciba peticiones HTTP. Es decir, puede suceder que la conversación haya expirado por estar inactiva un tiempo superior al timeout configurado, pero será *destruida* efectivamente cuando la aplicación reciba alguna petición HTTP.

4.6.2. Expiración selectiva por vista

Seam permite este tipo de política de expiración configurando el atributo `timeout` (en milisegundos) particular de cada página dentro del archivo `pages.xml` (ver Tabla 4.1):

```
<pages>
  ...
  <page view-id="/verHotel.xhtml" timeout="300000">
    ...
  </page>
  <page view-id="/reservar.xhtml" timeout="600000">
    ...
  </page>
  <page view-id="/confirmar.xhtml" timeout="600000">
    ...
  </page>
</pages>
```

En este ejemplo el timeout de conversación cuando se muestra la página `verHotel.xhtml` es de 5 minutos, mientras que para las otras dos páginas es de 10 minutos. Si el timeout no es especificado para una página en particular, se toma el valor global `conversation-timeout`.

4.7. Control de concurrencia

Seam tiene una capa de gestión de concurrencia en su modelo de contexto. Los contextos (o *scopes*) correspondientes a la Aplicación y Sesión de usuario son *multi-threaded* es decir, permiten el acceso desde múltiples threads de ejecución.[3] Seam permitirá que peticiones concurrentes a un contexto sean procesadas concurrentemente. Los contextos Event y Page son por naturaleza *single-threaded*, o sea que son accedidos por un único thread (el que está procesando la petición corriente). El contexto de Business Process es inherentemente multi-threaded, pero en la práctica es muy raro que haya concurrencia entre los distintos estados por los que pasa un proceso de negocios. Finalmente, Seam fuerza un modelo de *único thread por conversación por proceso* para el contexto Conversación, serializando las peticiones concurrentes para la misma conversación de larga duración.

Debido a que el contexto Sesión de usuario es multi-threaded, y a menudo contiene estado volátil, los componentes almacenados en ese scope siempre están protegidos por Seam de accesos concurrentes siempre y cuando los interceptores de Seam no estén deshabilitados para ese componente. Si los interceptores están deshabilitados, entonces el componente mismo es quien debe garantizar su consistencia. Por defecto Seam serializa las peticiones a los objetos almacenados en Sesión (y detecta y repara cualquier tipo de deadlock que pudiese ocurrir). Sin embargo este no es el comportamiento por defecto para los objetos almacenados en el contexto de Aplicación ya que éstos típicamente no contienen datos volátiles y porque la sincronización a este nivel es *extremadamente* costosa. Aún así, se puede serializar el acceso a un objeto en el contexto de Aplicación agregandole la anotación `@Synchronized`.

Este modelo de concurrencia permite que clientes AJAX pueden usar con seguridad estado volátil de sesión y conversacional, sin necesidad del agregado de código especial por parte del desarrollador.

4.7.1. Llamadas a componentes conversacionales

Los componentes en el contexto de una Conversación no permiten un acceso concurrente real, ya que Seam encola cada petición para procesarlas en forma serial; esto le permite a cada petición ser ejecutada de forma determinística. Sin embargo, una simple cola no es suficiente: en primer lugar, si a un método le toma un tiempo prolongado para completar, ejecutarlo una y otra vez cada vez que el cliente genera una petición no es bueno (es un potencial ataque de Negación de Servicio[4]) y en segundo lugar, cuando se usa AJAX es típicamente con el propósito de proveer una rápida actualización del estado al usuario, de manera que continuar ejecutando la acción después de un largo período de tiempo no es útil.

Por lo tanto, cuando se está trabajando dentro de una conversación de larga duración, Seam encola el evento por un período de tiempo especificado por el atributo `concurrent-request-timeout`: Si no se puede procesar el evento dentro de este tiempo, Seam eleva una `org.jboss.seam.ConcurrentRequestTimeoutException`, la cual puede configurarse en el archivo `pages.xml` para que retorne un error HTTP 503[5] o bien se crea una conversación temporaria e imprime un mensaje al usuario para hacerle saber lo que ha ocurrido. Entonces es muy importante no inundar al servidor con eventos AJAX!

Se puede especificar un valor global por defecto para este timeout (en milisegundos) dentro del archivo `components.xml`, agregando la siguiente línea:

```
<core:manager conversation-timeout="500"/>
```

También se puede configurar el `concurrent-request-timeout` en forma específica para una página en particular, por ej.:

```
<page view-id="/reservar.xhtml"
  conversation-required="true"
```

[3] Normalmente, cada petición HTTP hace que el servidor de aplicaciones obtenga desde un pool de threads uno donde poder realizar el procesamiento. Una vez finalizado el procesamiento y enviada la página renderizada al navegador cliente, el thread vuelve al pool para poder ser reutilizado en una nueva petición.

[4] Los ataques DoS (*Denial of Service*) son los que intentan anular un servidor o deteriorar la calidad del servicio, comúnmente saturando al servidor con peticiones de comunicación externas, de manera que no pueda responder a las peticiones legítimas, o bien lo haga tan lentamente que torne a dicho servicio inutilizable.

[5] RFC 2616 [14]: El error 503 es *Service unavailable*, el cual indica que el servidor no se encuentra disponible para manejar la petición debido a una sobrecarga temporaria o mantenimiento del servidor. La implicación es que ésta es una condición temporaria que será normalizada luego de algún momento.

```
  login-required="true"
  concurrent-request-timeout="2000" />
```

4.8. Integración con Mapeos Objeto-Relacional

Seam provee soporte para los dos frameworks de persistencia más populares para Java: Hibernate y JPA; y la solución está compuesta de dos partes:

- El uso de un contexto de persistencia extendido, almacenado en una conversación, como se describe en la Sección 3.8.5.
- El uso de dos transacciones por petición HTTP: La primera abarca el comienzo de la fase *Restore View* de JSF hasta el final de la fase de *Invoke Application*; la segunda abarca la fase de *Render Response*.

Seam provee su propia gestión de transacciones, el cual tiene activado por defecto para todas las peticiones JSF (se puede desactivar mediante unas líneas de código en el archivo `components.xml`, sin embargo es beneficioso dejarlo activado, aunque se puede usar el contexto de persistencia extendido sin la gestión de transacciones de Seam.

4.8.1. Seam transaction manager

Seam provee una abstracción de gestión de transacciones para comenzar, hacer *commit*, hacer *rollback* y sincronizar con una transacción. Por defecto, Seam usa un componente de transacciones JTA (*Java Transaction API* [31]) que se integra con transacciones tanto CMT[6] como programáticas. Si se está trabajando dentro de un ambiente Java EE 5, se debería instalar el componente de sincronización EJB en `components.xml` con `<transaction:ejb-transaction />`.

Sin embargo, si se está en un servidor que no cumpla con el estándar Java EE 5 (por ej. un contenedor de aplicaciones Web como Tomcat [15]), Seam intentará autodetectar el mecanismo de sincronización de transacción correcto, pero si no puede detectarse debe configurarse en `components.xml` con alguno de los siguientes:

- Transacciones JPA de tipo `RESOURCE_LOCAL` con la interface `javax.persistence.EntityTransaction`.
- Transacciones gestionadas por Hibernate con la interface `org.hibernate.Transaction`.
- Transacciones gestionadas por el framework Spring con la interface `org.springframework.transaction.PlatformTransactionManager`.
- Explícitamente inhabilitar las transacciones gestionadas por Seam.

4.8.2. Contextos de persistencia gestionados por Seam

Si se va a utilizar Seam fuera de un ambiente Java EE 5, no se podrá confiar en el contenedor web para gestionar el ciclo de vida del contexto de persistencia. Aún en ambientes Java EE 5, un programador podría tener una aplicación compleja con muchos componentes con bajo acoplamiento que colaboran juntos en el contexto de una sola conversación, y en este caso se podría dar el caso de que la propagación del contexto de persistencia entre los componentes es dificultoso y propenso a errores.

En ambos casos, se necesitará un contexto de persistencia gestionado por Seam (SMPC, del inglés *Seam-managed Persistence Context*), el cual es un componente de provisto por Seam que gestiona una instancia de un `EntityManager` (para JPA) o `Session` (para Hibernate) en el contexto de una conversación. Además, puede inyectarse con la anotación `@In`.

Los contextos de persistencia gestionados por Seam son extremadamente eficientes en ambientes de *clusters* de servidores. Seam es capaz de ejecutar una optimización que la especificación EJB 3.0 no permite usar a los contenedores para contextos de persistencia gestionados

[6] CMT, o *Container Managed Transactions*, que es la forma declarativa de gestionar las transacciones que poseen los servidores de aplicaciones Java EE.

por el contenedor. Seam soporta *failover* transparente de contextos de persistencia extendidos, sin la necesidad de replicar el estado del contexto de persistencia entre los nodos del *cluster*.

A continuación se verá un ejemplo de cómo configurar un contexto de persistencia gestionado por Seam usando JPA como tecnología de persistencia (para Hibernate es una configuración similar, y se invita al lector a leer la documentación del producto).

Ejemplo 1 *Cómo usar SMPC con JPA*

En el archivo `components.xml` *escribimos:*

```
<persistence:managed-persistence-context name="reservasDatabase"
  auto-create="true"
  persistence-unit-jndi-name="java:/EntityManagerFactories/reservasData"/>
```

donde `java:/EntityManagerFactories/reservasData` *es el nombre JNDI de una unidad de persistencia (*`EntityManagerFactory`*) que debe estar definida en el servidor de aplicaciones. Luego, dentro de las clases Action de Seam podemos inyectar el* `EntityManager` *para su uso:*

```
@Named("ReservarAction")
public class ReservarAction {
  @In EntityManager em;
  ...
}
```

4.8.3. Conversaciones atómicas con SMPC

Los contextos de persistencia almacenados en la conversación permiten programar transacciones optimistas que abarcan múltiples peticiones al servidor sin la necesitad de operaciones de `merge()`, ni re-cargar datos al comienzo de cada petición, y sin la necesidad de luchar con las `LazyInitializationException`, ya que utilizan un contexto de persistencia extendido.

Usando *optimistic locking* se logra el aislamiento y la consistencia de las transacciones. Afortunadamente, tanto Hibernate como JPA hacen muy fácil usar *optimistic locking* por medio de la anotación `@Version`.

Para lograr la atomicidad, hay mantener en memoria los cambios en los objetos persistentes hasta el momento en que termine la conversación, momento en el cual se debe forzar el `flush()` en forma explícita para sincronizar los cambios hacia la base de datos. En Seam se puede lograr este comportamiento cuando se usa con Hibernate, o bien con JPA con Hibernate como proveedor de persistencia, de la siguiente manera:

Ejemplo 2 *Cómo hacer conversaciones atómicas con SMPC*

```
@Named("CarritoAction")
public class CarritoAction {
  @In EntityManager em; // SMPC
  @In @Out Carrito carrito;

  @Begin(flushMode=MANUAL)
  public void beginCarrito() {
    carrito = new Carrito();
    em.persist(carrito); // no sincroniza con la base de datos porque
        flushMode==MANUAL
  }

  public void addProducto(Producto producto) {
    carrito.addProducto(producto);
  }

  @End
  public void salvarCarrito() {
    em.flush();
  }
}
```

Como se ve, el `flush()` *hay que llamarlo explícitamente en el método que termina la conversación.*

También se puede especificar el `flushMode` *a* `MANUAL` *desde el archivo* `pages.xml`*, por ejemplo en una regla de navecación:*

```
<begin-conversation flush-mode="MANUAL" />
```

4.8.4. Integración con Spring

JBoss Seam permite integrar las SMPC con el framework Spring. Spring tiene una gestión de transacciones avanzada (implementada por medio de alguna de las clases que implementan la interface `PlatformTransactionManager`) pero no posee soporte de contextos de persistencia extendido (solo filtro OSIV). Este gestor de transacciones avanzado de Spring se puede configurar para soportar SMPC usando un contexto de persistencia provisto por Spring.

4.9. Conversaciones anidadas

De los modelos extendidos de conversaciones descriptos en la Sección 3.9 “Modelos extendidos de conversaciones”, Seam soporta el modelo de conversaciones anidadas, lo que significa que Seam mantiene una pila de conversaciones donde se irán incorporando cada una de las conversaciones que se van anidando, donde la última conversación de la pila, es la conversación raíz.

4.9.1. Funcionamiento

Cuando se requiere acceder a un componente, Seam primero busca dentro de la conversación anidada, y si no lo encuentra, sigue buscando en el siguiente elemento de la pila de conversaciones, hasta llegar a la conversación raíz de ser necesario, y si no lo encuentra allí, seguirá buscando en el resto de los scopes.

Dentro del contexto de una conversación anidada, las conversaciones padre son de *solo lectura*, es decir puede se acceder a un componente pero no establecerlo dentro de la conversación. Igualmente, como este componente se obtiene *por referencia* éste puede ser modificado por algún método llamado, sin embargo esto no es deseable ya que permite que el componente se modifique para todas las conversaciones anidadas que comparten el mismo.

Las conversaciones anidadas permiten ser especificadas usando diferentes enfoques:

- Usando anotaciones, una conversación anidada comienza si el tipo de datos que retorna el método es `void` o bien el método no retorna `null`. Simplemente se anota el método con `@Begin(nested=true)`.
- Usando el archivo `pages.xml`, una conversación anidada comienza cuando se accede a una `view-id`. Se debe especificar el tag `<begin-conversation nested="true"/>` en la definición de la página.
- Desde una página o vista, una conversación anidada comienza cuando se selecciona un hipervínculo, especificado con el tag `<s:link ... propagation="nest"/>`.

Una vez comenzada la conversación anidada puede terminarse únicamente ésta, o bien puede terminarse la conversación raíz. En el primer caso la conversación anidada se destruye pero quedan vigentes las conversaciones padre para su posterior uso, y en el segundo caso se destruye toda la pila de conversaciones. Note que terminar una conversación implica también terminar todas las conversaciones hijas que pueden haberse derivado a partir de ella.

De manera análoga al comienzo de la conversación anidada, la terminación puede ser especificada de múltiples maneras:

- Usando anotaciones, si se especifica `@End`, sólo se termina la conversación anidada, y con `@End(root=true)` se termina la conversación raíz y todas sus hijas.
- En el archivo `pages.xml` se puede especificar `<end-conversation/>` para que termine la conversación anidada, o bien `<end-conversation root="true"/>` para que termine la conversación raíz.
- Desde una página se puede usar un hipervínculo para terminar la conversación anidada especificando `<s:link ... propagation="end"/>`, y para que termine la conversación raíz se puede usar `<s:link ... propagation="endRoot"/>`.

4.9.2. Timeout de conversaciones anidadas

La pila de conversaciones asociadas con la conversación en primer plano corriente no es susceptible a la variable `conversation-timeout`. Si la conversación en primer plano es una anidada, cada conversación en la pila también está en primer plano. Si se retoma una conversación de más abajo en la pila (una conversación padre) entonces la conversación anidada pasa a segundo plano.

Por ej. supongamos que existen las conversaciones C_1, C_2, C_3 y C_4, y existen dos pilas de conversación $P_1 = \{C1, C_2, C_4\}$ y $P_2 = \{C1, C_3\}$, donde C_1 es la conversación raíz y las restantes son anidadas. Si la C_4 es la conversación en primer plano, entonces C_2 y C_1 también están en primer plano. Por otro lado C_3 es una conversación que está en segundo plano. Aún si comparte un padre en común con la conversación en primer plano, ésta no es parte de la pila corriente. Similarmente, si C_2 está en primer plano, entonces C_1 también lo está, mientras que C_3 y C_4 están en segundo plano ya que no son parte de la pila corriente.

4.10. Resumen

JBoss Seam es uno de los frameworks de desarrollo de aplicaciones web con soporte nativo de conversaciones más utilizados, y fue uno de los primeros en ser implementados (a mediados del año 2006). Se integra muy bien con JSF y EJBs, al punto de que un EJB puede ser utilizado directamente como un *backing bean* de JSF. Provee soporte de conversaciones *temporarias* (abarcan una sola petición HTTP pero sobreviven un redirect) y de larga duración (abarcan múltiples peticiones HTTP).

Las aplicaciones definen su lógica de navegación en un archivo XML llamado `pages.xml` o bien utilizando un lenguaje de definición de procesos jPDL. Provee soporte de *dependency bijection* lo que le permite operar sobre el estado de la aplicación almacenado en distintos *scopes*. Las conversaciones pueden ser manipuladas a través de una API muy completa.

Soporta una política de expiración de conversaciones en primer y segundo plano, y selectiva por vista. Soporta el uso de casi cualquier combinación de frameworks de mapeo objeto-relacional como JPA o Hibernate, una gestión de transacciones basadas en JTA o CMT (gestionadas por el servidor de aplicaciones), y un contexto de persistencia extendido permite utilizar el mismo `EntityManager` (para JPA) o `Session` (para Hibernate) en las diferentes peticiones que dura la conversación.

Por último, Seam implementa un modelo extendido de conversaciones basado en conversaciones anidadas, lo que lo hace una opción muy potente.

Capítulo 5

Spring Web Flow

En este Capítulo se estudiará el framework Spring Web Flow (abreviado comúnmente SWF por sus letras iniciales), específicamente la versión 2.3 del mismo. Nuevamente se tomarán como base las definiciones y estructura del Capítulo 3, y el alcance del análisis estará limitado al soporte de conversaciones, dirigiendo al lector interesado en un estudio más completo al sitio Web del producto [36].

5.1. Introducción

SWF es un framework de desarrollo de aplicaciones web, que es parte del portafolios de soluciones de la empresa *SpringSource* y como tal es de fácil integración con el framework Spring [34] (un contenedor de Inversión de Control muy utilizado). Asimismo, SWF se integra tanto con el framework Web MVC propio de Spring como con el estándar JSF, de modo que lo hace muy versátil a la hora de seleccionar una tecnología para la confección de las pantallas.

Las aplicaciones desarrolladas usando SWF no requieren un servidor de aplicaciones completamente compatible con Java EE 5 para correr, ya que solo requiere de un contenedor web como Tomcat [15] o Jetty [16]. De hecho el producto principal ofrecido por *SpringSource* es el framework Spring (el contenedor de Inversión de Control) el cual nació como una alternativa simplificada a los servidores de aplicaciones J2EE, por lo tanto no requiere un servidor de aplicaciones para ejecutar; éste corre dentro de un contenedor web, y debido a esta razón todos los productos que dependen de este framework tampoco requieren un servidor de aplicaciones.

5.2. Conversaciones en SWF

SWF captura procesos de negocio en módulos llamados *flows* (o flujos).[1] Según la documentación oficial, un *flow* encapsula una secuencia de pasos reusable que puede ejecutarse en diferentes contextos. Es decir un *flow* es un plano o molde para la interacción que el usuario tiene con la aplicación web, donde esta reacciona a eventos producidos por el usuario para llevar a cabo el proceso de negocio. Se podría ver un flujo como una simple máquina de estados finitos (FSM—Finite State Machine) consistiendo de ciertos estados que definen actividades a ejecutar mientras se progresa a través del flujo. Un estado puede permitir participar a un usuario en el flujo, o puede llamar a servicios o lógica de negocio. El flujo pasa de un estado a otro usando transiciones disparadas por eventos.

La abstracción del flujo de navegación representada en forma análoga a una máquina de estados en realidad es muy utilizado. De hecho durante el diseño preliminar de una aplicación web, el desarrollador puede bosquejar (ya sea en una hoja de papel o bien en una herramienta CASE que permita generar diagramas de estados UML) las interacciones del usuario con la aplicación. Este diseño perfectamente puede tomarse como base para la implementación del flujo en SWF.

Como se mencionó anteriormente, los flujos abstraen una secuencia de pasos *reusable* y pueden ejecutarse en *diferentes contextos* esto es, un flujo puede definir como uno de sus pasos la ejecución de otro flujo (un *subflow* o subflujo) de manera análoga a un método que llama a

[1] De aquí en más en este Capítulo usaremos el término flow y flujo en forma intercambiable.

otro método en Java. El flujo llamador espera hasta que el flujo llamado termine su ejecución, y continuará cuando el subflujo termine. Por ej. en una aplicación de reserva de hoteles y de alquiler de autos, ambos procesos de negocio necesitan pasar por una última secuencia de pantallas cuando se realiza el *checkout*, solicitando un número de tarjeta de crédito para realizar el pago, una pantalla donde el usuario confirma la dirección postal de envío de la factura, y una última pantalla de confirmación de la operación. Este proceso puede modelarse como un flujo, y además ser "llamado" desde los flujos de reserva de hotel y alquiler de autos ya que es el mismo proceso de negocio y por lo tanto puede reutilizarse.

En SWF, la abstracción principal no es la conversación *per se* sino el *flow*. Tanto la conversación como el flujo representan dos *scopes* (o contextos) distintos, pero son muy semejantes ya que están íntimamente relacionados: El contexto del flujo (`flowScope`) es creado cuando un flujo comienza y destruido cuando éste termina, mientras que el contexto de la conversación (`conversationScope`) es creado cuando comienza un flujo de máximo nivel (es decir uno que no se está ejecutando como subflujo) y se destruye cuando este flujo de máximo nivel termina. El `conversationScope` es compartido tanto por el flujo de máximo nivel como por todos sus subflujos.

De lo anterior se deduce que una variable almacenada en `conversationScope` es accesible por el flujo de máximo nivel y por sus subflujos, mientras que una variable almacenada en `flowScope` es accesible dentro del flujo donde fue definida. Por lo general, el desarrollador almacenará estado de la aplicación dentro del `conversationScope` cuando los datos sean compartidos entre flujo y subflujos (aunque el mecanismo de subflujos permite definir parámetros de entrada al mismo) o bien cuando el dato es muy grande (ej. una imagen de varios MB) y no se desea que se replique entre distintos estados del flujo.

En la implementación por defecto, SWF almacena las variables de flujo y conversación en la sesión de usuario y requieren que las mismas implementen la interfase `Serializable`.

5.3. Identificación de conversaciones

En SWF los flujos se identifican por medio del parámetro `execution`, y es un identificador sintético que compone dos parámetros necesarios para identificar unívocamente el estado actual de la aplicación. Por un lado se identifica la *conversación* la cual se quiere retomar, mientras que el segundo parámetro es el identificador de *snapshot* (o fotografía) del estado actual de la aplicación al momento de generar la respuesta mostrada al usuario (este concepto quedará más claro más adelante en este Capítulo). El valor del parámetro `execution` tiene la forma `e<executionId>s<snapshot>`, donde el `executionId` es un identificador que representa al flujo actual que se está ejecutando. Un ejemplo de una URL identificando un flujo se muestra en el Listado 5.1:

```
http://www.misitio.com/miaplicacion/agenda?execution=e3s5
```

Listado 5.1: Forma de una URL especificando el identificador de conversación

Para realizar un hipervínculo o botón que apunte al próximo estado de la aplicación debe construirse una URL con el parámetro `execution` correcto. Para especificar esta URL desde una página JSP debe usarse la variable `${flowExecutionUrl}`, y para denotar el evento que tendrá lugar en el flujo cuando ocurre una acción se utiliza el parámetro `_eventId`, como en el siguiente ejemplo:

```
<a href="${flowExecutionUrl}&_eventId=cancelar">Cancelar</a>
```

Listado 5.2: URL identificando un flujo y un evento en SWF

5.4. Abstracciones para manipular flujos y conversaciones

En SWF es posible acceder o manipular el flujo actual accediendo desde expresiones del EL tanto dentro de páginas JSP como desde los archivos donde se define el flujo, así como desde objetos gestionados por Spring que ejecutan solamente lógica de negocio mediante una API. Los flujos *per se* son definidos en un archivo XML

5.4.1. Acceso mediante EL

SWF permite elegir al desarrollador qué implementación de EL quiere usar: Spring EL, Unified EL (el mismo que se usa en JBoss Seam y Tomcat 6) o bien OGNL, siendo Spring EL la implementación recomendada y la activada por defecto.

Existen algunas variables EL especiales que pueden ser accedidas dentro del flujo:

- `flowScope`: Hace referencia al contexto de almacenamiento del flujo. Se crea cuando el flujo comienza y es destruido cuando el flujo termina.
- `viewScope`: Hace referencia al contexto de almacenamiento de la vista. Se crea cuando el flujo entra en un `view-state` y se destruye cuando sale de ese estado. Este scope es referenciable solamente desde un `view-state`.
- `requestScope`: Hace referencia al contexto de almacenamiento de una variable de request. Se crea cuando el flujo es llamado y es destruido cuando el flujo retorna.
- `flashScope`: Hace referencia al contexto de almacenamiento de una variable *flash*. Se crea cuando un flujo comienza, se borra cada vez que una vista es renderizada, y se destruye cuando el flujo termina.
- `conversationScope`: Hace referencia al contexto de almacenamiento de una variable de conversación. Se crea cuando un flujo de máximo nivel (*i.e.* uno que no se está ejecutando como un subflujo de otro flujo) es comenzado, y es destruido cuando este flujo de máximo nivel termina. El contexto es compartido tanto por el flujo de máximo nivel como por sus subflujos.
- `requestParameters`: Se utiliza para acceder a los parámetros de petición HTTP enviados al servidor.
- `currentEvent`: Se utiliza para acceder a los atributos del evento (`Event`) actual.
- `currentUser`: Provee acceso al `Principal` del usuario autenticado.
- `messageContext`: Provee acceso a un contexto para retornar y crear mensajes en la ejecución de un flujo, incluyendo mensajes de éxito y error.
- `resourceBundle`: Provee acceso a un recurso de mensajes.
- `flowRequestContext`: Provee acceso a la API del `RequestContext`, una representación de la petición actual del flujo.
- `flowExecutionContext`: Provee acceso a la API del `FlowExecutionContext`, el cual es una representación del estado actual del flujo.
- `flowExecutionUrl`: Provee acceso a la URI relativa al contexto para el `view-state` de la ejecución del flujo actual.
- `externalContext`: Provee acceso al ambiente del cliente, incluyendo atributos de la sesión de usuario, usando el API de `ExternalContext`.

Cuando no se hace referencia el scope de la variable, SWF emplea un algorítmo de búsqueda de la misma en el siguiente orden: *request*, *flash*, *view*, *flow*, y *conversation*. Si no se ha encontrado la variable, se eleva una excepción `EvaluationException`.

5.4.2. Archivo de definición de flujo

Los flujos son escritos por los desarrolladores de aplicaciones web usando un lenguaje de definición de flujos basado en XML. Cada flujo se define en un archivo XML por separado.

El esquema del archivo XML tiene el elemento `flow` como raíz y permite varios tipos de elementos para definir el flujo, como se lista en el Cuadro 5.1.

Elemento	Descripción
<action-state>	Ejecuta una o más acciones. El resultado de una acción podría determinar la transación a tomar para el próximo estado en el flujo.
<attribute>	Junto con un elemento <value> anidado, <attribute> declara un meta atributo para describir o anotar el flujo.
<bean-import>	Importa beans (componentes inyectados desde Spring) definidos por el usuario que se resuelven usando expresiones de flujo.
<decision-state>	Evalúa una o más expresiones para decidir el próximo paso en el flujo.
<end-state>	Denota el estado final de un flujo. Cuando entra en el estado final, el flujo es terminado.
<exception-handler>	Designa un bean que manejará las excepciones que ocurran en este flujo.
<global-transitions>	Define una o más transiciones que están disponibles desde cualquier estado.
<input>	Declara un parámetro de entrada provisto por el llamador dentro de este flujo.
<on-end>	Acciones a ejecutar cuando finaliza el flujo.
<on-start>	Acciones a ejecutar cuando el flujo comienza.
<output>	Declara un valor de salida a ser retornado hacia el llamador de este flujo.
<persistence-context>	Crea un contexto de persistencia cuando el flujo comienza. Habilita hacer el *flush* de los cambios cuando el flujo termina. (Es usado con un *transaction manager* definido en Spring.)
<secured>	Usado con *Spring Security* (un framework para proveer seguridad a las aplicaciones Web) para restringir el acceso a un estado a partir de los atributos de seguridad del usuario actual.
<subflow-state>	Comienza otro flujo como un subflujo.
<var>	Declara una variable de instancia del flujo.
<view-state>	Un estado que involucra al usuario en el flujo representándolo con alguna salida y posiblemente con un formulario para su envío.

Cuadro 5.1: Algunos elementos utilizados dentro del archivo de definición de flujo.

Ejemplo de archivo de flujo

En el Listado 5.3 se puede ver un ejemplo de definición de un flujo para hacer la reserva de un hotel, extraído de la aplicación de ejemplo *booking-mvc* que viene con la distribución de SWF, el cual se explicará paso por paso.

En la línea 6, la etiqueta `secured` indica que para entrar a ejecutar el flujo, el usuario actualmente logueado en el sistema debe tener autorizado el rol de `ROLE_USER` (el cual es definido por el desarrollador), de lo contrario el flujo no comenzará.

La línea 8 define una variable de entrada al flujo, la cual debe ser provista como parámetro en una petición HTTP; como está especificado `required="true"` de no estar presente, no podrá iniciarse el flujo. Además, esta variable queda salvada en el contexto del flujo bajo el mismo nombre.

Luego de asignar el parámetro de entrada, en las líneas 10 a 12 se indica que al iniciar el flujo debe ejecutarse una expresión y asignar el resultado de la misma en una variable. En este caso, se solicita al `bookingService` que cree una nueva reserva sobre el hotel con identificador `hotelId` a nombre del usuario actualmente operando el sistema `currentUser.name`. El resultado de la llamada al método `createBooking` es un nuevo objeto asignado a la variable `booking`, la cual queda almacenada en el contexto del flujo (`flowScope`).

Las líneas 14 a 31 definen un `view-state`, denominado `enterBookingDetails`, que es un estado en el que se renderiza una vista para que el usuario tenga la posibilidad de interactuar con la aplicación. El atributo `model` define un objeto al cual se enlaza la vista, para poder presentar formularios pre-llenados con los campos ligados a los atributos del objeto usado como modelo, y luego al enviar el formulario se puede hacer la validación de los datos de entrada. En este caso se utiliza el objeto `booking` recién creado para almacenar los datos de la reserva, y los atributos a ser ligados para presentarlos en el formulario son los especificados entre las líneas 15 a 25. Por convención, al ser el primer estado definido en el archivo XML, es el estado inicial en que entra el flujo al ser creado. En las líneas 26 a 28 aparece un elemento `on-render` que es una solicitud de que se realice alguna acción antes de renderizar la vista; en este caso se solicita renderizar una parte de una una vista (esto se realiza comúnmente en aplicaciones Ajax donde solo se necesita redibujar un fragmento de una vista en particular. Finalmente, las líneas 29 y 30 definen transiciones de estado, las cuales dependen del botón que seleccione el usuario en la pantalla al enviar el formulario: si presionó el botón `proceed`, el flujo pasa al estado `reviewBooking` actualizando los datos del objeto `booking`, mientras que si presiona `cancel`, pasa al estado `cancel` sin actualizar los datos (atributo `bind="false"`).

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <flow xmlns="http://www.springframework.org/schema/webflow"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://www.springframework.org/schema/webflow http://
          www.springframework.org/schema/webflow/spring-webflow.xsd">
5
6   <secured attributes="ROLE_USER" />
7
8   <input name="hotelId" required="true" />
9
10  <on-start>
11    <evaluate expression="bookingService.createBooking(hotelId, currentUser.
          name)" result="flowScope.booking" />
12  </on-start>
13
14  <view-state id="enterBookingDetails" model="booking">
15    <binder>
16      <binding property="checkinDate" />
17      <binding property="checkoutDate" />
18      <binding property="beds" />
19      <binding property="smoking" />
20      <binding property="creditCard" />
21      <binding property="creditCardName" />
22      <binding property="creditCardExpiryMonth" />
23      <binding property="creditCardExpiryYear" />
24      <binding property="amenities" />
25    </binder>
26    <on-render>
27      <render fragments="body" />
28    </on-render>
29    <transition on="proceed" to="reviewBooking" />
30    <transition on="cancel" to="cancel" bind="false" />
31  </view-state>
32
33  <view-state id="reviewBooking" model="booking">
34    <on-render>
35      <render fragments="body" />
36    </on-render>
37    <transition on="confirm" to="bookingConfirmed">
38      <evaluate expression="bookingService.persistBooking(booking)" />
39    </transition>
40    <transition on="revise" to="enterBookingDetails" />
41    <transition on="cancel" to="cancel" />
42  </view-state>
43
44  <end-state id="bookingConfirmed"/>
45
46  <end-state id="cancel" />
47 </flow>
```

Listado 5.3: Forma de una URL especificando el identificador de conversación

Entre las líneas 33 y 42 se define el estado `reviewBooking` cuya intención es mostrar los datos ingresados por el usuario para que confirme o rectifique su reserva. Define tres transiciones que dependen de la selección que haga el usuario: Si presiona el botón `confirm`, el flujo evaluará una expresión (línea 38) que salva el objeto reserva en la base de datos y luego pasa al estado `bookingConfirmed`; si presiona el botón `revise`, el flujo vuelve al estado `enterBookingDetails` para que el usuario modifique el formulario previamente enviado, y si selecciona `cancel`, se va al estado `cancel`.

Finalmente, las líneas 44 y 46 definen estados de fin con la etiqueta `end-state`, lo que significa que cuando se alcance este estado, se dará por terminado el flujo.

5.4.3. API

En SWF las tareas que se ejecutan durante el ciclo de vida del flujo las realizan *beans* definidos y gestionados por el framework Spring, los cuales son inyectados automáticamente dentro del flujo por medio de la etiqueta `<evaluate>` usando el lenguaje de expresión (EL). La lógica de presentación del flujo suele ser modelada en su totalidad por el flujo en el archivo XML

que lo define, mientras que la lógica de negocio es resuelta por los *beans* de servicio y DAOs referenciados desde el flujo.

Siendo tan potente y expresivo el lenguaje XML para el modelado del flujos, no se requiere tener acceso a una API que permita gestionar los flujos programáticamente en Java. No obstante es posible hacerlo: Para utilizar SWF es necesario definir ciertos servicios en un archivo `applicationContext.xml` de Spring. Estos servicios controlan y configuran el comportamiento de SWF y, como servicios son *beans* definidos en Spring, es posible inyectarlos dentro de cualquier objeto nuestro que así lo requiera: esto podrá permitir un manejo más granular en la configuración y comportamiento de SWF, pero realmente no es necesario para modelar la lógica de presentación de la aplicación. En las siguientes subsecciones se describirán algunos de estos servicios.

Flow Executor

El *Flow Executor* maneja la ejecución de los flujos, y se configura en Spring con el elemento `<flow:flow-executor>`:

```
<flow:flow-executor id="flowExecutor" />
```

Listado 5.4: Definición de un *Flow Executor*

Cualquier *bean* definido en Spring que quiera usar una instancia del *Flow Executor*, deberá declarar un objeto de tipo `org.springframework.webflow.FlowExecutor` (que es una interface Java):

```
import org.springframework.webflow.FlowExecutor;

public class MiBean {
  @Autowired
  private FlowExecutor flowExecutor;
}
```

Listado 5.5: Inyección de un *Flow Executor* dentro de un *bean* de Spring

Flow Registry

Los flujos creados con SWF se definen en archvos XML separados, pero es necesario decirle a Spring dónde puede encontrar esos archivos de definición de flujos; para esa tarea se configura un *Flow Registry*:

```
<flow:flow-registry id="flowRegistry" base-path="/WEB-INF/flows">
  <flow:flow-location-pattern value="/**/*-flow.xml" />
</flow:flow-registry>
```

Listado 5.6: Definición de un *Flow Registry*

En el ejemplo del Listado 5.6, se instruye a SWF que debe buscar todos los archivos de definición de flujos dentro del directorio `/WEB-INF/flows` (camino base), donde la búsqueda se restringe a los archivos con nombre de la forma `XXXX-flow.xml`, inclusive dentro de subdirectorios del camino base.

Integración con Spring MVC

SWF provee un *handler adapter* para Spring MVC llamado `FlowHandlerAdapter`. Este *handler adapter* hace de puente entre el `DispatcherServlet` (el servlet punto de entrada de Spring MVC) y el *flow executor* (punto de entrada de SWF), manejando peticiones HTTP y manipulando el flujo basándose en esas peticiones. Para configurar un `FlowHandlerAdapter` hace falta la siguiente declaración:

```
<bean class="org.springframework.webflow.mvc.servlet.FlowHandlerAdapter">
  <property name="flowExecutor" ref="flowExecutor"/>
</bean>
```

Listado 5.7: Declaración del *handler adapter*

El `DispatcherServlet` de Spring MVC sabe cómo despachar las peticiones HTTP consultando con uno o más *handler mappings*. Para integrarlo con SWF hay que configurar un `FlowHandlerMapping`, quien es el que ayuda al `DispatcherServlet` a enviar peticiones de flujo al `FlowHandlerAdapter`:

```
<bean class="org.springframework.webflow.mvc.servlet.FlowHandlerMapping">
  <property name="flowRegistry" ref="flowRegistry" />
</bean>
```

Listado 5.8: Declaración del `FlowHandlerMapping`

5.4.4. Listeners

Se pueden registrar *Listeners* a los flujos para poder ser notificados de eventos que suceden durante la ejecución de los mismos. Dentro de las opciones del *Flow Executor*, se pueden añadir estos listeners (note también que pueden adjuntarse *listeners* solo a flujos específicos, como muestra la línea 4):

```
1 <webflow:flow-executor id="flowExecutor">
2   <webflow:flow-execution-listeners>
3     <webflow:listener ref="securityFlowExecutionListener" />
4     <webflow:listener ref="securityListener" criteria="securedFlow1,
          securedFlow2"/>
5   </webflow:flow-execution-listeners>
6 </webflow:flow-executor>
```

Listado 5.9: Declaración de `Listeners`

Cualquier objeto que desee recibir notificaciones de eventos que sucedan durante la ejecución de un flujo debe implementar la interface `FlowExecutionListener` o bien extender la clase `FlowExecutionListenerAdapter`. En general, un *listener* permitirá insertar comportamiento en puntos bien definidos dentro del ciclo de vida de de la ejecución de un flujo. Por ejemplo, un *listener* podría aplicar chequeos de seguridad, evitando que un flujo no comience su ejecución o no pueda entrar en cierto estado si el usuario actual no tiene los permisos necesarios. Otro *listener* podría hacer un seguimiento de la historia de navegación de ejecución del flujo para soportar *bread crumbs* (*i.e.* mostrar un rastro de los lugares por donde ha estado el usuario). Otro podría ejecutar una tarea de auditoria, o establecer conexiones a un recurso que sea transaccional, como por ejemplo obtener un `EntityManager` de JPA para poder acceder a una base de datos.

SWF trae implementados cuatro *listeners* con funcionalidad muy común, a saber:

- `SecurityFlowExecutionListener`: Provee la integración necesaria con el framework Spring Security [35], a efectos de que funcionen las etiquetas `<secured>` dentro del archivo de definición del flujo. Cuando haya algún problema de seguridad, y no se encuentra logueado ningún usuario en el sistema, Spring Security automáticamente presentará al usuario con una pantalla de login que permitirá al usuario autenticarse y seguir ejecutando el flujo en caso de que tenga los permisos necesarios.
- `JpaFlowExecutionListener`: Implementa *Flow Managed Persistence Context* (FMPC) usando el framework de mapeo objeto-relacional JPA. (Más información en la Sección 5.8.)
- `HibernateFlowExecutionListener`: Implementa FMPC usando el framework de mapeo objeto-relacional Hibernate. (Más información en la Sección 5.8.)
- `FlowFacesContextLifecycleListener`: Sirve para integrar SWF con JSF (Java Server Faces). Crea una instancia de un `FlowFacesContext` cuando la petición hacia un flujo es enviada, y lo libera cuando la petición ha sido procesada. Un `FlowFacesContext` es un `FacesContext` de JSF, que delega toda la funcionalidad de manejo de mensajes hacia un `MessageSource` accesible al flujo actual ejecutando la petición. Adicionalmente, sirve para gestionar el `flashScope` de manera de que los objetos de JSF estén disponibles en una próxima petición facilitando la implementación del patrón Post-Redirect-Get.

5.5. Terminación de flujos

SWF posee un enfoque muy simple y predecible a lo que respecta a la culminación de los flujos, tanto cuando terminan normalmente como bajo condiciones de excepcionales.

5.5.1. Terminación normal

Como se mostró en el Cuadro 5.1, un flujo termina cuando se completa la ejecución asociada a la etiqueta `<end-state>`. Por un lado, al llegar al estado final de un flujo raíz efectivamente termina el flujo, mientras que cuando se llega al estado final de un subflujo (*i.e.* un flujo llamado desde otro flujo), efectivamente termina ese subflujo, pero el flujo llamador sigue vigente, retornando en el estado en que se encontraba cuando se llamó el subflujo.

Además, la etiqueta `<end-state>` permite especificar algunos atributos que modifican el comportamiento de la terminación de flujos, como el atributo `view`, que permite que se especifique una vista en particular, o URL de respuesta cuando el flujo termina. A menudo no es especificada cuando el llamador maneja el resultado de este flujo y gestiona la respuesta. Cuando es especificada, puede ser cualquier página JSP, una URL externa, o bien un redirect hacia una URL externa.

5.5.2. Comportamiento ante la presencia de excepciones

SWF posee varias opciones de manejo de excepciones pero ninguna de ellas termina la ejecución del flujo, sino que solo se interrumpe (en el peor de los casos) cuando la excepción nunca es manejada por alguno de los mecanismos descritos a continuación.

Una de las opciones más simples es gestionar la excepción dentro de un objeto Action llamado desde el flujo, para luego retornar a una transición de estado en particular si se presenta la excepción. Por ejemplo, para realizar una reserva de hotel, el archivo de definición del flujo, puede evaluar una expresión de la siguiente manera:

```
<evaluate expression="bookingAction.makeBooking(booking, flowRequestContext)"
    />
```

Donde el código fuente de la clase Action sería algo así:

```
public class BookingAction {

  @Autowired
  private BookingService bookingService;

  public String makeBooking(Booking booking, RequestContext context) {
    try {
      BookingConfirmation confirmation = bookingService.make(booking);
      context.getFlowScope().put("confirmation", confirmation);
      return "success";
    } catch (RoomNotAvailableException e) {
      context.addMessage(new MessageBuilder().error().
        defaultText("Ya no hay habitaciones disponibles en este hotel").build
            ());
      return "error";
    }
  }
}
```

En el ejemplo se ve que la invocación al método `BookingAction.makeBooking()` llama a un método del objeto `bookingService` inyectado, quien se encarga de hacer la reserva: Si la reserva pudo hacerse se prosigue con la ejecución del flujo según la transición `success`, pero cuando ya no hay más lugar en el hotel, puede llegar a elevar una excepción del tipo `RoomNotAvailableException`; en tal caso, se agrega un mensaje de error y se continúa la ejecución del flujo hacia el estado declarado para la transición `error`.

Otra forma de gestionar excepciones es por medio de un objeto que implemente la interface `FlowHandler`, a través de su método `handleException()`. Se utiliza cuando se desea tener un control más fino sobre excepciones de flujo no manejadas. El comportamiento por defecto intenta reiniciar el flujo cuando un usuario intenta acceder a una ejecución de flujo que ya ha terminado o expirado. Cualquier otra excepción es re-elevada hacia la infraestructura de `ExceptionResolver` de Spring MVC. Para instalar un `FlowHandler`, simplemente hay que declararlo como un *bean* dentro del archivo de configuración de Spring, usando como atributo `id` el mismo nombre del flujo donde operará.

SWF también permite hacer manejo de excepciones en forma declarativa, dentro del archivo de definición de flujo, usando la etiqueta `<exception-handler>`, la cual referencia un *bean* (a través del atributo `bean`) que implementa la interface `FlowExecutionExceptionHandler`. Un

<exception-handler> puede ser declarado a nivel de un estado o a nivel del flujo; cuando ocurre una excepción, SWF intentará manejarla usando un <exception-handler> declarado dentro del estado desde donde se origina la excepción, y si no encuentra ninguno, lo hará usando el declarado a nivel de flujo. Si no encuentra ninguno el flujo será interrumpido. Por ejemplo:

```
<flow>
  ...
  <view-state id="mostrarView" view="view">
    <transition on="next" to="nextState" />
    <exception-handler bean="handlerDeclaradoDentroDelEstado"/>
  </view-state>
  ...
  <exception-handler bean="handlerDeclaradoANivelDelFlujo"/>
  ...
</flow>
```

Finalmente, también puede seguirse una transición en particular cuando ocurre una excepción, declarando la misma usando el atributo on-exception:

```
<flow>
  ...
  <view-state id="mostrarView" view="view">
    <transition on-exception="java.sql.SQLException" to="errorBaseDeDatos" />
  </view-state>
  ...
  <global-transitions>
    <transition on-exception="java.lang.Exception" to="errorCualquiera" />
  </global-transitions>
</flow>
```

El valor del atributo on-exception es el nombre de clase completo de la excepción. Una transición con atributo on-exception definido dentro de un estado adjuntará un exception handler a ese estado, mientras que cuando se define dentro del elemento <global-transitions> (donde se declaran transiciones que pueden ocurrir en cualquier estado del flujo) se adjuntará un exception handler al flujo.

5.5.3. Acceso a flujos ya terminados

Una vez que un flujo termina el procesamiento de un <end-state>, se envía la respuesta al cliente especificando una vista en particular en el <end-state>; si esta vista no se especifica, se asume que ya se ha generado una respuesta directamente (por ej. usando un Action).

Un caso especial es el de los subflujos: cuando un subflujo termina, la selección de la vista queda bajo la responsabilidad del flujo padre que retoma su ejecución.

Otro dato a tener en cuenta es que un <end-state> no soporta el patrón POST-REDIRECT-GET, lo que significa que no se podrá hacer un *redirect* hacia una vista en particular y esperar que se pueda acceder a los datos almacenados en el flowScope ya que el flujo ya ha sido destruido.

Por otro lado, el intento de acceder a un flujo ya terminado, usando el parámetro execution en la petición HTTP, hace que se genere una excepción de tipo NoSuchFlowExecutionException. Se puede definir un FlowHandler implementando el método handleException() a efectos de seleccionar una vista a mostrar cuando sucede esta excepción, como en el siguiente ejemplo:

```
public class BookingFlowHandler extends AbstractFlowHandler {

  @Override
  public String handleException(FlowException e,  HttpServletRequest request,
       HttpServletResponse response) {
    if (e instanceof NoSuchFlowExecutionException) {
      return "hoteles/buscar";
    } else {
      // otra excepcion que genere un error
      throw e;
    }
  }

}
```

5.6. Políticas de expiración de conversaciones

SWF tiene una política de expiración de conversaciones por cantidad de conversaciones activas dentro de la sesión de usuario, como se describió en el Capítulo 3, Sección 3.6.4. El servicio que actúa como punto de entrada para gestionar conversaciones en SWF es un objeto que implementa la interface `ConversationManager`, y el servicio que viene configurado por defecto es el `SessionBindingConversationManager`, que es una implementación que almacena las conversaciones en la sesión de usuario.

Usando la propiedad `maxConversations` se puede limitar el número de conversaciones activas concurrentemente dentro de una sesión. Si el número máximo es excedido, el *conversation manager* automáticamente terminará la conversación más antigua. El valor por defecto es 5, el cual debería ser suficiente para la mayoría de los casos. Si se establece en -1, significa que no se limitará la cantidad de conversaciones por sesión, mientras que si se establece en 1 solo será permitido tener una conversación activa por sesión de usuario. A continuación un ejemplo de su configuración en un archivo de configuración de Spring:

```
<beans ...>
  ...
  <webflow:flow-executor id="flowExecutor">
    <webflow:flow-execution-repository max-executions="10" max-execution-
        snapshots="5"/>
  </webflow:flow-executor>
  ...
</beans>
```

En el ejemplo se ve el atributo `max-executions` establecido a 10, que es el equivalente a la propiedad `maxConversations` del *conversation manager*. El ejemplo también muestra el uso del atributo `max-execution-snapshots`, que pone una cota superior al número de *snapshots* que pueden ser almacenadas durante la ejecución de un flujo. Como se verá posteriormente en la Sección 5.9, SWF basa su ejecución de flujos en *continuations*. Cada estado por los que pasa un flujo genera una copia de las variables de estado del flujo (un *snapshot* o fotografía del flujo) y se almacena en el servidor ante la eventualidad que el usuario decida presionar el botón de Retroceso del navegador Web y quiera continuar la ejecución del flujo desde ese punto anterior, pero con otros datos. Como estas copias de estados genera un costo considerable en la memoria del servidor, también se puede configurar este parámetro para poder limitar este número. Para inhabilitar la generación de *snapshots*, se puede establecer este valor en 0, mientras que un valor de -1 habilita un número ilimitado de *snapshots*. En el ejemplo de arriba, un valor de 5 indica que el usuario puede presionar el botón de retroceso del navegador hasta 5 veces y retomar la ejecución del flujo desde el estado en el que estaba en ese momento; si presionase el botón de retroceso 6 veces, SWF sería incapaz de encontrar el *snapshot* correspondiente y por lo tanto no sería posible continuar ejecutando desde esa página. Finalmente, cuando un flujo termina su ejecución, todos los *snapshots* asociados a esa ejecución también se eliminan automáticamente.

5.7. Control de concurrencia

Puesto simplemente, SWF asegura al desarrollador de la aplicación que la misma conversación no podrá ser accedida por dos threads a mismo tiempo, es decir la ejecución es *thread-safe*.

Como se describió anteriormente, el punto de entrada para gestionar conversaciones es una instancia de `ConversationManager`, del cual la implementación por defecto provista por SWF es la clase `SessionBindingConversationManager`. Cada vez que la aplicación recibe una petición HTTP, SWF llama al método `getConversation(ConversationId)`, al cual se le pasa como parámetro un objeto de tipo `ConversationId` y retorna un objeto que implementa la interface `Conversation`, y esta interface provee métodos `lock()` y `unlock()` para operar con ellas en forma segura.

De la misma manera, el `FlowExecutionRepository` tiene un método llamado `getLock-(FlowExecutionKey)` donde recibe como parámetro el identificador de flujo (usando la clase `CompositeFlowExecutionKey`, como se describió en la Sección 5.3). Con esto SWF se asegura que todo procesamiento en la ejecución de un flujo se realiza en forma serializada de otros accesos que puedan llegar a tener lugar sobre la misma ejecución de flujo provenientes de otra petición HTTP.

5.8. Integración con Mapeos Objeto-Relacional

SWF se integra tanto con Hibernate como con JPA, que son las opciones de mapeo Objeto-Relacional más utilizadas, y soporta el uso de contextos de persistencia extendidos, como se describe en el Capítulo 3, Sección 3.8.5, a través de la funcionalidad de *flow managed persistence* (persistencia gestionada por el flujo) donde un flujo puede crear, hacer commit y cerrar un contexto de persistencia en forma automática. También se puede utilizar encapsulando la gestión del contexto de persistencia dentro de la capa de servicio de la aplicación, de modo que la capa de presentación no se involucra con el código de persistencia, y se trabaja con objetos en estado *detached* que son pasados y retornados por la capa de servicio.[2]

5.8.1. Persistencia gestionada por flujo

Este mecanismo (en inglés *Flow Managed Persistence*) almacena un `PersistenceContext` de JPA (o una `Session` de Hibernate) como una variable en `flowScope` cuando el flujo comienza, usa ese contexto de persistencia durante toda la ejecución del flujo, y hace un commit de los cambios realizados a las entidades persistentes cuando el flujo termina.

Se trata de una implementación de gestión de persistencia usando un contexto de persistencia extendido, como se vio en el Capítulo 3, Sección 3.8.5, y como tal permite aislar cambios en objetos persistentes modificados durante la ejecución del flujo hasta el momento en que se hace un commit de la transacción al finalizar el flujo; si el flujo se cancela o expira (o bien expira la sesión de usuario), los cambios nunca afectan al almacenamiento durable de la base de datos. También se puede usar implementando *optimistic locking* para evitar interferencias en los datos modificados concurrentemente por otro usuario.

Para poder usar la funcionalidad de *Flow Managed Persistence* en un flujo, hay que incorporar la etiqueta `<persistence-context>` al archivo de definición de flujo (línea 3, listado de abajo). Cuando el flujo llega a un estado final como el que se muestra en la línea 5, se hará el commit de todos los cambios realizados a los objetos persistentes siempre y cuando se incluya el atributo `commit` establecido a `true`:

```
1 <flow>
2   ...
3   <persistence-context />
4   ...
5   <end-state id="confirmarReserva" commit="true" />
6   ...
7 </flow>
```

Luego hay que configurar un `FlowExecutionListener` que escuche los eventos que van sucediendo durante la ejecución del flujo, tomando las acciones necesarias para implementar el mecanismo. SWF ya tiene implementado dos *listeners*:

- `JpaFlowExecutionListener`: Implementa *Flow Managed Persistence* usando JPA.
- `HibernateFlowExecutionListener`: Implementa *Flow Managed Persistence* usando Hibernate.

A continuación, se ve una configuración del archivo XML de Spring, utilizando JPA:

```
<bean id="dataSource"
    class="org.springframework.jdbc.datasource.DriverManagerDataSource">
  <property name="driverClassName" value="org.hsqldb.jdbc.JDBCDriver" />
  <property name="url" value="jdbc:hsqldb:mem:hotel" />
  <property name="username" value="usuario" />
  <property name="password" value="clave" />
</bean>

<bean id="entityManagerFactory"
    class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean"
        >
  <property name="dataSource" ref="dataSource" />
    <property name="jpaVendorAdapter">
      <bean class="org.springframework.orm.jpa.vendor.
          HibernateJpaVendorAdapter">
```

[2] Aunque no se describirá aquí esta forma de acceso ya que es el comúnmente usado por las aplicaciones que utilizan un framework web MVC sin soporte de conversaciones.

```
        <property name="database" value="HSQL" />
        <property name="showSql" value="true" />
      </bean>
    </property>
</bean>

<bean id="transactionManager" class="org.springframework.orm.jpa.
    JpaTransactionManager">
  <property name="entityManagerFactory" ref="entityManagerFactory"/>
</bean>

<bean id="jpaFlowExecutionListener"
  class="org.springframework.webflow.persistence.JpaFlowExecutionListener">
  <constructor-arg ref="entityManagerFactory" />
  <constructor-arg ref="transactionManager" />
</bean>

<webflow:flow-executor id="flowExecutor" flow-registry="flowRegistry">
  <webflow:flow-execution-listeners>
    <webflow:listener ref="jpaFlowExecutionListener" />
  </webflow:flow-execution-listeners>
</webflow:flow-executor>
```

La forma en que funciona el *listener* es la siguiente: Cuando comienza una nueva ejecución del flujo, el *listener* creará un nuevo `EntityManager` y lo almacenará en `flowScope`. Cada vez que se realice un acceso a datos persistentes dentro de un DAO gestionado por Spring, se hará uso de este `EntityManager` en forma automática y con el modo de *flush* configurado para que se efectúe en forma manual. Cuando termina el flujo (con el atributo `commit="true"`) se forzará el *flush* del contexto de persistencia, efectivamente persistiendo los cambios a la base de datos.

Una aclaración especial con respecto al manejo del mecanismo de gestión del contexto de persistencia y la ejecución de subflujos: Un contexto de persistencia gestionado por flujo será automáticamente propagado hacia los subflujos asumiendo que el subflujo también declare la etiqueta `<persistence-context/>`. Cuando un subflujo reusa el contexto de persistencia creado por un flujo padre, este ignora los atributos `commit="true"` cuando un estado final es alcanzado, por lo tanto se difiere la decisión final de realizar el commit (o no) al flujo padre.

5.9. Continuations

SWF usa el modelo extendido que utiliza el concepto de *continuations*, como se describió en el Capítulo 3, Sección 3.9.2. La implementación de *continuations* utilizada se basa en sacar fotografías (*snapshots*) del estado de las variables del flujo luego de procesar cada petición HTTP.

Cuando se procesa una petición HTTP durante la ejecución de un flujo, SWF leerá el identificador de la ejecución del flujo, llamado `execution`, que viene como un parámetro en la petición GET HTTP, por ej. `execution=e5s2`. Como se describió en la Sección 5.3, el mismo consta de dos partes: la primera parte identifica la ejecución del flujo en particular, y la segunda parte a un *snapshot* en particular dentro de esa ejecución de flujo. Cada *snapshot* contiene las variables del flujo en el que estaban la última vez que se ejecutó.

El objeto encargado de almacenar las ejecuciones de flujo activas es el `FlowExecutionRepository`, donde buscará retomar la ejecución del flujo buscando por el parámetro `execution` un *snapshot* de la ejecución del flujo en el estado en que fue salvado por última vez. Si encuentra el *snapshot* buscado, se re-inicializa el flujo con las variables con los valores guardados previamente, y se continúa con el procesamiento del flujo. Puede pasar que no se encuentre el *snapshot* buscado debido a que ya ha sido desalojado como se describió en la Sección 5.6.

Los *snapshot* se crean usando el mecanismo de serialización de objetos de Java, razón por la cual, todo objeto que se guarde en un *scope* de SWF debe implementar la interface `java.io.Serializable`. Específicamente, el repositorio serializará el objeto `FlowExecution` y almacenando el resultado en bytes en la sesión de usuario.

Además, este mecanismo de implementación de *continuations* funciona de igual manera tanto para la ejecución de un flujo raíz como para un subflujo, lo que lo hace muy potente y a la vez simple.

En la Figura 5.1 se puede ver un ejemplo del ciclo de vida de la ejecución de un flujo que consiste de tres estados, junto con la representación de los *snapshots* almacenados en el `FlowExecutionRepository`:

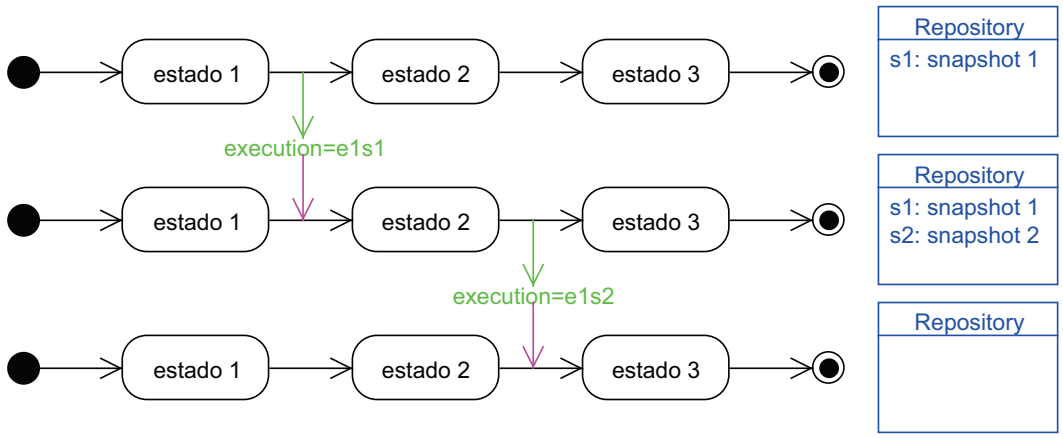

Figura 5.1: Ejemplo de implementación de *continuations* en SWF.

1. La primera petición que llega al *flow executor* no contiene ningún identificador de ejecución de flujo, lo que hace que el *flow executor* inicie una nueva ejecución de flujo (con identificador `e1`) para este flujo simple con tres estados. El flujo progresa desde el primer estado hacia el segundo estado (un `view-state`) y pausa. Antes de retornar la respuesta al navegador, el repositorio de la ejecución del flujo genera un *snapshot* del `FlowExecution` y le asigna una clave única `s1`. Esta clave es embebida en la vista renderizada para que pueda ser retomada la ejecución del flujo en una petición subsecuente. Al finalizar la primera petición, el repositorio contiene el *snapshot* `snapshot 1`, indizada por la clave `s1`.

2. La segunda petición que llega contiene el parámetro `execution=e1s1` como identificador de ejecución de flujo. El *flow executor* busca y restaura esta ejecución de flujo usando el valor de las variables almacenados en el `snapshot 1`. El flujo retoma la ejecución, se mueve hacia el tercer estado y pausa. En este punto, el repositorio toma otro *snapshot* (`snapshot 2`) del `FlowExecution` y le asigna una nueva clave única `s2`, la cual es embebida en la vista renderizada. Note que el primer *snapshot* aún sigue en memoria: esto le permite al usuario presionar el botón de Retroceso del navegador, saltando hacia atrás a la petición previa, y continuar la ejecución del flujo desde ese punto. Es más, abrir una nueva pestaña sobre la misma conversación permitiría al usuario continuar desde el *snapshot* corriente (`snapshot 2`) desde esa ventana.

3. La tercer petición continúa desde la *continuation* identificada con el parámetro `execution=e1s2`. Aquí el flujo retoma el procesamiento y termina alcanzando su `end-state`. Como consecuencia, la ejecución del flujo y todos sus *snapshots* serán removidos del repositorio. Esto previene la posibilidad de recibir un envío duplicado, aún cuando se usen *continuations*: si el usuario presiona el botón de Retroceso del navegador para ir hacia atrás hacia la petición dos, se producirá un error ya que el `snapshot 2` no estará disponible, ya que fue eliminado junto con todos los demás *snapshots* cuando la petición con el `end-state` terminó la conversación corriente.

5.10. Resumen

En este Capítulo se dio un vistazo a las características del framework Spring Web Flow (SWF), el cual depende de su integración con Spring para funcionar.

El elemento de procesamiento principal de SWF es el flujo (*flow*), que encapsula una secuencia de pasos reusable capaz de ejecutarse en diferentes contextos. Es reusable porque un flujo puede llamarse desde otro flujo de manera análoga a un método que llama a otro método en un lenguaje de programación orientado a objetos.

Un flujo es comparable en funcionalidad a una máquina de estados finito, donde cada estado representa actividades a ejecutar, y el flujo pasa de un estado a otro mediante las transiciones

disparadas por eventos. De hecho, el desarrollador puede prototipar la navegación en forma de una máquina de estados para luego hacer una traducción directa a un archivo XML que define el flujo.

SWF adopta el modelo extendido de conversaciones basado en *continuations*. Cada paso en la ejecución de un flujo tiene un identificador de ejecución de flujo compuesto por dos partes: la parte que identifica la instancia actual del flujo que se está ejecutando, y la parte que identifica el *snapshot* dentro del flujo al que se hará referencia para continuar la ejecución cuando el usuario envíe un formulario al servidor. El *snapshot* es una fotografía del estado de la aplicación al momento de recibir un evento.

La política de expiración de conversaciones tiene en cuenta la cantidad máxima de conversaciones y flujos que se pueden ejecutar concurrentemente: cuando se supera esa cantidad, se selecciona uno para su descarte.

Con respecto al uso con frameworks de mapeo objeto-relacional, SWF ofrece integración con JPA e Hibernate. El flujo soporta contextos de persistencia extendidos (llamado *Flow Managed Persistence*) utilizando el mecanismo de gestión de transacciones de Spring.

Capítulo 6

Una Solución para el Soporte de Conversaciones en Struts 2

En este Capítulo se propone una solución para el soporte de conversaciones específica para el framework Struts 2 [5], que es uno de los frameworks de desarrollo de aplicaciones web con arquitectura MVC más utilizados del mundo. Dado que Struts 2 no provee un soporte nativo de conversaciones, proveer una implementación sería muy conveniente ya que se podría ampliar la gama de soluciones que se pueden programar con Struts 2, aprovechando la experiencia de miles de desarrolladores alrededor del planeta.

La siguiente Sección describe los conceptos básicos de Struts 2, pero no profundiza en cada uno de ellos en detalle, de manera que para obtener más información sobre el framework se dirije al lector a leer la documentación oficial [6].

6.1. Introducción a Struts 2

Struts 2 es un framework que sigue la arquitectura MVC (*Model-View-Controller*), es decir están separadas las responsabilidades de la aplicación en el Modelo, la Vista y el Controlador. En líneas generales el Modelo es el encargado de implementar la lógica de presentación de acuerdo a las necesidades de la aplicación web. La Vista se encarga de tomar la información producida por el Modelo y mostrarla al usuario a efectos de poder interactuar con la aplicación, mientras que el Controlador se encarga de gestionar tanto los Modelos como las Vistas.

Struts 2 no necesita un servidor de aplicaciones completamente compatible con el estándar Java EE 5, ya que puede correr sobre cualquier contenedor web que implemente la especificación estándar de Servlets y páginas JSP (como Tomcat o Jetty).

6.1.1. Arquitectura

En la Figura 6.1 se muestra un diagrama que representa la arquitectura interna del framework. A continuación se describirán los elementos esenciales que componen esta arquitectura:

- `HttpServletRequest` y `HttpServletResponse`: Son los objetos estándar definidos por las especificaciones de Servlets que representan una petición HTTP y su respuesta, respectivamente. Struts 2 los utiliza extensivamente durante todo el ciclo de vida de la petición del usuario.
- `StrutsPrepareFilter`, `StrutsExecuteFilter`, y otros filtros: Son filtros definidos en el archivo `web.xml` de la aplicación web. Representan el punto de entrada para la ejecución del framework. Estos dos filtros preparan y ejecutan la petición HTTP cuando se necesita por alguna razón separar estas dos etapas para poder integrar Struts 2 con filtros pertenecientes a otros frameworks (por ej. Sitemesh [33]). Cuando esta separación no se necesita es conveniente usar el `StrutsPrepareAndExecuteFilter`.
- `ActionProxy`: Es un objeto que abstrae la petición, preparando la misma para su ejecución mediante la configuración de un `ActionInvocation`. La definición del Action a ejecutar se carga mediante llamadas al `ConfigurationManager`, donde la forma más común de

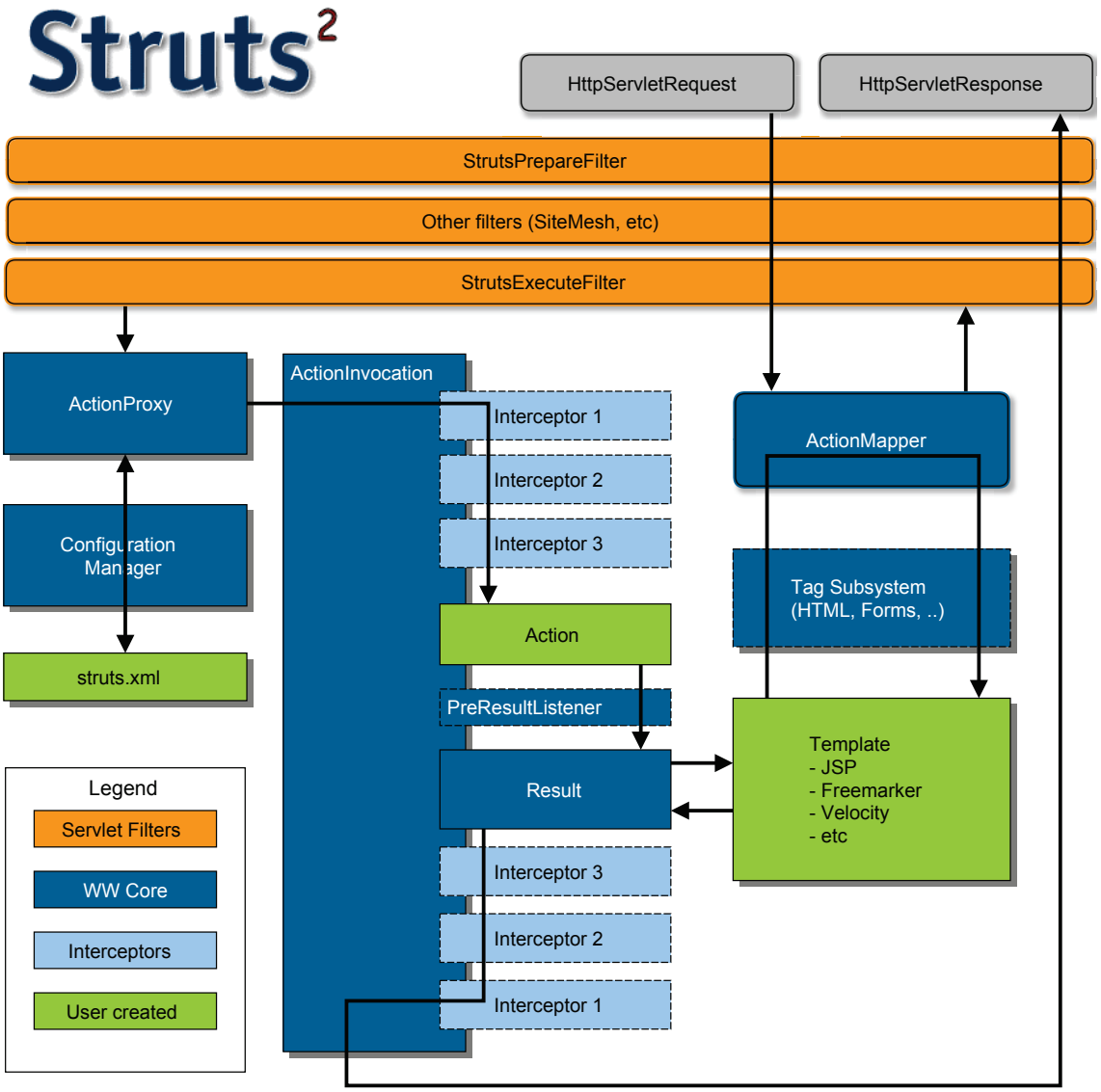

Figura 6.1: Arquitectura de Struts 2 (extraída de [6]).

definirlas es mediante la escritura del archivo `struts.xml` (pero también se pueden definir Actions mediante anotaciones en el código fuente).

- `ActionInvocation`: Representa el estado del Action en ejecución. Básicamente se encarga de ejecutar los interceptores definidos antes y después de la ejecución del Action, así como gestionar la generación del resultado de la petición.
- Interceptores: Similar en concepto a los *servlet filter*, se pueden configurar para que ejecuten código antes y/o después de la ejecución del Action misma. También tienen la capacidad de prevenir que un Action sea ejecutado. Por lo general los interceptores encapsulan funcionalidad común entre un conjunto de Actions.
- `Action`: Un Action es el objeto que ejecuta una petición HTTP, donde normalmente el desarrollador provee la lógica de presentación de la aplicación. Puede ser cualquier POJO que tenga un método `String execute()`, que implemente la interface `Action`, o bien puede ser un objeto que extienda la clase `ActionSupport`, la cual provee una implementación por defecto de las funcionalidades más comunes. Cada Action y el resultado que genera se configura en el archivo `struts.xml` o bien mediante anotaciones. El resultado de la ejecución del Action es un String que identifica el resultado que debe ser generado.
- `Result`: Interface general del sistema que implementa un tipo de resultado emitido por

un Action. La clase base de todos los tipos de resultados es `StrutsResultSupport`, y existen diferentes subclases que generan una respuesta de diferente tipo, por ej. un archivo de texto plano, el resultado de procesar un archivo programado en FreeMarker[1], delegar la generación de la respuesta hacia una página JSP, hacer un *redirect* HTTP hacia una URL, generar un archivo PDF, etc.

- `PreResultListener`: Es una interface que puede implementar un objeto, a efectos de ser registrado al `ActionInvocation` para que sea llamado (mediante un *callback*) después de la ejecución del Action, pero antes de que sea ejecutado el Result.
- *Templates*: Son los resultados que serán emitidos en respuesta al procesamiento de la petición HTTP, y son creados por el usuario, por lo general son páginas JSP o *templates* de FreeMarker. Desde estos *templates* se puede hacer referencia a la biblioteca de etiquetas (*taglibs*) de Struts 2 para facilitar la creación de formularios y diseños gráficos.
- Subsistema de etiquetas (*Tag Subsystem*): Es una *taglib* que facilita al desarrollador la creación de páginas HTML, y provee funciones tanto para generar formularios y sus elementos (campos de texto, botones, listas, textarea, etc.) como funciones lógicas (if-then-else, iteradores sobre colecciones, etc.). El código HTML resultante para los componentes de los formularios se generan mediante una serie de *templates* en FreeMarker, lo cual permite al desarrollador redefinirlos para generar un diseño gráfico o comportamientos acorde a sus necesidades.
- `ActionMapper`: Es el componente de Struts 2 encargado de tomar una petición HTTP y resolver a cual Action se hace referencia en la misma. Struts 2 se puede configurar para que acepte peticiones de acciones con URLs de la forma `miAction.action` o `myAction.do` entre otros, también se puede configurar para que acepte URLs con estilo REST.[2] Las etiquetas de Struts 2 utilizadas para generar las respuestas también hacen uso del `ActionMapper` cuando necesitan generar un String con la URL de algún recurso.

6.1.2. Actions

Un Action (en castellano, "acción") es un objeto que ejecuta una petición HTTP, es decir es la porción de código que realiza el trabajo que está solicitando el usuario, y asiste al framework en decidir qué contenido debe generarse en respuesta a la petición del usuario.

Como se explicó anteriormente, un Action debe implementar un método `execute()` que retorne un String, y si bien la clase puede ser cualquier POJO que implemente este método, normalmente se crea una subclase de `ActionSupport` ya que provee ya implementadas funcionalidades típicas como:

- Poder agregar una lista de mensajes de error, o bien un mensaje de error para un campo del formulario en particular (con soporte de internacionalización de los mismos).
- Poder agregar mensajes informativos (con soporte de internacionalización de los mismos).
- Validación de los parámetros de la petición HTTP, sin ejecutar el método `execute()` en caso de que alguna validación no sea aceptada.

El Listado 6.1 muestra una implementación típica de una clase Action. La clase `CancelarReservaAction` define en la línea 2 una variable de instancia `bookingId` con sus correspondientes *setter* y *getter* que representa un parámetro que necesita ser proporcionado en la petición HTTP. Struts 2 se encarga de convertir el parámetro HTTP en un objeto de tipo `Long` y luego asignarlo a la variable `bookingId` dentro de la instancia del Action creada para ejecutar la petición.

Una vez establecidos los parámetros de la petición HTTP en la instancia del Action, Struts 2 ejecuta el método `validate()` (líneas 4 a 9) cuyo propósito es realizar validaciones antes de la ejecución de la lógica asociada al Action. En este caso, el `bookingId` representa la clave primaria

[1] FreeMarker [17] es un lenguaje para generar texto a partir de *templates*, similar a las páginas JSP.

[2] REST (de las siglas *Reprensentational State Transfer* [41]) es un estilo de arquitectura de aplicaciones web, que intenta representar cada Action como un recurso Web, y la red de páginas Web que navega el usuario como una máquina de estados, donde el usuario progresa a través de la aplicación seleccionando hipervínculos (transiciones de estado) resultando en la próxima página mostrada (representando el próximo estado de la aplicación). Además de usar las peticiones `GET` y `POST` de HTTP, también utiliza `PUT` y `DELETE`.

```
1  public class CancelarReservaAction extends ActionSupport {
2    private Long bookingId;
3
4    @Override
5    public void validate() {
6      if (bookingId == null || bookingId <= 0) {
7        addActionError("El identificador de Reserva es invalido");
8      }
9    }
10
11   @Override
12   public String execute() throws Exception {
13     final BookingService bookingService = new BookingService();
14     final Booking booking = bookingService.getBookingById(bookingId);
15     if (booking == null) {
16       addActionError("La reserva con id " + bookingId + " no existe!");
17       return INPUT;
18     }
19
20     bookingService.cancel(booking);
21     addActionMessage("La Reserva id " + booking.getId() + " en el Hotel <b>"
22       + booking.getHotel().getName() + "</b> ha sido eliminada!");
23     return SUCCESS;
24   }
25
26   public Long getBookingId() {
27     return bookingId;
28   }
29
30   public void setBookingId(Long bookingId) {
31     this.bookingId = bookingId;
32   }
33 }
```

Listado 6.1: Ejemplo de una clase Action en Struts 2

de una tupla dentro de una tabla, con lo cual se valida que el parámetro haya sido especificado (distinto de `null`) y que sea un número positivo; en caso contrario, se genera un mensaje de error (línea 7).

Luego de llamar al método `validate()`, Struts 2 chequea si se ha agregado algún mensaje de error. Si lo hay, Struts 2 automáticamente retorna el resultado `INPUT` como respuesta a la ejecución del Action para luego renderizar la respuesta asociada al resultado `INPUT` (que se define dentro del archivo `struts.xml`).

Si no hay ningún mensaje de error, se continúa con la ejecución del método `execute()`, procesando así la petición sabiendo que las validaciones básicas han sido aceptadas. En el ejemplo, se crea un objeto que busca la reserva (objeto `booking` en línea 14). Si no la encuentra genera un mensaje de error y retorna el resultado `INPUT`. Si la encuentra, en la línea 20 se llama al método `cancel()`, encargado de realizar el `DELETE SQL`, para luego agregar un mensaje informando el éxito de la operación (líneas 21 y 22) y retornar el resultado `SUCCESS`, que indica a Struts 2 la página que debe renderizarse en respuesta a ese resultado.

6.1.3. Archivo `struts.xml`

El archivo `struts.xml` es el archivo de configuración principal de Struts 2. Se encarga de definir los Actions soportados por la aplicación y qué respuesta renderizar en base a los resultados emitidos por los Actions, entre otros elementos. En el Listado 6.2 se puede ver un típico ejemplo de un archivo de configuración `struts.xml`.

La línea 2 del listado indica que la estructura del archivo `struts.xml` esta gobernada por un DTD (*Document Type Definition*). En las líneas 4 a 7 se declaran algunas de las constantes que indican al framework el comportamiento por defecto para algunos de los muchos elementos configurables de Struts 2:

- `struts.devMode`: Cuando esta en `true` hace que Struts 2 re-cargue y re-procese el archivo `struts.xml` automáticamente cuando éste se cambia (es útil cuando se está desarrollando).

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"
    "http://struts.apache.org/dtds/struts-2.3.dtd" >
<struts>
  <constant name="struts.devMode" value="false" />
  <constant name="struts.locale" value="es_AR" />
  <constant name="struts.i18n.encoding" value="ISO-8859-1" />
  <constant name="struts.objectFactory" value="spring" />

  <package name="public" extends="struts-default">
    <interceptors>
      <interceptor-stack name="publicStack">
        <interceptor-ref name="defaultStack">
          <param name="exception.logEnabled">true</param>
          <param name="exception.logLevel">ERROR</param>
          <param name="exception.logCategory">com.miempresa</param>
        </interceptor-ref>
      </interceptor-stack>
    </interceptors>

    <default-interceptor-ref name="publicStack" />

    <default-action-ref name="index" />

    <global-results>
      <result name="sqlexception">WEB-INF/jsp/errorSQLException.jsp</result>
      <result name="exception">WEB-INF/jsp/exception.jsp</result>
    </global-results>

    <global-exception-mappings>
      <exception-mapping exception="java.sql.SQLException"
        result="sqlexception" />
      <exception-mapping exception="org.springframework.dao.DataAccessException"
        result="sqlexception" />
      <exception-mapping exception="java.lang.Exception"
        result="exception" />
    </global-exception-mappings>

    <action name="index">
      <result name="success">WEB-INF/jsp/capacitacion.jsp</result>
    </action>

    <action name="buscarPorArea" class="com.miempresa.web.action.BuscarPorAreaAction">
      <result name="success">WEB-INF/jsp/capacitacionPorArea${nombrePaginaSegunTipoEvento
          }.jsp</result>
      <result name="input">WEB-INF/jsp/capacitacion.jsp</result>
    </action>

    <action name="reunioncientifica">
      <result name="success">WEB-INF/jsp/reunioncientifica.jsp</result>
    </action>

    <action name="detallereunioncientifica" class="com.miempresa.web.action.
        DetallereunioncientificaAction">
      <result name="success">WEB-INF/jsp/detallereunioncientifica.jsp</result>
      <result name="success_ciclo">WEB-INF/jsp/detallereunioncientifica_ciclo.jsp</result
          >
      <result name="input">WEB-INF/jsp/reunioncientifica.jsp</result>
    </action>

  <action name="bajarMaterial" class="com.miempresa.web.action.BajarMaterialAction">
    <result name="success" type="stream">
      <param name="contentType">${material.mimeType}</param>
      <param name="inputName">materialStream</param>
      <param name="contentDisposition">attachment; filename=${material.filename}</param>
      <param name="contentLength">${material.length}</param>
      <param name="bufferSize">1024</param>
    </result>
    <result name="input">WEB-INF/jsp/exception.jsp</result>
  </action>
  </package>
</struts>
```

Listado 6.2: Ejemplo de archivo `struts.xml`

- `struts.locale`: Se configura el *locale* (es decir el idioma a usar) por defecto para la aplicación. En el ejemplo se define `es_AR` (idioma español, adaptado a Argentina).
- `struts.i18n.encoding`: Se configura el esquema de codificación por defecto de los caracteres alfanuméricos transmitidos por la aplicación.
- `struts.objectFactory`: Especifica la forma de crear objetos inyectados en la aplicación. Struts 2 viene integrado con un framework de *dependency injection* propietario, pero puede especificarse otro; en particular el valor `spring` indica que desea usarse el framework Spring para inyectar las dependencias.

En la línea 9 se define un `package` (paquete), que representa una unidad lógica de configuración, una agrupación de Actions, interceptores, y pilas de interceptores y otros elementos. Los paquetes pueden ser extendidos mediante un mecanismo de herencia (similar a las clases de los lenguajes orientados a objetos). El paquete debe tener un nombre (en este caso se llama `public`) y el atributo `extends` indica que hereda del paquete llamado `struts-default` (que ya viene pre-configurado en Struts 2), o sea que hereda todas las definiciones realizadas en ese paquete.

En las líneas 10 a 18 se ve el elemento `interceptor-stack`, que define una nueva "pila de interceptores". Una pila de interceptores indica el orden en que se van llamando los interceptores definidos por la aplicación (en este caso, no se definió ninguno nuevo). Aquí se define una nueva pila llamada `publicStack`, que en realidad es una copia de la pila llamada `defaultStack` (definida en el paquete heredado `struts-default`) con el agregado de unos parámetros de configuración del interceptor llamado `exception` (líneas 13 a 15).

La línea 20 indica que la pila de interceptores por defecto que se aplicarán a los Actions es la llamada `publicStack`. La línea 22 define que cuando no se especifique ningun Action en la URL, se ejecutará el Action llamado `index`.

Las líneas 24 a 27 definen resultados globales con el elemento `<global-results>`, lo que significa que si algún Action o interceptor retorna el resultado especificado con el atributo `name`, se renderizará la respuesta especificada como cuerpo del elemento. En el ejemplo, si el resultado de la ejecución es `sqlexception`, se renderizará la página JSP `errorSQLException.jsp`, y si es `exception` se responderá la página `exception.jsp`.

En las líneas 29 a 36 se especifican mapeos globales de excepción usando el elemento `<global-exception-mappings>`, que definen un resultado en particular en caso de que se detecte la presencia de una excepción no manejada por el Action (o bien por ningún interceptor de la pila). Aquí cada vez que se produzca una `SQLException` o `DataAccessException` se retornará el resultado `sqlexception`, mientras que si se detecta una `Exception` se retornará el resultado `exception`. Esto funciona en tandem con lo definido en los `<global-results>`.

Las líneas restantes (a partir de la 38) definen los Actions que es capaz de ejecutar esta aplicación, usando la etiqueta `<action>`. El atributo `name` es el nombre con que se hará referencia al Action desde una petición HTTP. Dentro de la etiqueta del Action se definen los resultados posibles que pueda arrojar la ejecución del mismo (junto con los interceptores). Por ej. el Action `index` renderizará la página `capacitacion.jsp` si el resultado de su ejecución es `success`. Este Action en particular no define el atributo `class` que indica la clase que se ejecutará, lo que supone que se ejecutará una clase por defecto (esta clase es `ActionSupport`, heredada del paquete `struts-default`, cuyo método `execute()` retorna `success`).

El Action `buscarPorArea`, que es implementado por la clase `BuscarPorAreaAction` (definida por el desarrollador) tiene dos posibles resultados: `success`, cuya página a renderizar depende del valor a variable de instancia `nombrePaginaSegunTipoEvento` definida en el Action: por ej. si su valor es el string "`Congreso`", entonces la página a mostrar es `capacitacionPorAreaCongreso.jsp`. El resultado `input` hace que vuelva a la página `capacitacion.jsp`.

El Action `detallereunioncientifica` tiene tres resultados posibles, cada uno renderizando una respuesta apropiada a cada caso.

Por último, el Action `bajarMaterial` tiene dos resultados posibles: cuando el resultado es `success` retorna el contenido de un archivo utilizando el tipo de resultado `stream`, y cuando el resultado es `input` se muestra una página JSP.

Como nota final, es posible realizar la configuración de Struts 2 usando anotaciones. Esto es, Struts 2 provee anotaciones que permiten configurar las Actions, interceptores, resultados, paquetes y otros elementos, embebiendo las mismas en el código fuente. Sin embargo, al momento en que se escribió este documento, las anotaciones no son tan poderosas como las configuraciones que se pueden realizar en el archivo `struts.xml`.

6.1.4. El `ValueStack`

El `ValueStack` (o pila de valores) es una estructura de datos que mantiene una pila de objetos. Cuando Struts 2 desea evaluar una expresión utilizando OGNL, se busca en la pila hacia abajo, comenzando desde el último objeto insertado, hasta llegar al primer objeto insertado en la pila. Lo que se busca es un objeto con un *getter* o *setter* de una propiedad especificada o bien un método con ese nombre (dependiendo de la expresión siendo evaluada).

Cada vez que se ejecuta un Action se crea una nueva pila, donde los interceptores pueden ir insertando datos que se utilizarán más adelante en la ejecución de la petición HTTP. De particular importancia es que antes de renderizar la respuesta, Struts 2 inserta en la pila *el Action mismo*, de modo que el objeto que ha atendido la petición HTTP queda en el tope de la pila. Luego, desde la página que renderiza la respuesta pueden llamarse métodos (típicamente *getters* de variables de instancia del Action) que retornan la información a mostrar en la página.

Por último, la pila está disponible en todo momento durante la ejecución de una petición, inclusive cuando se decide cómo generar la respuesta, pudiendo evaluar expresiones OGNL dentro del archivo `struts.xml`, como en el Listado 6.2, líneas 58 a 64, donde se utilizan expresiones OGNL que obtienen datos del `material`, que es una variable de instancia del Action (con su respectivo *getter*).

6.2. La Solución Propuesta para Soporte de Conversaciones

En las siguientes Secciones hasta llegar al final del Capítulo se describirá la solución propuesta para soportar el concepto de conversaciones utilizando el framework Struts 2. Se comunicarán los Principios de Diseño que se han seguido para definir la solución, se enumerarán y especificarán en detalle las funcionalidades implementadas y se darán instrucciones para compilar y construir los componentes a efectos de poder usarse en otros proyectos.

6.2.1. Los Principios de Diseño

Los siguientes son los principios de diseño que se han usado como guía para implementar la solución de proveer soporte de conversaciones a Struts 2. Esto es, los pilares fundamentales que han dirigido la toma de decisiones cuando más de una opción de diseño puede encajar en la solución a un problema.

Simplicidad

Tanto el uso de la solución como la implementación de la misma deben ser simples, es decir que debe tener una curva de aprendizaje muy corta para poder usarla.

La implementación de conversaciones propuesta, además de ser funcional (aunque se trate de un prototipo tomado como base para una implementación más sofisticada) debe ser simple de aprender, de manera de no insumir una gran curva de aprendizaje para alguien que quiere comenzar a utilizar la funcionalidad.

Estilo

La solución debe respetar en lo posible el estilo de programación provisto por Struts 2 sin usar conversaciones, es decir que no debe introducir artefactos que no sean conocidos por alguien que ya conoce cómo desarrollar con el framework. De esta manera se intenta que las conversaciones se incorporen a la caja de herramientas del desarrollador de una manera natural.

6.2.2. Resumen de las capacidades básicas implementadas

En esta Sección se enumerarán las capacidades básicas implementadas en la solución de soporte de conversaciones. Algunas de ellas coinciden con las capacidades proporcionadas por JBoss Seam o SWF, mientras que otras de las capacidades ofrecidas por estos frameworks no han sido implementadas para simplificar las funcionalidades ofrecidas.

- Conversaciones con claves sintéticas: Se genera y asigna una clave única a cada conversación.
- Conversaciones con claves naturales: Soportan el uso de un String especificado por el desarrollador para identificar la conversación.
- Almacenamiento en sesión de usuario: Las conversaciones se almacenan en una tabla de *hashing* en la sesion de usuario.
- Soporte de *Bijection*: Se agregó soporte para realizar *injection* y *outjection* de objetos dentro de los Actions. Los *scopes* contemplados para esta funcionalidad son *Request*, *Conversation*, *Session*, *Application* y *Cookie*.
- Control comportamiento de conversaciones: Permite definir cómo se ejecutará el Action ante la presencia o ausencia de una conversación.
- Implementación transparente del patrón Post-Redirect-Get: Permite que las conversaciones como los mensajes emitidos por el Action sobrevivan un *redirect* HTTP.
- Políticas de expiración: Se proveen implementaciones de las políticas de expiración por tiempo fijo, en primer y segundo plano, y por vista. El diseño es extensible para que se pueda experimentar con otras políticas.
- Integración con mapeos objeto-relacional: Provee una implementación de conversaciones usando contextos de persistencia extendidos sobre JPA y Spring. El diseño es extensible para poder utilizar otras tecnologías de mapeos objeto-relacional o de gestión de transacciones.
- Tipos de resultados mejorados: Se mejoran los tipos de resultados *dispatcher*, *redirect*, *redirectAction* y *tiles* (estándar de Struts 2) con otros similares con propagación automática de conversaciones.
- Biblioteca de etiquetas: Se provee una *taglib* con etiquetas convenientes para simplificar el uso de conversaciones al momento de renderizar la página de respuesta.
- Tema `simpleconv`: Una extensión del tema `simple` preparado para renderizar formularios que propaguen la conversación corriente en forma transparente.
- Una pila de interceptores pre-configurada: Se provee un *interceptor stack* ya configurado para proveer soporte de conversaciones a una aplicación con las opciones más comunes.

En las próximas secciones se describirán cada una de las funcionalidades de la implementación propuesta siguiendo la división de temas sugerida por el Capítulo 3.

6.2.3. Identificación de conversaciones

El identificador de las conversaciones, tanto para las sintéticas como las naturales utiliza el parámetro HTTP `conversationId`. Para el caso de las claves sintéticas, se utiliza un contador el cual va otorgando un número secuencialmente a medida que se van creando nuevas conversaciones, mientras que para el caso de las claves naturales, las mismas las selecciona el desarrollador.

A diferencia de Seam o SWF, no hay ningún archivo especial donde se declaren las conversaciones, sino que se declaran en el código mismo de la clase Action, con la anotación *@Begin*. Esta anotación se coloca en el método `execute()` que ejecutará el Action al ser llamado por una petición HTTP, y cuando se especifica sin el parámetro `naturalIdExpression`, significa que se creará una conversación sintética. Caso contrario, especificando una `naturalIdExpression` indica que debe crearse una conversación natural cuya clave será el objeto resultante de evaluar la expresión OGNL declarada en el parámetro.

En el Listado 6.3 se puede ver que el método `executeSintetica()` crea una nueva conversación sintética, mientras que `executeNatural()` crea una nueva conversación natural, cuya clave será el primer nombre de la persona, valor que proviene de la evaluación de la expresión OGNL `person.firstname`.

```
public class MiAction extends ActionSupport {
  private Person person;

  @Begin
  public String executeSintetica() {
    return SUCCESS;
  }

  @Begin(naturalIdExpression="person.firstname")
  public String executeNatural() {
    return SUCCESS;
  }

  public Person getPerson() {
    return person;
  }
  public void setPerson(Person person) {
    this.person = person;
  }
}
```

Listado 6.3: Declarando una conversación sintética y natural en Struts 2

```
<sconv:surl action="confirmar" conversationPropagation="join" />
```

Listado 6.4: Propagando una conversación en Struts 2

Desde una página JSP se puede retormar una conversación utilizando la biblioteca de etiquetas (*taglib*) `sconv`. Específicamente, para crear un hipervínculo a un Action propagando la conversación actual, se puede usar la etiqueta `<sconv:url>` mostrada en el Listado 6.4, que acepta todos los mismos parámetros que la etiqueta `<s:url>` de Struts 2, y agrega el parámetro `conversationPropagation` para controlar la propagación de la conversación actual.

6.2.4. Abstracciones para manipular conversaciones

Se han implementado diversos mecanismos para acceder a las conversaciones tanto desde los Actions, las páginas JSP y desde el archivo `struts.xml`, las cuales se describirán a continuación.

Acceso mediante EL

Desde una página JSP puede accederse a los datos de la conversación corriente incorporando al inicio de la misma la etiqueta `<sconv:useConversation/>`, cuyo efecto es almacenar las variables de la conversación en el objeto `pageContext` de la página JSP, de modo que quedan disponibles para su uso utilizando el EL estándar de definido en el estándar JSP. Esto es necesario debido a que el patrón Post-Redirect-Get suele implementarse haciendo un redirect hacia la página JSP directamente (sin pasar por la ejecución de un Action), entonces los datos de la conversación no están disponibles en el `ValueStack`.

Por el contrario, si la página se renderiza en base a que se llama a un Action, éste es colocado en el tope de la pila de valores, y por lo tanto la información almacenada en la conversación (que está en una variable de instancia del Action) es posible accederla desde una expresión OGNL o del EL de las páginas JSP.

En el Listado 6.5 se muestra un ejemplo usando la expresión `${currentHotel.nombre}`: suponiendo que la conversación corriente tiene almacenada la variable `currentHotel` de tipo

```
<%@ taglib uri="/conversation-struts2-tags" prefix="sconv" %>
<sconv:useConversation/>

El hotel reservado es ${currentHotel.nombre}.
```

Listado 6.5: Acceso a la conversación mediante el EL dentro de una página JSP

```
public class MiAction extends ActionSupport {
  @In
  private Person person;
  @In(value="currentUser", scope=ScopeType.SESSION, required=false)
  private User usuarioLogueado;

  private Address address;

  @In
  public void setAddress(Address address) {
    this.address = address;
  }
}
```

Listado 6.6: Declarando una injección con la anotación `@In` en Struts 2

Hotel, entonces la expresión permite imprimir su nombre.

Archivos de flujo de navegación

En Struts 2, el único archivo de flujo de navegación es el `struts.xml`. Desde este archivo no se permite declarar conversaciones, o controlar el inicio o fin de las mismas. Sin embargo, cuando los datos de la conversación son almacenados como variables de instancia del Action, éstas quedan al alcance del archivo `struts.xml` a través del `ValueStack` y las expresiones OGNL.

De los parámetros de control del mecanismo de conversaciones, solo dos permiten ser modificados desde el archivo `struts.xml`, y ambos actúan dentro de un resultado en particular (*i.e.* la etiqueta `<result>`):

- `conversationPropagation`: especifica si el resultado propaga la conversación corriente (valor `join`) o bien no se propaga (valor `none`). Si no se especifica, por defecto se supone `join`.
- `maxInactiveInterval`: especifica el valor del timeout en segundos luego del cual expirará la conversación cuando no haya sido previamente retomada. Este valor toma vigencia a partir de ese momento hasta que otro resultado lo redefina posteriormente. Este mecanismo corresponde a una política de “Expiración selectiva por vista”, Sección 3.6.3.

Dependency bijection

De manera similar al framework JBoss Seam, la implementación de conversaciones desarrollada soporta *depedency injection* y *dependency outjection*, es decir *dependency bijection*, ya que es una forma muy simple de especificar los objetos que se manipularán en el Action desarrollado.

El mecanismo de *dependency bijection* es implementado por la clase `BijectionInterceptor` (que es un interceptor de Struts 2) y ejecuta antes y después de ejecutar el método del Action *per se*. El interceptor tiene capacidad tanto de ejecutar bijections sobre métodos como variables de instancia (aunque estén declaradas `private`).

Anotación `@In` Esta anotación establece una inyección de una valor dentro de una variable, o sobre un método *setter* como se muestra en el Listado 6.6. Es muy similar en comportamiento a la anotación `@In` de JBoss Seam. En el Action de ejemplo se inyecta una variable de instancia llamada `person`, y también se inyecta el `address` mediante una llamada al método `setAddress()`. El comportamiento de la anotación `@In` puede configurarse de varias maneras, pero todas ellas aceptan valores y comportamiento por defecto. Los parámetros son los siguientes:

- `value`: El nombre que debe buscarse en el *scope* apropiado para realizar la inyección. Si no se especifica, se asume por defecto que el nombre de la variable a buscar es igual al nombre de la variable de instancia anotada, o bien si se trata de un método *setter* anotado, es el nombre de la propiedad que representa (por ej. para el método `setAbc()`, se buscará el valor `abc`).

ScopeType	Alcance
REQUEST	El objeto HttpServletRequest de la petición.
CONVERSATION	El objeto Conversation de la petición, *i.e.* la conversación corriente.
SESSION	El objeto HttpSession, esto es la sesión de usuario.
APPLICATION	El objeto ServletContext, *i.e.* dentro del contexto global de la aplicación.
COOKIE	Busca entre los objetos Cookie de la petición, o sea las *cookies* recibidas por el servidor en la petición.

Cuadro 6.1: Valores de la enumeración ScopeType.

- scope: El *scope* donde se buscará el valor a inyectar. Si se especifica debe utilizarse uno de los valores del enum ScopeType mostrados en el Cuadro 6.1. Si no se especifica, se asume por defecto que el valor se hallará en CONVERSATION, es decir la conversación corriente.
- required: Especifica si el valor a ser inyectado debe ser no nulo (valor por defecto) o puede ser null. En caso de que el valor sea especificado como requerido pero sea null se elevará una IllegalArgumentException.
- create: Especifica que el objeto a ser inyectado sea creado si no es encontrado dentro del *scope* (create=true). El objeto a crear debe declarar un constructor con acceso public sin argumentos. Por defecto es false.

Retormando el Listado 6.6, a la variable person se inyectará el valor asociado al String "person" almacenado en la conversación corriente (no pudiendo este ser null). Para el método setAddress() el comportamiento es similar, mientras que en la variable usuarioLogueado se inyectará el valor almacenado en la sesión de usuario con la clave "currentUser", pudiendo este último ser null. Note en el último caso la versatilidad y facilidad de uso de la anotación: Debido a que el valor de la clave puede estar almacenado en la sesión de usuario con otro nombre al de la variable (lo que permite una fácil integración con aplicaciones *legacy*), y el hecho de que no haga falta que el Action implemente la interface SessionAware que inyecta un Map<String, Object> representando la sesión de usuario, pero donde también el programador es el que debe obtener y colocar valores dentro de ese Map, hace que su uso sea muy conveniente.

Por último, en una aplicación web hay veces que se requiere acceder a los objetos de bajo nivel para acceder a ciertos parámetros de configuración o valores especiales. Si bien Struts 2 intenta abstraerse lo máximo posible de la especificación de Servlets (que es la tecnología básica que implementa la aplicación web) a veces hace falta "bajar" de nivel y acceder directamente a los objetos definidos por la API estándar de Servlets. Por ejemplo si la aplicación necesita acceder directamente a la sesión de usuario (como se describió arriba) Struts 2 provee la interface SessionAware, y si se requiere acceder a los atributos del *scope* Request, Struts 2 provee la interface RequestAware, que requiere implementar un método que inyecta un Map<String, Object> con los objetos asociados, lo mismo pasa con la interface ApplicationAware, CookieAware o ParameterAware, las cuales inyectan un Map<String, Object> con los datos almacenados en el contexto de la aplicación, cookies o parámetros de la petición HTTP, respectivamente. La anotación @In permite prescindir de dichas interfaces, ya que se puede declarar (e inyectar) una variable de instancia del objeto de la especificación de Servlet deseada, directamente sin necesidad de implementar una interface. El Listado 6.7 muestra una clase Action que implementa la interface RequestAware versus una que inyecta el mismo objeto de la especificación de Servlets.

Específicamente, los tipos de datos especiales (pertenecientes a la especificación de Servlets) que es capaz de inyectar la anotación @In son los siguientes:

- ServletContext: El objeto representando el contexto de la aplicación (o *application scope*).
- HttpSession: El objeto representando la sesión de usuario actual.
- Conversation: El objeto representando la conversación corriente.[3]
- HttpServletRequest: El objeto representando la petición HTTP corriente.

[3]Note que esto permite tener un control de bajo nivel sobre el comportamiento de la conversación (aunque su manipulación desmedida no sea recomendable).

```
public class MiActionConRequestAware extends ActionSupport implements
    RequestAware {
  private Map<String, Object> requestMap;

  public void setRequest(Map<String,Object> requestMap) {
    this.requestMap = requestMap;
  }

  public String execute() {
    Object dato = requestMap.get("undato");
  }
}

public class MiActionConAnotacionIn extends ActionSupport {
  @In
  private HttpServletRequest request;

  public String execute() {
    Object dato = request.getAttribute("undato");
  }
}
```

Listado 6.7: Inyectando la petición HTTP con interface `RequestAware` versus anotación `@In`

```
public class MiAction extends ActionSupport {
  @Out
  private Person person;
  @Out(value="currentUser", scope=ScopeType.SESSION, required=false)
  private User usuarioLogueado;

  private Address address;

  @Out
  public Address getAddress() {
    return address;
  }
}
```

Listado 6.8: Declarando una *outjection* con la anotación `@Out` en Struts 2

Anotación `@Out` Esta anotación también sigue el mismo diseño que el de JBoss Seam. La anotación `@Out` aplicada sobre una variable de instancia o un método *getter* indica que se desea hacer una *outjection* del valor hacia uno de los *scopes* que maneja la aplicación web.

En el Listado 6.8 se puede apreciar un ejemplo de uso. Cuando termine de ejecutar el método asociado al Action, la variable de instancia `person` se almacenará en el contexto de la conversación corriente, de la misma forma que lo retornado por el método `getAddress()`, mientras que la variable `usuarioLogueado` será almacenada en la sesión de usuario.

La anotación `@Out` puede configurarse para generar un comportamiento personalizado, dependiendo de los parámetros que se le especifiquen, a saber:

- `value`: El nombre que debe utilizarse en el *scope* apropiado para realizar la *outjection*. Si no se especifica, se asume por defecto que el nombre de la variable a salvar es igual al nombre de la variable de instancia anotada, o bien si se trata de un método *getter* anotado, es el nombre de la propiedad que representa (por ej. para el método `getAbc()`, se almacenará con el nombre `abc`).
- `scope`: El *scope* donde se almacenará el valor. Si se especifica debe utilizarse uno de los valores del `enum ScopeType` mostrados en el Cuadro 6.1. Si no se especifica, se asume por defecto que el valor se hallará en `CONVERSATION`, es decir la conversación corriente.
- `required`: Especifica si el valor a ser almacenado debe ser no nulo (`required=true`) o puede ser `null`. En caso de que el valor sea especificado como requerido pero sea `null` se elevará una `IllegalArgumentException`. El valor por defecto es `false` (no requerido).

Biyección y el `BijectorInterceptor` La biyección es realizada por el `BijectorInterceptor` (que debe estar incluido dentro de la pila de interceptores) y se produce cuando una misma

variable es anotada con `@In` y `@Out`, ya sea colocando ambas anotaciones sobre la variable de instancia, o bien colocandolas sobre los correspondientes métodos *setter* y *getter*, respectivamente.

Específicamente, la biyección hace que antes de ejecutar el Action, se inyecten todos los datos anotados con `@In`, luego se ejecute la misma, y previamente a renderizar la página de respuesta se realice la *outjection* de los datos generados por el Action, a efectos de que estén disponibles para la generación de la respuesta.

Note que el `BijectorInterceptor` debe ejecutar *antes* que el `ParametersInterceptor` para que sea posible modificar los datos inyectados por medio de los parámetros de la petición HTTP.

API

Como se explicó en la Sección 6.2.4, la anotación `@In` permite inyectar objetos especiales dentro de los Actions, entre ellos la conversación corriente (que es una instancia de la clase `Conversation`) lo cual hace posible llamar a métodos de esta API para obtener dentro del Action información específica de la conversación. A continuación se listan los métodos disponibles:

- `String getId()`: Retorna el identificador de la conversación.
- `Map<String, Object>getMap()`: Retorna el `Map` con los datos almacenados en la conversación corriente. La clave es un String con el nombre de los datos guardados en forma manual o por el mecanismo de biyección.
- `boolean isNew()`: Retorna `true` si la conversación fue creada en la petición HTTP actual, o `false` si la conversación ha sido retomada.
- `boolean isEnded()`: Retorna `true` si la conversación ha sido terminada en una petición previa, o `false` si no ha sido terminada.
- `end(boolean beforeRedirect)`: Termina la conversación. Si `beforeRedirect` es `true` la conversación termina al finalizar el Action, de lo contrario los datos sobreviven un redirect HTTP.
- `boolean isMarkedForDeletion()`: Retorna `true` si los datos de la conversación serán eliminados al finalizar la petición HTTP actual, o `false` si no serán eliminados.
- `setMaxRequestCountAfterEnded(int maxReqCountAfterEnded)`: Método que configura la cantidad de peticiones HTTP máximas que sobrevivirán los datos de una conversación luego de haber *terminado*. La cantidad por defecto es 1 (sobrevive una sola petición HTTP).
- `int getMaxRequestCountAfterEnded()`: Este método retorna la cantidad máxima de peticiones HTTP que sobrevive una conversación luego de haber *terminado*.
- `long getLastAccessTime()`: Retorna un *timestamp* de la última vez que se accedió a la conversación corriente. El número representa la cantidad de milisegundos transcurridos a partir de Enero de 1970, 00:00:00 horas.
- `setLastAccessTime(long lastAccessTime)`: Actualiza el momento en que se accedió a la conversación por última vez.
- `updateLastAccessTime()`: Actualiza la última vez que se accedió a la conversación con la fecha y hora actual.
- `int getMaxInactiveInterval()`: Retorna el valor del *timeout* de expiración de la conversación, *i.e.* el intervalo de tiempo máximo (en segundos) que se esperará antes de expirar la conversación.
- `setMaxInactiveInterval(int maxInactiveInterval)`: Establece el tiempo máximo (en segundos) que se esperará antes de expirar la conversación.
- `boolean isNaturalId()`: Retorna `true` si la conversación corriente es natural, y retorna `false` si tiene un identificador sintético.

```
public class MiAction extends ActionSupport {
  ...
  @End
  public String confirm() {
    // salva la reserva en la base de datos
    bookingService.saveBooking(reserva);

    DateFormat df = new SimpleDateFormat("dd/MM/yyyy");

    addActionMessage(
        "La reserva en el hotel <b>" + currentHotel.getName()
        + "</b> desde el dia " + df.format(reserva.getCheckinDate()) + " al "
        + df.format(reserva.getCheckoutDate())
        + " ha sido salvada exitosamente.");

    return SUCCESS;
  }

  @End(beforeRedirect=true, commit=false, endResult="cancelado")
  public String cancel() {
    return "cancelado";
  }
}
```

Listado 6.9: Declarando la anotación `@End` para terminar la conversación

6.2.5. Terminación de conversaciones

Tanto en las condiciones de operación normales o cuando ocurre algún evento inesperado es necesario definir el comportamiento del framework al intentar terminar conversaciones, o bien cuando se intenta acceder a una que ya no existe más.

Terminación normal

La terminación de conversaciones se declara utilizando la anotación `@End` sobre un método `execute()` del Action. Básicamente, la anotación `@End` instruye al `ConversationInterceptor` que termine la conversación al finalizar la petición HTTP actual.

El Listado 6.9 muestra un ejemplo de un Action con dos métodos ejecutables. Cuando el método `confirm()` termina su ejecución también se termina la conversación y se hace un commit con los objetos persistentes modificados durante la conversación (si los hubiera), pudiendo acceder a sus datos una vez más luego de hacer un redirect. En el caso del método `cancel()` se está especificando que termine la conversación siempre y cuando el resultado de su ejecución sea `cancelado`, sin hacer un commit de las modificaciones realizadas sobre los objetos persistentes, así como también se está solicitando que los datos de la conversación se destruyan inmediatamente al finalizar la petición HTTP actual.

En detalle, la anotación `@End` permite configurar su comportamiento usando los siguientes parámetros:

- `beforeRedirect`: Especifica si la conversación corriente debe preservar sus datos para ser accedidos luego de realizar un redirect (`false`, valor por defecto), o bien debe destruirla al terminar la petición HTTP actual (`true`).
- `commit`: Se debe especificar `true` si se desea que se realice el commit de la transacción al finalizar la conversación, o `false` si se desea terminarla sin realizar el commit. Por defecto se asume `true`.
- `endResult`: Solo termina la conversación si el resultado del método del Action es el especificado aquí. Si no se especifica, se asume el string `success` (en realidad es la constante `ActionSupport.SUCCESS`).

Comportamiento en presencia de excepciones

Nunca se destruye o termina una conversación si ocurre una excepción durante la ejecución de un Action de la aplicación y ésta no es manejada por la aplicación.

```
public class MiAction extends ActionSupport {
  @Begin
  @ConversationAttribute(ConversationAttributeType.REQUIRED) // redundante:
      @Begin implica REQUIRED
  public String inicio() {
    return SUCCESS;
  }

  @ConversationAttribute(ConversationAttributeType.MANDATORY)
  public String pasoDos() {
    return SUCCESS;
  }

  @End
  @ConversationAttribute(ConversationAttributeType.MANDATORY) // redundante:
      @End implica MANDATORY
  public String final() {
    return SUCCESS;
  }
}
```

Listado 6.10: Declarando las anotaciones `@Begin`, `@End` y `@ConversationAttribute` para especificar los atributos de la conversación

Un caso especial puede ocurrir cuando la excepción ocurre fuera del control de la aplicación del usuario. Por ejemplo, cuando se termina una conversación luego de ejecutar un método anotado con `@End` y se especifica que se realice el commit de los objetos persistentes: en este caso, la operación de commit con la base de datos queda bajo control del `PersistenceTransactionManager` (que es el componente encargado de realizar operaciones de persistencia contra la base de datos) el cual ejecuta *después* de haber terminado la ejecución del método `execute()` del Action. En este caso, la conversación no podrá ser commiteada (debido a que se produjo una excepción mientras se intentaba hacer el commit) pero tampoco se perderá, de modo de poder realizar una (o reintentar otra) funcionalidad que permita efectivizar el commit.

Acceso a conversaciones

Cuando una página o Action es relevante solo en el contexto de una conversación, es necesario contar con un mecanismo que detecte si la misma esta siendo accedida en forma correcta o en una manera anormal, por ejemplo, cuando dicha conversación ya ha sido terminada.

Atributos de conversación Para ello, la implementación provee cinco *atributos de conversación* que controlan el accionar del `ConversationInterceptor` al recibir peticiones HTTP e influye en la ejecución del Action; puede adquirir alguno de los siguientes valores (definidos en el `enum ConversationAttributeType`):

- `REQUIRED`: Se crea una nueva conversación si no existe previamente. Si ya existe se utiliza dicha conversación.
- `REQUIRES_NEW`: Siempre se crea una nueva conversación.
- `MANDATORY`: Se requiere que ya exista una conversación previamente creada.
- `SUPPORTS`: Si existe una conversación previamente creada se utilizará dicha conversación. De lo contrario no se creará una nueva conversación.
- `NONE`: No provee soporte de conversaciones. El Action se ejecutará sin la intervención del `ConversationInterceptor`.

Por ejemplo, es común anotar un método con `@ConversationAttribute(ConversationAttributeType.MANDATORY)` para indicar que la conversación debe existir previamente, y si no existe el interceptor debe retornar un resultado que indique que la conversación que se intenta acceder no existe (sin ejecutar el Action que se requería).

La implementación provee la anotación `@ConversationAttribute` que se aplica al método `execute()` del Action, donde se puede especificar como parámetro el valor del atributo de

conversación deseado para la ejecución de ese método en particular. A pesar de que esta anotación provee un control preciso sobre el atributo de conversación, muchas veces este valor puede ser inferido, o bien puede no tener sentido, dependiendo de la combinación de anotaciones `@Begin`, `@End` y `@ConversationAttribute` aplicadas al mismo método `execute()`. El Listado 6.10 muestra un ejemplo de cómo podrían llegar a presentarse las anotaciones en el código fuente: el método `inicio()` comienza una nueva conversación, pero el programador especifica (en forma innecesaria) el atributo `REQUIRED`; algo similar sucede con el método `final()`. El método `pasoDos()` es un caso típico de un paso intermedio en la ejecución de una funcionalidad donde se requiere que la conversación referenciada ya exista previamente.

El `ConversationInterceptor` chequea si la combinación de estas anotaciones se contradicen unas con otras, y en ese caso se eleva una `IllegalStateException` informando al desarrollador del error cometido. Específicamente el interceptor lleva a cabo las siguientes validaciones sobre los métodos `execute()`:

1. Si la anotación `@Begin` y `@ConversationAttribute(`$atributo$`)` están presentes (`@End` puede estar presente o ausente), la combinación es válida si $atributo =$ `REQUIRED` o bien $atributo =$ `REQUIRES_NEW`.

2. Si la anotación `@Begin` y `@End` están presentes (y `@ConversationAttribute` está ausente) se infiere que el atributo es `REQUIRES_NEW`.

3. Si la anotación `@End` y `@ConversationAttribute` están presentes (y `@Begin` está ausente) solo es válido si el atributo es `MANDATORY`.

4. Si solo la anotación `@Begin` esta presente se infiere que el atributo es `REQUIRED`.

5. Si solo la anotación `@End` esta presente se infiere que el atributo es `MANDATORY`.

6. Si solo la anotación `@ConversationAttribute(`$atributo$`)` esta presente se infiere que el atributo es el indicado por el parámetro $atributo$.

7. Si las tres anotaciones están ausentes, pero en la petición se proporcionó el parámetro `conversationId` con un identificador de conversación que no sea el string vacío, se infiere que el atributo es `SUPPORTS`.

8. En cualquier otro caso, se infiere el atributo `NONE`.

Como se ve arriba, los casos que pueden elevar una excepción son los que incluyen la anotación `@ConversationAttribute` ya que esta *fuerza* el atributo a uno en particular en conjunción con `@Begin` o `@End`. Por ejemplo en el caso 1, se intenta iniciar una conversación con `@Begin` y no tiene sentido aplicar `@ConversationAttribute(ConversationAttributeType.MANDATORY)` al mismo método porque resulta en una situación contradictoria ya que antes de comenzar la nueva conversación no habrá ninguna creada previamente. En el caso 3 ocurre que solo tiene sentido especificar `@End` con `@ConversationAttribute(ConversationAttributeType.MANDATORY)`, ya que si se espera terminar la conversación, también se supone que ya estaba previamente creada, de modo que otro atributo que no sea `MANDATORY` no tiene sentido.

Acceso a conversaciones ya terminadas Cuando una conversación termina, por defecto puede sobrevivir una cantidad predefinida de redirects HTTP (por defecto solo uno, pero puede cambiarse a través del parámetro `maxRequestCountAfterEnded` de la `Conversation`), es decir está terminada, pero no eliminada de la sesión de usuario, permaneciendo allí hasta que se alcance la cantidad máxima de accesos luego de ser creada, o bien expire debido a que no ha sido accedida por un periodo prolongado de tiempo.

Si se intenta acceder a una conversación terminada (o expirada, o inexistente) mediante un método con atributo `MANDATORY`, no se ejecutará el Action llamado sino que se retornará directamente el resultado `conversation_not_found` que podrá ser utilizado para redireccionar la interacción del usuario hacia un lugar específico de la aplicación (o bien una pantalla de error indicando que los datos ya no están disponibles). Este resultado puede ser cambiado mediante un parámetro del `ConversationInterceptor` llamado `conversationNotFoundResult`.

El Listado 6.11 muestra un ejemplo donde se define un resultado global que se muestra cuando la conversación ya no existe, haciendo que se haga un redirect hacia el Action `index`,

```
<struts>
  ...
  <package ...>
    ...
    <global-results>
      <result name="conversation_not_found" type="redirectAction">
        <param name="actionName">index</param>
        <param name="namespace">/</param>
      </result>
    </global-results>
    ...
    <action name="confirmar">
      <result name="success">WEB-INF/jsp/exito.jsp</result>
      <result name="conversation_not_found">WEB-INF/jsp/reservaNoExiste.jsp</
          result>
    </action>
    ...
  </package>
</struts>
```

Listado 6.11: Archivo `struts.xml` declarando un resultado `conversation_not_found`

```
<struts>
  ...
  <constant name="com.hexaid.struts2.conversation.expiration.policy" value="
      perview" />
  ...
</struts>
```

Listado 6.12: Ejemplo de configuración de política de expiración de conversaciones

mientras que en particular cuando se llama al Action `confirmar` y la conversación no existe, se muestra la página `reservaNoExiste.jsp`.

En tanto, si se intenta acceder a una conversación terminada (pero aún guardada en la sesión de usuario), y el atributo de la conversación no es ni `MANDATORY` ni `NONE`, entonces *si* es posible accederla, aunque debe tenerse la precaución de no modificar ni sobreescribir los datos en la misma ya que esto podría complicar innecesariamente la lógica de la aplicación. El caso común de este tipo de acceso ocurre luego de terminar la conversación y hacer un redirect propagando el identificador de conversación, para implementar un patrón Post-Redirect-Get y así poder mostrar mensajes de éxito o error, o bien mostrar el resultado de la funcionalidad recién ejecutada (la cual por supuesto, está almacenada en la conversación recién terminada).

6.2.6. Políticas de expiración de conversaciones

La implementación soporta tres de las políticas de expiración de conversaciones descritas en la Sección 3.6: expiración por tiempo fijo (`fixedtime`), expiración en primer y segundo plano (`foreback`), y expiración selectiva por vista (`perview`).

La política configurada por defecto es la `foreback`, la cual puede cambiarse utilizando una constante llamada `com.hexaid.struts2.conversation.expiration.policy` definida dentro del archivo `struts.xml` de la aplicación, como muestra el ejemplo del Listado 6.12.

Un detalle a tener en cuenta es que el chequeo de conversaciones expiradas se realiza en cada petición HTTP recibida y gestionada por el `ConversationInterceptor`, de modo que puede suceder que una conversación que esté expirada permanezca almacenada en la sesión de usuario hasta que se reciba una petición HTTP y se tenga la oportunidad de procesar las expiraciones y eliminarla.

Extensibilidad de las políticas de expiración

La implementación provee un mecanismo extensible de manejo de políticas de expiración, pudiendo el desarrollador implementar o experimentar con su propia política si así lo desea. Para ello se ha utilizado el mecanismo de inyección de dependencias nativo de Struts 2 para proveer al `ConversationInterceptor` de una implementación de la interface `ConversationExpirationPolicy` (interface que debe implementar la clase que defina la nueva política de expiración).

```
<struts>
  <bean type="com.hexaid.struts2.expiration.ConversationExpirationPolicy"
    name="mipolitica"
    class="com.miempresa.MiPoliticaConversationExpirationPolicy" />
  <constant name="com.hexaid.struts2.conversation.expiration.policy"
    value="mipolitica" />
  ...
</struts>
```

Listado 6.13: Implementación de una nueva política de expiración

```
<struts>
  ...
  <constant name="com.hexaid.struts2.conversation.expiration.
      maxInactiveInterval" value="600" />
  ...
</struts>
```

Listado 6.14: Ejemplo de configuración de la constante `maxInactiveInterval` del timeout de expiración de la conversación

El Listado 6.13 muestra la forma de declarar una nueva política llamada `mipolitica` dentro del archivo `struts.xml`.

`fixedtime`: Expiración por tiempo fijo

Esta es la política más sencilla pero la menos efectiva, ya que una conversación se da por expirada si han transcurrido más de `maxInactiveInterval` segundos que no ha sido accedida. El valor `maxInactiveInterval` es una constante que está establecida por defecto en 5 minutos (300 segundos) pero puede configurarse mediante una constante dentro del archivo `struts.xml` de la aplicación. El Listado 6.14 muestra un ejemplo que la establece a 10 minutos (600 segundos).

Una particularidad del valor `maxInactiveInterval` es que si se establece con un número menor o igual a cero, entonces las conversaciones no expirarán sino hasta que lo haga la sesión de usuario.

`foreback`: Expiración en primer y segundo plano

Esta política es equivalente a la política de expiración con el mismo nombre de JBoss Seam, en que la conversación en primer plano (*i.e.* la conversación corriente) nunca expira (a menos que expire la sesión de usuario) mientras que las otras conversaciones iniciadas en la misma sesión de usuario son las que están en segundo plano y son susceptibles de ser expiradas luego de que el tiempo de acceso supere el definido por la constante `maxInactiveInterval`. El diagrama de estados de la Figura 4.2 refleja el comportamiento de esta política de expiración.

`perview`: Expiración selectiva por vista

Se permite esta política de expiración por medio de la modificación de la variable de instancia `maxInactiveInterval` de la clase `Conversation`, la cual puede realizarse de dos formas:

- Inyectando la conversación corriente dentro del Action y llamando al método `setMaxInactiveInterval(int)` sobre esa conversación, como se muestra en el Listado 6.15.
- Cambiando el `maxInactiveInterval` de la conversación desde la etiqueta `<result>` dentro del archivo `struts.xml` de la aplicación, como se ve en el Listado 6.16. La implementación actual provee cuatro tipos de resultado que soportan la propagación y gestión de conversaciones, y reemplazan a sus homónimos definidos en la distribución estándar de Struts 2, a saber:

  1. `dispatcher`: Realiza un *forward* del procesamiento hacia una vista (por lo general una página JSP) dentro de la misma petición HTTP.

```
public class MiAction extends ActionSupport {
  @In Conversation conversation;

  @ConversationAttribute(ConversationAttributeType.MANDATORY)
  public String execute() {
    conversation.setMaxInactiveInterval(60); // 1 minuto
    return SUCCESS;
  }
}
```

Listado 6.15: Cambiando el timeout de expiración de la conversación en un Action

```
<struts>
  ...
    <action name="confirmar"
      class="com.hexaid.examples.hotel.web.action.ReservaHotelAction">
      <result name="success" type="tiles">
        <param name="location">/protected/reservaHotel_showConfirm.tiles</
            param>
        <param name="maxInactiveInterval">120</param>
      </result>
      <result name="input" type="tiles">
        <param name="location">/protected/reservaHotel_showForm.tiles</param>
        <param name="maxInactiveInterval">600</param>
      </result>
    </action>
  ...
</struts>
```

Listado 6.16: Cambiando el timeout de expiración de la conversación en `struts.xml`

2. `redirect`: Realiza un redirect HTTP hacia una URL cualquiera.
3. `redirectAction`: Realiza un redirect HTTP hacia una URL que invoca otro Action de la aplicación.
4. `tiles`: Realiza un *forward* del procesamiento hacia una vista (por lo general una página JSP) dentro de la misma petición HTTP, pero la vista está implementada utilizando el framework Tiles [3].

6.2.7. Control de concurrencia

La implementación del `ConversationInterceptor` —que es el componente que gestiona el ciclo de vida de las conversaciones— está implementado para acceder a cualquier dato almacenado en la sesión de usuario de forma exclusiva, esto asegura que no haya múltiples peticiones HTTP que intenten acceder a la sesión de usuario en forma concurrente, lo que evita problemas de corrupción de datos. Como las conversaciones están almacenadas en la sesión de usuario, lo anterior implica que las mismas gozan de la misma protección esto es, no se podrá acceder a la misma conversación desde dos peticiones HTTP.

La única excepción a esta regla es si el Action a ejecutar tiene un atributo de conversación `NONE` (ver Sección 6.2.5). En este caso no se requiere el procesamiento de conversaciones y la petición se ejecuta sin ningún control sobre el acceso concurrente a ese Action.

La decisión de diseñar el `ConversationInterceptor` para que todo procesamiento tenga un *gran lock* sobre la sesión de usuario es bastante conservadora pero a la vez sencilla de implementar y efectiva. Se requerirán más pruebas para saber en qué grado afecta a la escalabilidad de la aplicación (en el sentido de cuantas peticiones por segundo se pueden atender).

6.2.8. Integración con Mapeos Objeto-Relacional

La solución está basada en la estrategia de conversaciones usando un contexto de persistencia extendido (como se describe en la Sección 3.8.5), es decir que el contexto de persistencia se extiende durante todo el ciclo de vida de la conversación. Por el momento solo se implementó el

```
<struts>
  ...
    <interceptor-stack name="jpaConversationStack">
      <interceptor-ref name="abstractConversationSupportStack">
        <param name="conversation.persistence">jpa-spring</param>
      </interceptor-ref>
    </interceptor-stack>
  ...
</struts>
```

Listado 6.17: Configurando el `jpa-spring` en `struts.xml`

```
<struts>
  ...
  <bean type="com.hexaid.struts2.persistence.PersistenceTransactionManager"
    name="mi-ptm"
    class="com.miempresa.MiPersistenceTransactionManager" />
  ...
</struts>
```

Listado 6.18: Declarando un `PersistenceTransactionManager`

soporte para utilizar con JPA y Spring, pero una implementación para Hibernate y Spring sería bastante directa y sencilla de implementar.

Cabe aclarar que el uso de frameworks de mapeo objeto-relacional no es obligatorio para utilizar conversaciones y persistir cambios en una base de datos relacional. El desarrollador puede elegir utilizar un acceso a base de datos basado en JDBC y gestionar la persistencia de los objetos almacenados en la conversación de forma manual.

`PersistenceTransactionManager`

El `PersistenceTransactionManager` (gestor de transacciones de persistencia) es el componente que gestiona la persistencia de los objetos hacia y desde la base de datos relacional, es decir es el componente que se encarga de asegurar que siempre se utilice el mismo contexto de persistencia en las distintas peticiones HTTP que hagan referencia a la misma conversación.

En la implementación vienen definidas dos estrategias de `PersistenceTransactionManager` que pueden ser configuradas mediante el parámetro `persistence` del `ConversationInterceptor`:

- `default`: Es el `PersistenceTransactionManager` que se asume por defecto si no se especifica ninguno, y se utiliza cuando no se requiere contextos de persistencia extendidos, o bien no se utilizará ninguna solución de mapeo objeto-relacional (por ej. JDBC).
- `jpa-spring`: Un `PersistenceTransactionManager` que implementa la gestión de persistencia usando contextos de persistencia extendidos cuando la solución de mapeo objeto-relacional utilizada es JPA, y las transacciones son gestionadas por el framework Spring. Para configurar este gestor, se puede declarar como parámetro del interceptor dentro del archivo `struts.xml` de la aplicación (ver Listado 6.17).

Extensibilidad del `PersistenceTransactionManager`

Como se vio más arriba, se proveen dos implementaciones básicas del `PersistenceTransactionManager`, pero pueden ser agregadas fácilmente nuevas implementaciones que soporten otras tecnologías, por ej: Hibernate como solución de mapeo objeto-relacional y Spring como gestor de transacciones, o bien solo Hibernate o solo JPA (donde las transacciones las gestiones un servidor de aplicaciones Java EE certificado, o bien directamente que las transacciones se gestionen por otro medio).

Para implementar un nuevo `PersistenceTransactionManager` es necesario crear una clase que implemente la interface `PersistenceTransactionManager`, o bien extender la clase `PersistenceTransactionManagerAdapter`, y luego declararlo con una etiqueta <bean> en el archivo `struts.xml` de la aplicación, como se muestra en el ejemplo del Listado 6.18, donde

```
<bean id="dataSource"
  class="org.springframework.jdbc.datasource.DriverManagerDataSource">
  <property name="driverClassName" value="org.hsqldb.jdbc.JDBCDriver" />
  <property name="url" value="jdbc:hsqldb:mem:hotel" />
  <property name="username" value="sa" />
  <property name="password" value="" />
</bean>

<!-- Crea un EntityManagerFactory para usar con Hibernate como proveedor de
    JPA y un data source de la conexion a la base de datos a usar -->
<bean id="entityManagerFactory"
  class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean"
      >
  <property name="dataSource" ref="dataSource" />
  <property name="jpaVendorAdapter">
    <bean class="org.springframework.orm.jpa.vendor.
        HibernateJpaVendorAdapter">
      <property name="database" value="HSQL" />
      <property name="showSql" value="true" />
    </bean>
  </property>
</bean>

<bean id="transactionManager"
  class="org.springframework.orm.jpa.JpaTransactionManager">
  <property name="entityManagerFactory" ref="entityManagerFactory"/>
</bean>
```

Listado 6.19: Declaración de un gestor transacciones basado en JPA con Spring

se declara con el nombre `mi-ptm`, el cual luego debe ser utilizado para configurar el `ConversationInterceptor` como muestra el Listado 6.17.

`default`

Básicamente esta estrategia indica que no se hará ninguna gestión de la persistencia o manejo de transacciones, de modo que dicha tarea queda a cargo del desarrollador de la aplicación. Es implementada por la clase `PersistenceTransactionManagerAdapter`.

`jpa-spring`

Este `PersistenceTransactionManager` se encarga de gestionar las conversaciones con contexto de persistencia extendido cuando en la aplicación se está usando JPA como framework de mapeo objeto-relacional y Spring como gestor de transacciones, y está implementado por la clase `JPASpringPersistenceTransactionManager`.

Específicamente, el componente `jpa-spring` depende de que en Spring se declare un `JpaTransactionManager` para gestionar las transacciones que referencien a objetos persistentes de JPA utilizando un `EntityManagerFactory` también configurado y gestionado por Spring. Un ejemplo de esta configuración es el que se muestran en el Listado 6.19. Spring tiene una gestión muy poderosa y a la vez sencilla de transacciones para diversas tecnologías transaccionales.[4]

El efecto resultante de lo anterior es que por cada conversación (y en forma transparente para el programador de la aplicación web) se almacenará un `EntityManager` que estará disponible en la aplicación con la anotación `@PersistenceContext` cada vez que se ejecute una petición HTTP utilizando la misma conversación, hasta que ésta última termine (en cuyo caso se hará el commit o rollback de las operaciones realizadas).

El Listado 6.20 muestra un ejemplo de una clase DAO que realiza operaciones de persistencia sobre los objetos de una aplicación. Note que el código no depende de ningún componente que tenga que ver con la gestión de conversaciones (o sea es independiente de la capa de presentación de la aplicación) ni de Spring (es independiente de si las transacciones las gestiona Spring o un servidor de aplicaciones Java EE), y lo más notable es que *siempre* a la clase

[4] Más información sobre la gestión de transacciones en Spring puede obtenerse en la documentación oficial [34], Capítulo 10.

```
public class BookingDAO {
  @PersistenceContext
  private EntityManager em;

  public void saveBooking(Booking reserva) {
    em.persist(reserva);
  }

  @SuppressWarnings("unchecked")
  public List<Booking> getBookingsByUserId(String username) {
    String sql =
      "from Booking as b join fetch b.hotel where b.user.username = :username
          " +
      "order by b.checkinDate";
    Query query = em.createQuery(sql);
    query.setParameter("username", username);
    return query.getResultList();
  }

  public void remove(final Booking booking) {
    em.remove(booking);
  }

  public Booking getBookingById(final Long bookingId) {
    return em.find(Booking.class, bookingId);
  }
}
```

Listado 6.20: DAO realizando operaciones de persistencia con JPA

BookingDAO le será inyectada la instancia correcta del EntityManager (la que está asociada a la conversación corriente).

Funcionamiento del componente jpa-spring

El ConversationInterceptor es quien comanda la ejecución del PersistenceTransactionManager configurado y llama a sus métodos en momentos clave dentro del ciclo de vida de la conversación. A continuación se enumeran los métodos de la interface PersistenceTransactionManager, junto al comportamiento que implementa el componente jpa-spring:

- init(): Inicializa el PersistenceTransactionManager al momento de arrancar la aplicación y solo se ejecuta una vez. Se obtiene una referencia al JpaTransactionManager y al EntityManagerFactory.
- conversationStarting(Conversation): Es llamado cuando se inicia una nueva conversación. Se obtiene un nuevo EntityManager y se guarda como un dato más de la conversación, además se asocia a la transacción actual gestionada por Spring.
- conversationPaused(Conversation): Es llamado cuando termina la ejecución de la petición HTTP y la conversación es pausada. Desasocia el EntityManager de la transacción gestionada por Spring.
- conversationResumed(Conversation): Es llamado cuando la conversación es retomada. Si no está terminada se asocia el EntityManager a la transacción gestionada por Spring.
- conversationEnding(Conversation, boolean): Es llamado cuando la conversación termina. Se indica si debe hacerse un commit o rollback. Si hay que hacer un commit se asocia el EntityManager a la transacción actual gestionada por Spring. Finalmente se desasocia el EntityManager de la transacción, se elimina de la conversación actual y se cierra.
- exceptionThrown(Conversation, Exception): Es llamado cuando ocurre una excepción durante la ejecución de la petición HTTP. Solo desasocia el EntityManager de la transacción actual.

6.2.9. Construcción del plugin

El plugin de Struts 2 está desarrollado usando Maven [2]. Para construirlo a partir del código fuente alcanza con ubicarse en el mismo directorio donde está ubicado el archivo `pom.xml` y tipear el comando "`mvn clean install`". Al terminar de ejecutar, en la carpeta `target` se podrá encontrar el archivo `struts2-conversations-1.0-SNAPSHOT.jar` el cual debe incluirse como dependencia en la aplicación web donde se desea usar. Asimismo, la dependencia quedará instalada en el repositorio Maven local a la computadora actual.

El código fuente puede consultarse en el Apéndice A, Sección A.1. Si bien no se muestra el código fuente de las pruebas unitarias, las mismas son accesibles desde el CD que acompaña la presente Tesina.

6.3. Resumen

En este Capítulo se hace una presentación del framework Struts 2 y se describen las características implementadas en el *plugin* que ofrece soporte de conversaciones, uno de los objetivos de esta Tesina.

Struts 2 es uno de los frameworks de desarrollo de aplicaciones web más utilizados del mundo. Es un proyecto de código abierto de la Fundación Apache. Está basado en la arquitectura MVC (*Model-View-Controller*), donde el *Controller* es un componente de entrada que recibe las peticiones HTTP, el *Model* está compuesto por las clases Action encargadas de ejecutar la lógica de presentación y las *View* son las páginas renderizadas que se mostrarán en el navegador web del usuario. Los componentes internos que integran el framework se presentan en forma general.

Las aplicaciones constan de clases Action (normalmente son subclases de `ActionSupport`) y las mismas se declaran en el archivo `struts.xml`, que es el encargado de informar a Struts 2 qué Actions se pueden ejecutar, y dependiendo de cual es el resultado de su ejecución, cual vista debe ser renderizada.

El desarrollo del *plugin* que implementa el soporte de conversaciones tienen como principios de diseño la simplicidad de uso, mientras que se intenta mantener el estilo de programación provisto por Struts 2.

Específicamente, el *plugin* soporta conversaciones tanto con claves sintéticas como naturales. Las conversaciones se almacenan en la sesión de usuario. Se implementó el soporte de *dependency bijection* (similar a JBoss Seam), que simplifica la modificación del estado de la aplicación sin recurrir a la API de bajo nivel de Struts 2 o de Servlets.

Posee un control fino de la ejecución de un Action mediante la especificación de los *atributos de conversación*, que modifican el resultado de la ejecución dependiendo de la ausencia o presencia de una conversación.

Se provee una implementación transparente del patrón *Post-Redirect-Get* (PRG) permitiendo almacenar los mensajes del Action en la conversación a efectos de poder accederlos luego de realizar el redirect HTTP. De la misma manera, los tipos de resultado estándar han sido mejorados para que propaguen el contexto de la conversación en forma transparente para el desarrollador.

Se implementaron varias políticas de expiración de conversaciones, y se ofrece una opción de integración con frameworks de mapeo objeto-relacional. Ambas son configurables y extensibles, *i.e.* se pueden realizar nuevas implementaciones y configurarlas en el *plugin* para su uso en aplicaciones.

Se incluye una biblioteca de etiquetas para uso de conversaciones y el tema `simpleconv` para renderizar formularios que propaguen el contexto de la conversación en forma transparente.

Por último se incluye una pila de interceptores pre-configurada con soporte de conversaciones con las opciones más comunes.

Capítulo 7

Aplicación de ejemplo: UNLP Viajes

En este Capítulo se estudiará el desarrollo de una aplicación de ejemplo llamada "UNLP Viajes". Si bien la aplicación es sencilla en complejidad, es adecuada para demostrar la utilización de la implementación de conversaciones para Struts 2 que se desarrolló para la presente Tesina.

UNLP Viajes es una aplicación web cuya funcionalidad principal es permitir a las personas realizar reservas de hoteles. Para ello, el cliente que desee reservar un hotel debe primero estar registrado (y logueado) en el sitio. Específicamente, la aplicación permite:

- Identificarse por medio de un nombre de usuario y contraseña para poder efectuar reservas.
- Buscar hoteles adheridos a UNLP Viajes a efecto de poder realizar reservas.
- Seleccionar un hotel en particular y ver detalles del mismo, como el precio por día de las habitaciones.
- Efectuar reservas sobre los hoteles seleccionados, indicando preferencias de habitación, y efectuando pagos con tarjeta de crédito.
- Visualizar las reservas efectuadas por el usuario logueado al sistema.
- Cancelar reservas efectuadas con anterioridad.

De hecho, dentro de los archivos de las distribuciones del framework JBoss Seam como de Spring Web Flow viene esta misma aplicación de reserva de hoteles como ejemplo de uso del respectivo framework. Es por eso que es muy conveniente implementar la misma aplicación a efectos de luego poder comparar la complejidad de las implementaciones resultantes, y entonces poder evaluar la facilidad de uso del framework seleccionado.

En las siguientes Secciones se describirán las clases del dominio de la aplicación, su arquitectura y se verá en detalle la implementación de las funcionalidades provistas por el sistema.

7.1. Clases del dominio del problema

Son esencialmente tres las clases que modelan el problema de reserva de hoteles (Figura 7.1), y constituyen las clases del dominio del problema, a saber:

- `User`: Representa un usuario del sistema que puede realizar reservas a su nombre. Sus atributos son el nombre, usuario y contraseña para poder operar en el sistema. El usuario con que ingresa al sistema es único dentro de la aplicación.
- `Hotel`: Representa un hotel sobre el cual pueden realizarse reservas. Consta de un identificador único dentro del sistema, el nombre del mismo, su dirección, código postal, ciudad, provincia y país. También tiene asociado un precio único por habitación.
- `Booking`: Es la reserva propiamente dicha y tiene asociada tanto el usuario que la realizó como el hotel sobre el cual está realizada la reserva. Otros datos importantes son la fecha de *check-in* y de *check-out*, y los datos de pago (nombre de la tarjeta, número de la tarjeta de crédito, mes y año de expiración). Otro dato es si la reserva es sobre una habitación

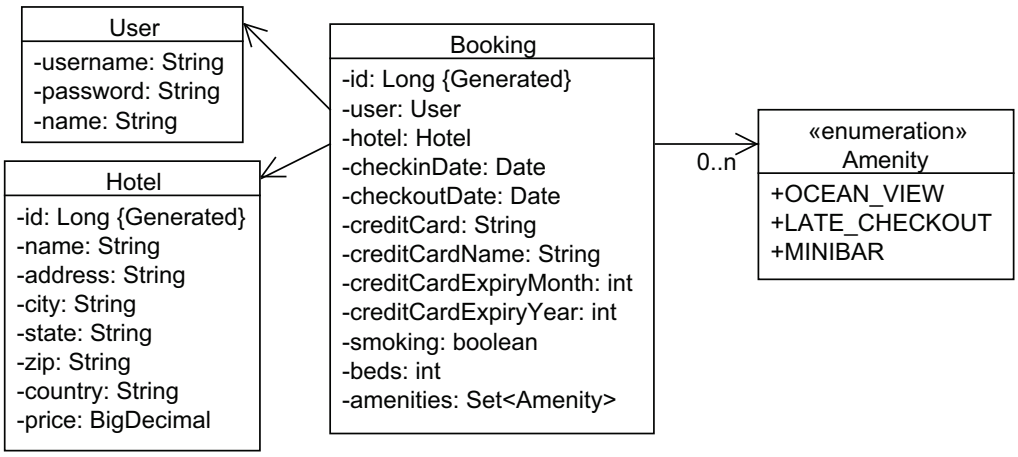

Figura 7.1: Clases del dominio del problema.

para fumador o no fumador, el tipo de cama que debe tener la habitación, y una lista de comodidades (*amenities*) varias que pueden ser seleccionadas (vista al mar, minibar, y/o si se desea realizar un *late check-out*).

7.2. Arquitectura de UNLP Viajes

El sistema implementa una arquitectura en tres capas:

- Presentación: Se encarga de la lógica de navegación de la aplicación, y es la parte que utiliza la implementación de conversaciones con Struts 2. LLama a la capa de servicios para llevar a cabo las funcionalidades del sistema.

- Servicios: Son clases que implementan la lógica y reglas del negocio, y son POJOs configurados como *beans* en Spring. Estos objetos de servicio son llamados por la lógica de presentación. A su vez, los servicios llaman a la capa de acceso a datos para acceder a la base de datos relacional.

- Datos: Son las clases que implementan los detalles del acceso a la base de datos relacional, e implementan el patrón DAO (*Data Access Object*). Estas clases son declaradas como *beans* en Spring, y son llamadas por la capa de servicios. Los DAOs a su vez llaman a los métodos de gestión de persistencia provistos por la API de JPA, que es la tecnología de persistencia elegida para la aplicación.

El sistema no requiere un servidor de aplicaciones Java EE, sino que funciona en un contenedor de aplicaciones web que soporte la especificación de Servlet 2.5 y JSP 2.1. La aplicación fue desarrollada y probada usando Apache Tomcat 6.0.x. A continuación se detallan la selección de frameworks y tecnologías que componen la aplicación:

- Vistas: Las vistas generadas por el sistema están realizadas con páginas JSP. A efectos de componer las páginas en secciones predefinidas (un encabezado arriba, una foto a la izquierda, un pie de página abajo y el cuerpo al centro) se utilizó Apache Tiles 2.2.2. Las tablas se renderizan utilizando la biblioteca de etiquetas displaytag 1.2.

- Presentación: Para la lógica de presentación se usa Struts 2 versión 2.2.3.1 y por supuesto la implementación de conversaciones.

- Seguridad: La autenticación y autorización de usuarios es gestionada por Spring Security 3.0.7.RELEASE, que protege las áreas sensibles de la aplicación solicitando el usuario y contraseña al intentar accederlas.

Figura 7.2: Página principal de la aplicación.

- Base de datos: La base de datos relacional seleccionada fue HSQLDB 2.2.8. Puesto que es una aplicación de ejemplo, se simplifica enormemente el despliegue y ejecución de la aplicación ya que la base de datos está almacenada en memoria, y arranca cuando inicia la aplicación. La base de datos es pre-cargada con hoteles y usuarios de ejemplo en ese mismo momento también.

- Mapeo objeto-relacional: Se seleccionó JPA 2.0 ya que es un estándar, e Hibernate 3.6.8 como la implementación de JPA.

- Infraestructura: Se seleccionó el framework Spring 3.0.6.RELEASE como base para la gestión de la infraestructura de la aplicación. Se hace uso del mecanismo de inyección de dependencias para configurar los componentes de la lógica de la aplicación. También se hace uso de la gestión de transacciones y la configuración de la base de datos en memoria, JPA e Hibernate. Por último, Spring Security utiliza Spring para su propia configuración.

Además, el sistema de desarrolló utilizando Eclipse como entorno de desarrollo integrado, y como herramienta de construcción de utilizó Apache Maven.

7.3. Página principal

Cuando se referencia la aplicación utilizando la url `http://localhost:8080/hotel`, se llama por defecto al `index.action`, cuya única funcionalidad es mostrar la página principal de la aplicación, que se muestra en la Figura 7.2. Desde allí se puede entrar al sistema de dos formas: Haciendo clic en el texto central en color rojo se accede a la búsqueda de hoteles, mientras que haciendo clic en el hipervínculo “Ingresar” (arriba a la derecha) se accede a la página de login que permite ingresar al sistema.

Figura 7.3: Búsqueda de hoteles.

7.4. Búsqueda de hoteles

La aplicación permite buscar hoteles ingresando un texto (en el campo “Criterio”) que coincida en todo o en parte con su nombre, dirección, u otros de los datos asociados a los hoteles. En el ejemplo de la Figura 7.3 se buscaron los hoteles que contengan el String “car” en alguno de sus atributos, resultando en los tres hoteles mostrados en la tabla. La búsqueda puede realizarse en forma anónima (*i.e.* sin necesidad de loguearse al sistema).

En la tabla de resultados, la única acción disponible es hacer clic sobre el hipervínculo “Ver”, que muestra información detallada del hotel. Aquí el usuario puede abrir distintas pestañas o ventanas, seleccionando la acción en cada uno de los hoteles.

7.5. Visualización de detalles del hotel

La ejecución del Action de la vista de los detalles del hotel, donde se incluye (entre otros datos) el precio de sus habitaciones muestra la página de la Figura 7.4. Note que por una decisión de diseño, el Action que obtiene los datos del detalle del hotel no crea una nueva conversación[1], *i.e.* los datos son almacenados en el contexto de la petición HTTP.

Desde esta pantalla, se puede presionar el botón “Reservar”, que inicia el proceso de reserva de una habitación en ese hotel, y por lo tanto requiere que el usuario esté identificado dentro del sistema. Si aún no se ha logueado el usuario en cuestión, se le presentará la pantalla de login, como se muestra en la Figura 7.5.

[1] Esta decisión es idéntica a la implementación del ejemplo de reserva de hoteles de Spring Web Flow, mientras que la implementación de JBoss Seam *si* crea una nueva conversación a partir de la visualización de los detalles del hotel.

Figura 7.4: Visualización de detalles del hotel.

Figura 7.5: Ingreso al sistema de reserva de hoteles.

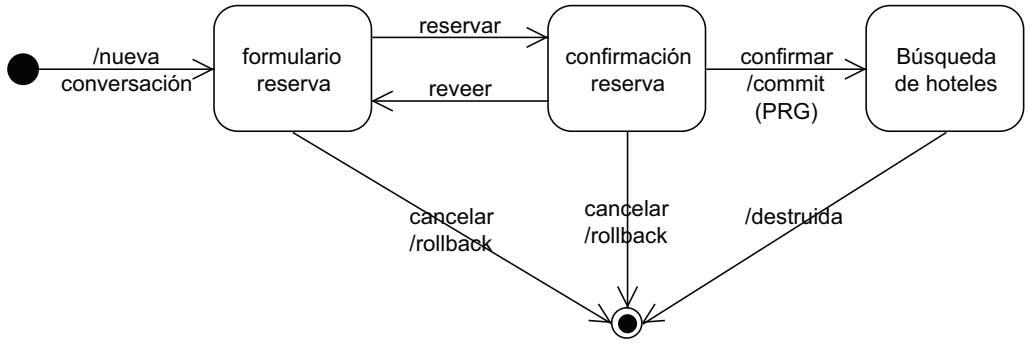

Figura 7.6: Secuencia de pantallas que abarca la conversación.

7.6. Reserva de hoteles

Esta es la funcionalidad central de la aplicación, donde se crea una conversación que contiene la información de la reserva ingresada por el usuario, y al finalizar se sincroniza el estado de los objetos persistentes con la base de datos, por medio del commit de los cambios realizados al contexto de persistencia.

La Figura 7.6 muestra un diagrama donde se puede ver el alcance de la conversación utilizada para gestionar la reserva.

7.6.1. Formulario de carga de reserva

Primero se muestra el formulario donde el usuario entra los datos de la reserva (Figura 7.7) junto con los datos de pago con tarjeta de crédito. En esa pantalla puede realizar una de dos acciones posibles:

- Puede presionar el botón “Cancelar” para abandonar la reserva y terminar inmediatamente la conversación: se retornará a la pantalla de búsqueda de hoteles de la Figura 7.3.
- Puede presionar el botón “Reservar” para continuar con la reserva, lo que produce el efecto de mostrar la página de confirmación de la misma.

7.6.2. Confirmación de reserva

En esta pantalla (Figura 7.8) se muestra un resumen de los datos ingresados en el formulario de reserva para que el usuario tenga la oportunidad de revisarlos antes de efectivizar la misma en el sistema, teniendo a disposición tres acciones posibles:

- Botón “Reservar”: Se procede a efectivizar la reserva y salvarla en forma permanente en la base de datos.
- Botón “Reveer”: Permite volver a la página anterior (Figura 7.7) para modificar algún dato introducido en el formulario.
- Botón “Cancelar”: Abandona la reserva en curso y termina inmediatamente la conversación, retornando a la pantalla de búsqueda de hoteles (Figura 7.3).

7.6.3. Salvar reserva

Al presionar el botón “Reservar” en la pantalla de la Figura 7.8, la aplicación salva el objeto que representa la reserva (una instancia de la clase `Booking`) mediante la terminación de la conversación actual. La reserva exitosa es confirmada por la página de búsqueda de la Figura 7.9, la cual muestra una leyenda en color verde con los datos de la reserva, así como una tabla titulada “Reservas de hoteles realizadas” donde muestra todas las reservas realizadas por el usuario logueado actualmente al sistema.

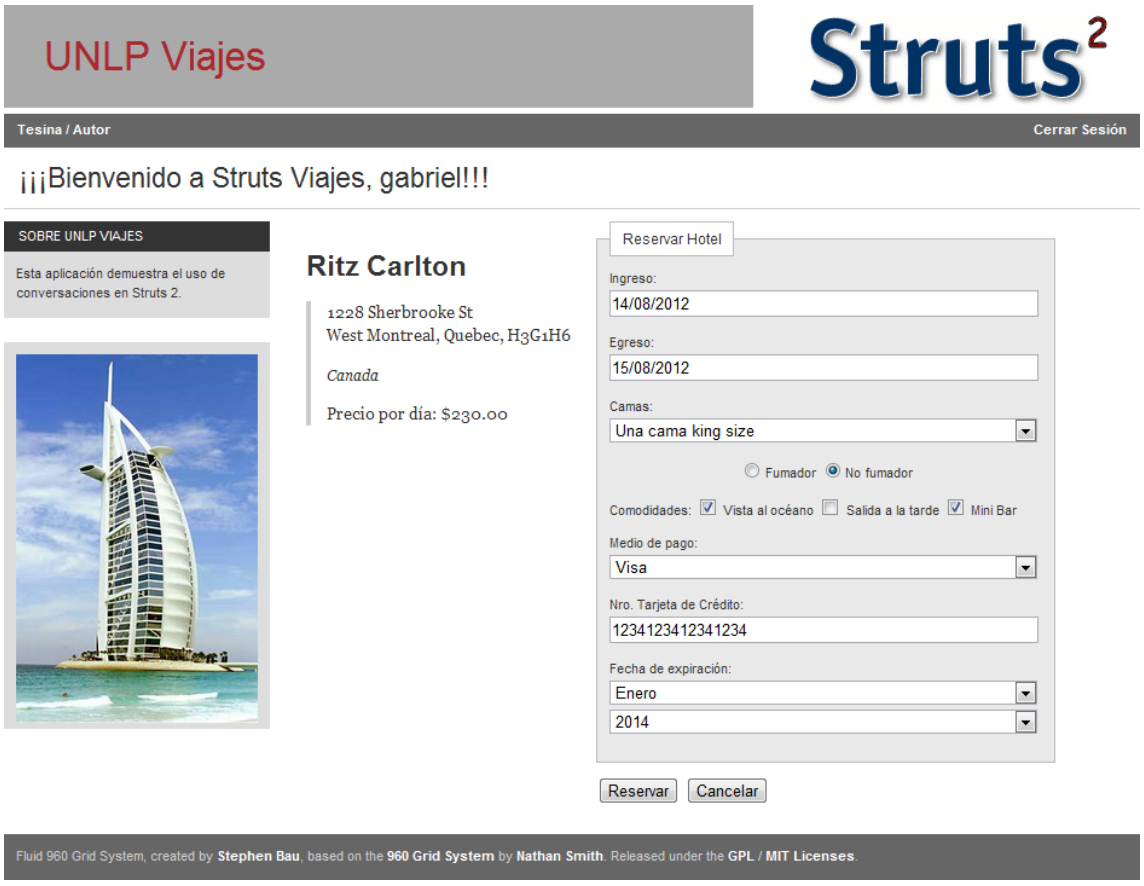

Figura 7.7: Reserva de hoteles: formulario de ingreso de datos.

Figura 7.8: Reserva de hoteles: confirmación de datos.

Figura 7.9: Reserva de hoteles: reserva confirmada.

Figura 7.10: Reserva de hoteles: confirmación de cancelación.

Note que esta página es mostrada mediante el patrón PRG (Post-Redirect-Get) lo que implica que si el usuario presiona el botón Refrescar del navegador, no se salvará una nueva reserva sino que se eliminará definitivamente la conversación terminada en la petición HTTP anterior.

7.7. Cancelación de Reservas

Tal como se muestra en la Figura 7.9, el sistema muestra una tabla de reservas efectuadas por el usuario. Cada fila de la tabla representa una reserva, y la última columna contiene las acciones posibles que pueden realizarse sobre una reserva. A efectos de demostración, la única acción disponible es la Cancelación de reservas (hipervínculo “Cancelar” en la tabla).

Al presionar sobre “Cancelar” el sistema mostrará una caja de dialogo que le permite al usuario confirmar (o arrepentirse) de la cancelación (Figura 7.10). Si el usuario confirma, el sistema elimina la reserva seleccionada de la base de datos y muestra la pantalla de la Figura 7.11, la cual se muestra utilizando el patrón PRG.

Lo importante de esta funcionalidad es que demuestra el uso de la creación de una conversación que solo dura una petición HTTP.[2] En efecto, eliminar la reserva (una instancia de la clase `Booking`, gestionada por JPA) en el contexto de una conversación permite de manera muy simple utilizar el patrón PRG, ya que la conversación se crea y se termina luego de completar la petición HTTP, pero se configura para que sus datos sean preservados hasta recibir una nueva petición sobre esa conversación (*i.e.* el redirect del PRG).

7.8. Construcción de la aplicación

La aplicación de ejemplo está desarrollada usando Maven [2]. Para construir la aplicación a partir del código fuente alcanza con ubicarse en el mismo directorio donde está ubicado el archi-

[2] Lo que en JBoss Seam es llamado una conversación temporaria (ver Sección 4.2).

Figura 7.11: Reserva de hoteles: reserva cancelada.

vo `pom.xml` y tipear el comando "`mvn clean package`". Al terminar de ejecutar, en la carpeta `target` se podrá encontrar el archivo `hotel.war` el cual puede instalarse en un contenedor web como Tomcat.

El código fuente puede consultarse en el Apéndice A, Sección A.2. Si bien no se muestra el código fuente de las pruebas unitarias, las mismas son accesibles desde el CD que acompaña la presente Tesina.

7.9. Resumen

En este Capítulo se demostró a alto nivel el desarrollo de una sistema que ejemplifica el uso de conversaciones. La reserva de hoteles es una aplicación que proveen otros frameworks de desarrollo con soporte de conversaciones y por lo tanto sirve como referencia para probar la implementación ofrecida en esta Tesina.

La funcionalidad de reserva de hoteles demuestra una conversación que dura múltiples peticiones HTTP, mientras que la cancelación de reserva demuestra el uso de una conversación que dura una única petición.

Capítulo 8

Conclusiones

La presente Tesina propuso la utilización de conversaciones en el ámbito del desarrollo de aplicaciones web en Java. Para ello se implementó un *plugin* de gestión de conversaciones para el framework Struts 2, así como una pequeña aplicación de ejemplo demostrando su uso.

A continuación se presentan los objetivos alcanzados y los resultados obtenidos de esta experiencia. Luego se proponen distintas ideas y posibles trabajos futuros para seguir investigando o complementar los resultados obtenidos.

8.1. Objetivos alcanzados

Se logró desarrollar una implementación de conversaciones para el framework Struts 2 mediante la creación de un *plugin*, que es el mecanismo de extensión por defecto del framework. Se seleccionó Struts 2 como base para el desarrollo porque no posee soporte nativo de conversaciones y por su gran base de usuarios instalada alrededor del mundo.

- **La implementación aportada** en esta Tesina provee las características y funcionalidades más básicas deseables como para que un desarrollador de aplicaciones pueda usarla con una curva de aprendizaje muy corta. Como Principios de Diseño se priorizaron la *simplicidad* en el uso (que las aplicaciones de desarrollen de una manera simple, a la vez que produzcan el resultado deseado) y el *estilo* de programación, *i.e.* que resulte natural comenzar a utilizar conversaciones a un desarrollador con experiencia en Struts 2. En la Sección 6.2.2 se enumeran las funcionalidades implementadas.
- **La aplicación de ejemplo** de reserva de habitaciones de hoteles proporciona un marco de referencia para evaluar el soporte de conversaciones. Tanto JBoss Seam como Spring Web Flow vienen con la *misma* aplicación implementada con su respectivo framework. La implementación en esta Tesina está basada en la de Spring Web Flow.

8.2. Resultados obtenidos

Luego de haber pasado por la experiencia de desarrollar el *plugin* de Struts 2 y la aplicación de ejemplo, pueden derivarse las conclusiones y resultados enumerados a continuación.

8.2.1. Mejor calidad del producto final

El uso de conversaciones en las aplicaciones web produce una mejora de la calidad de la misma en forma instantánea y gratuita, ya que el mecanismo de conversaciones previene algunos tipos de errores que son endémicos a las aplicaciones web, por ejemplo:

- El uso inapropiado del almacenamiento del estado de la aplicación dentro de la sesión de usuario, lo que redunda en comportamiento erróneo al utilizar la aplicación desde múltiples ventanas o pestañas del navegador web. Las conversaciones ayudan a ubicar el estado de la aplicación en un *scope* más conveniente, ya que tiene un alcance y tiempo de vida más reducido que el de la sesión de usuario.

- La ejecución inadvertida de un Action de la aplicación que no debería ser llamado a menos que se hayan ejecutado previamente otros, lo que sucede si se almacena la URL del Action dentro de los Favoritos del navegador web. Cuando este Action es parte de una conversación, y ésta no existe, entonces se procede de forma automática redirigiendo al usuario hacia una página de error.

- Cuando un usuario presiona el botón Refrescar del navegador web luego de enviar un formulario HTML, es posible que se ejecute nuevamente el Action que envía el formulario (posiblemente involucrando la ejecución de operaciones no idempotentes, como la modificación de datos en una base de datos). Las conversaciones ayudan en este caso ya que cuando la misma finaliza, no se podrá ejecutar nuevamente el mismo Action (ya que no podrá ser encontrada). Además todos los frameworks de desarrollo web estudiados aquí (y el implementado en esta Tesina) poseen soporte automático del patrón PRG (Post-Redirect-Get), el cual puede utilizarse en forma transparente para que cuando se presione el botón Refrescar, no se vuelva a llamar al Action que produce el procesamiento final, sino que se llama a otro que solo muestra resultados (*i.e.* no modifica el estado de la aplicación).

- Cuando se presiona el botón de Retroceder o Avanzar en el navegador web queda desincronizado el estado de la aplicación con respecto a lo que ve el usuario. Cuando el estado de la aplicación es almacenado en una conversación, el re-envío de un formulario (cuando la conversación no ha finalizado aún) solamente sobre-escribe los datos actuales de la conversación sin mayores repercusiones. Si la conversación terminó y se presiona el botón de Retroceder, un reenvío del formulario es detectado como inválido y redirige al usuario hacia una pantalla de error. Además, frameworks como Spring Web Flow poseen un modelo extendido de conversaciones basado en *continuations* que hace que cada interacción con el usuario sea distinta e independiente una de otra, de modo que el botón de Retroceder no lo afecta.

- El múltiple reenvío de formularios también puede producirse si la conexión hacia el servidor web posee poco ancho de banda, o bien cuando el procesamiento del servidor es demasiado prolongado y el usuario presiona nuevamente el botón de envío del formulario. En tal caso, es posible utilizar el patron Synchronizer Token para evitar procesar la misma petición HTTP nuevamente, pero el patrón original funciona con un token almacenado en la sesión de usuario, lo que lo hace vulnerable a envíos desde múltiples ventanas. La implementación del patrón Synchronizer Token que almacene el token en el contexto de la conversación corriente corrige automáticamente este problema.

8.2.2. Modelo de programación simplificado

Como se vio arriba, el uso de conversaciones para desarrollar una aplicación web permite aislar al programador de algunas interacciones indeseadas que puede producir el usuario, y además esto se realiza con un mínimo esfuerzo por parte del desarrollador ya que los frameworks estudiados soportan una gran cantidad de patrones resueltos como el Synchronizer Token o PRG, resultando en aplicaciones más robustas.

El uso de los frameworks con soporte nativo de conversaciones, al incorporar estos muchos mecanismos automáticos de control de la lógica de navegación, elevan el nivel de abstracción con que debe lidiar el desarrollador de aplicaciones. Los frameworks estudiados y el *plugin* de Struts 2 implementado ya realizan muchas de las tareas manuales que debe realizar el programador en caso de que no utilice conversaciones. A continuación se enumeran algunos beneficios proporcionados en cuanto a la simplificación del desarrollo de aplicaciones web:

- Por definición, la conversación provee un contexto de almacenamiento del estado de la aplicación capaz de sobrevivir entre múltiples peticiones HTTP (similar a la sesión de usuario) pero con un alcance y tiempo de vida mucho más reducido, y cuando la conversación termina los datos almacenados son automáticamente destruidos. Esto permite al desarrollador ubicar los datos que tienen un tiempo de vida limitado en el *scope* correcto, en vez de almacenarlos en la sesión de usuario o propagarlo entre petición y petición como parámetros ocultos de un formulario. Además la destrucción de los datos se produce sin escribir código explícitamente para este fin.

- Cuando se pretende que la aplicación tenga capacidad de operar desde múltiples ventanas abiertas por el mismo usuario, la forma tradicional de realizar este objetivo es reescribir el estado de la aplicación como campos ocultos de un formulario y esto es costoso de mantener ante nuevos requerimientos ya que al agregar una nueva variable al estado de la aplicación, también hay que agregar campos ocultos similares a las páginas que se renderizan. Las conversaciones evitan este problema ya que el estado es mantenido en el servidor y lo único que se propaga es el identificador de la conversación, el cual se realiza en forma automática.

- En Spring Web Flow directamente se diseña la interacción del usuario utilizando un lenguaje de alto nivel en XML donde se define el flujo de navegación de la aplicación así como los estados por los que va pasando y los eventos que permiten hacer transiciones entre ellos. Esto hace que sea un paso directo el pasar desde el prototipado de la lógica de navegación de la aplicación hacia el archivo de definición de flujos.

- Frameworks como JBoss Seam y el *plugin* para Struts 2 implementado ofrecen la funcionalidad de *dependency bijection*, lo que permite inyectar o almacenar el estado de la aplicación desde y hacia un *scope* en particular con tan solo declarar una variable de instancia o su correspondiente método *getter* o *setter*: esto proporciona mucho más legibilidad al código escrito, una propiedad importante si se tiene en cuenta que el código fuente se lee una cantidad mayor de veces que la que se escribe.

- Cuando se trabaja con la gestión de la persistencia de objetos mediante un framework de mapeo objeto-relacional como JPA o Hibernate, el uso de conversaciones usando un contexto de persistencia extendido (Sección 3.8.5) simplifica significativamente el desarrollo, ya que el manejo del contexto de persistencia lo realiza el framework en vez del programador. Es más, el uso del patrón *Open Session In View* resulta absolutamente innecesario cuando se utilizan conversaciones, ya que el contexto de persistencia se almacena (transparentemente para el programador) en la conversación misma.

- El patrón PRG generalmente se utiliza para mostrar pantallas que indican el éxito de una operación, mostrando datos relevantes a la misma. Estos datos deben almacenarse en algún lugar para luego recuperarlos luego de realizar el redirect HTTP. Los frameworks con soporte de conversaciones permiten almacenarlos como parte de los datos de la conversación misma y en forma transparente, en vez de utilizar otro *scope* como el *flash scope* o la sesión de usuario.

- Los modelos extendidos de conversaciones tales como los basados en *continuations* (implementado en Spring Web Flow) y con conversaciones anidadas (JBoss Seam) hacen más fácil y natural modelar cierto tipo de interacciones y mantener el estado de la aplicación en forma consistente. Implementar estos modelos en forma manual es sumamente engorroso y propenso a errores, de manera que es más confiable y transparente utilizar algún framework ya probado y que proporciona la funcionalidad en forma nativa.

8.2.3. Cuando usar conversaciones

Las conversaciones son un elemento más dentro de la caja de herramientas del desarrollador de aplicaciones web y como tal, existen situaciones en que su uso es muy conveniente, mientras que en otras situaciones no lo es. A continuación se detallan algunos aspectos a tener en cuenta al momento de tomar la decisión de utilizar esta abstracción.

- Productividad: Debido a la simplificación en la programación de la aplicación, desarrollar utilizando conversaciones provee un grado más alto de productividad con respecto a no utilizarlas esto es, utilizar la sesión de usuario como *scope* para mantener el estado de la aplicación o bien hacer reescritura de los datos entre páginas. Además los frameworks proveen gratuitamente la implementación de patrones como PRG o Synchronizer Token.

- Escalabilidad: Las aplicaciones que usan conversaciones también hacen un uso intensivo de la sesión de usuario, ya que es ahí donde generalmente se almacenan todas las conversaciones abiertas. Esto implica que el servidor hará uso de una mayor cantidad de memoria (comparado a reescribir el estado de la aplicación en cada petición HTTP), y en

los ambientes de *cluster* de servidores habrá una mayor comunicación en la red para sincronizar el estado entre cada nodo del *cluster*. Como consecuencia de lo anterior hay que analizar cuidadosamente la capacidad de escalar que tendrá la aplicación a medida que se incremente la cantidad de usuarios concurrentes que la estén utilizando.

- Patrones de interacción: Existen formas de interactuar con la aplicación que hacen que el uso de conversaciones sea una ventaja significativa dado que existe una correspondencia directa entre lo que ve el usuario en la pantalla con respecto a la lógica de navegación que debe programar el desarrollador. Los siguientes son solo algunos patrones de diseño que toman ventaja del uso de conversaciones (extraídos de la biblioteca de patrones de diseño en [37]):

  - *Breadcrumbs* visualiza la secuencia de navegación jerárquica que ha realizado el usuario, permitiendo volver a algún paso anterior. Aquí es conveniente almacenarlo en la conversación para que cada ventana muestre su propio historial.
  - *Action Button* ejecuta una acción importante en el sistema. Si la operación no es idempotente es inmediato utilizar una conversación junto con un PRG para ejecutar el Action.
  - *Stepping* y *Wizard* dictan esencialmente una secuencia de pasos donde la pantalla final normalmente salva el acumulado de información recolectada. Una conversación que termina cuando se procesa el último paso tiene una implementación directa.
  - *Paging* navega a través de una lista larga de elementos. Si se opta por salvar la lista en memoria, hacerlo en el contexto de una conversación brinda capacidad multi-ventana (mientras que almacenar en el contexto de la sesión de usuario no provee).
  - *Search Results* muestra una lista de resultados proveniente de una búsqueda realizada por el usuario, posiblemente abriendo cada elemento encontrado en una pestaña o ventana diferente. Aquí la capacidad multi-ventana de las conversaciones simplifican el diseño considerablemente.
  - *Booking* y *Product Configurator* genera una reserva (o bien otra operación) sobre un objeto, ya cargado en memoria o no. Como la operación genera modificaciones permanentes es conveniente efectuarla creando una conversación que contenga ese cambio.
  - *Country Selector* y *Language Selector* permiten visualizar el contenido de la aplicación por país, idioma u otro criterio. Las conversaciones permiten tener múltiples ventanas abiertas sobre el mismo conjunto de datos, visualizandolas con distintos criterios.

  Por otro lado, hay patrones que no resultan atractivos para usar con conversaciones, o bien no resultan en ninguna mejora tangible desde lo funcional, la calidad o bien simpleza del desarrollo, por ejemplo:

  - *Shopping Cart* donde se espera que el usuario solo gestiones un único carrito de compras por sesión. En este caso, el uso de conversaciones tal vez no es aconsejable ya que almacenarlo en el contexto de la sesión de usuario evita que se tengan múltiples carritos de compra por pestaña o ventana. Aunque el tiempo de vida del carrito de compras es más limitado que la sesión de usuario, efecto que debe subsanarse codificando su destrucción luego de haber terminado el proceso de compra. Aunque otra opción sería almacenar el carrito en una conversación, y luego codificar la lógica para que no pueda crearse más de uno por sesión de usuario. Ambas estrategias tienen sus desventajas.
  - *Poll* donde los usuarios seleccionan una opción de una lista dada para reflejar su opinión. Aquí depende si el mismo poll mostrado en diferentes ventanas tiene sentido que se completen en forma independiente o bien es posible completar la encuesta una sola vez por usuario (o sesión de usuario).

8.3. Mejoras y trabajos futuros

La presente Tesina tuvo como objetivo investigar el uso de conversaciones en el desarrollo de aplicaciones web en Java, cuyo resultado ha sido muy positivo, ya que provee una serie

de mejoras en la calidad y productividad de desarrollo a un costo muy bajo con respecto al esfuerzo de capacitación necesario para utilizarlas. Se ha desarrollado un prototipo de un *plugin* del framework Struts 2 para proveer soporte de conversaciones.

Sin embargo, mucho dista la implementación en particular realizada, y la investigación en general para entender el impacto del uso de las conversaciones. La siguiente es una enumeración de mejoras y trabajos futuros útiles para expandir el entendimiento actual de esta tecnología:

8.3.1. Estudios de productividad y calidad

Resulta evidente que la simplificación en el desarrollo de aplicaciones provista por el uso de conversaciones impacte positivamente en la productividad del programador, pero una interrogante justa sería investigar en que orden de magnitud se ve mejorada esta productividad en términos de comparar el esfuerzo requerido para desarrollar una aplicación de medio o gran porte utilizando conversaciones versus desarrollar la misma sin utilizar esta abstracción.

Igualmente positivo sería establecer en qué medida resulta en una mejora de calidad de las aplicaciones desarrolladas. Mediante esta Tesina se vio que algunas mejoras se obtienen en forma gratuita ya que los frameworks con soporte de conversaciones tienen un comportamiento estándar y ya probado, lo que beneficia al desarrollador ya que no debe escribir código para implementarlo, por ej. redirigir al usuario a una página cuando se intenta acceder a una conversación ya terminada o no existente.

8.3.2. Modelos extendidos de conversaciones y su usabilidad

La presente Tesina dio como resultado una implementación del modelo básico de conversaciones, pero sería interesante complementarlo con la adición de alguno de los modelos extendidos, ya sea basado en conversaciones anidadas o bien el basado en *continuations*. Ambos modelos tienen sus ventajas y desventajas.

Otro área de estudio sería identificar en qué grado estos modelos extendidos mejoran la aplicación desde el punto de vista de la usabilidad, navegabilidad y facilidad de uso. Por ejemplo, en el caso de una aplicación de reserva de pasajes de avión, donde se incorpora tanto la reserva de hoteles y de alquiler de autos: ¿Es mejor diseñar la aplicación de manera tal que el usuario busque un destino, que produce una lista de vuelos y seleccione uno de ellos, luego busque un hotel y seleccione uno (produciendo una conversación anidada) y por último que busque y seleccione un auto que querrá alquilar produciendo otra conversación anidada? ¿O es mejor sencillamente que estos tres procesos (selección de vuelo, de hotel y de auto) se lleven a cabo en forma independiente, cada una con una conversación que no esté relacionada con la anterior? En el primer caso, la navegavilidad implica cierto tipo de *wizard* para llevar a cabo las tres tareas, mientras que en el segundo caso se puede solucionar con un menú en la aplicación que permita realizar cada una de ellas en forma independiente.

8.3.3. Agregar funcionalidades a la implementación actual

Además de implementar algún modelo extendido de conversaciones, sería conveniente proveer extensiones para soportar el uso de otras herramientas, eliminar algunas dependencias con Spring que tiene la implementación actual, o bien simplemente agregar nueva funcionalidad no contemplada en esta Tesina.

Integración con otras tecnologías

Por ej. la integración del framework de mapeo objeto-relacional (ORM) soportada actualmente solo soporta JPA como herramienta de ORM utilizando Spring como gestor de transacciones. Sería conveniente que se puedan soportar las siguientes combinaciones, entre otras:

- Hibernate utilizando transacciones gestionadas por Spring.
- JPA utilizando transacciones gestionadas por un servidor de aplicaciones Java EE.
- Hibernate utilizando transacciones gestionadas por un servidor de aplicaciones Java EE.

Agregar Synchronizer Token

En la presente versión del *plugin* no se creó ningún interceptor de Struts 2 que implemente el patrón Synchronizer Token a nivel de conversación. Este interceptor no sería difícil de implementar si se sigue el diseño de los interceptores `token` y `tokenSession` provistos por Struts 2. Por supuesto, esta funcionalidad es necesaria solamente si se desea detectar múltiples envíos de formularios en Actions que procesen en el contexto de una conversación pero *sin terminar* la misma, ya que en el caso de que se termine, el reenvío se puede detectar automáticamente por otros medios.

Acceso directo a los datos de la conversación en Struts 2

La implementación actual pone a disposición de la página JSP a renderizar los datos almacenados en la conversación como variables del objeto `pageContext` usando explícitamente la etiqueta `<sconv:useConversation/>`. Sería muy conveniente que también estén disponibles en el `ValueStack` de Struts 2 en forma automática, pero implementar esto requiere un conocimiento más avanzado de Struts 2.

Generalizar el concepto de *scope* en Struts 2

Sería muy interesante que se pueda modificar el framework Struts 2 para que se puedan definir diferentes *scopes*, cada uno con sus reglas de alcance y tiempo de vida, así como un orden de búsqueda en caso de que se haga una referencia a una variable. Esto permitiría que se puedan integrar *plugins* que aporten nuevos *scopes* (por ej. el implementado en esta Tesina) tan solo modificando las reglas de operación de los mismos. Por ej. sería sencillo también implementar un *flash scope* o bien un *scope* para datos almacenados en una instancia de un *workflow* (gestionado por un motor de *workflow*, como posee JBoss Seam).

Esta modificación requiere poder extender el framework Struts 2 y publicar los cambios en la distribución original para que sea utilizable por toda la comunidad de desarrolladores.

8.3.4. Estudios de escalabilidad

Cuando se requiere desarrollar una aplicación que tendrá una gran base de usuarios concurrentes es necesario prestar especial atención a la escalabilidad de la misma ya que es una condición necesaria si se desea comenzar un emprendimiento con pocos recursos para luego ir aumentándolos según sea necesario para mejorar el tiempo de respuesta de la aplicación.

Se sabe que una arquitectura *stateless* es mucho más sencilla de escalar que una arquitectura *stateful*, en el sentido de que si la aplicación requiere un uso intensivo de la sesión de usuario (a nivel contenedor web) o bien se requiere crear un número significativo de *Stateful EJBs*, la memoria que necesitará el servidor será superior en el caso de la arquitectura *stateful*, así como la arquitectura de *clustering* de un conjunto de servidores requerirá más comunicación para mantenerse sincronizados. También puede cambiar la forma en que escala la aplicación dependiendo de si las sesiones HTTP se manejan con un servidor *proxy reverso* que gestione las *cookies* de sesión en forma *sticky* (pegajosa, significa que la misma sesión de usuario será atendida siempre por el mismo nodo) o *no sticky* (la misma sesión de usuario puede ser atendida por un nodo diferente cada vez que se reciba una petición HTTP).

Estos temas son esenciales al momento de definir la arquitectura de la aplicación, la cual es una actividad muy importante que tiene profundo impacto en la calidad de la solución.

8.3.5. Apertura del código fuente

La implementación realizada del *plugin* de Struts 2 para soporte de conversaciones es básico pero usable, en el sentido de que está probado con pruebas unitarias realizadas con JUnit, y por lo tanto se espera que contenga un número bajo de errores.

Se requerirían algunos cambios en la estructura del proyecto, adoptar una licencia de código libre, utilizar algún repositorio gratuito de proyectos (por ej. Google Code, Github o Sourceforge), y realizar cambios de variables (y otros nombres o documentación que se encuentren en castellano) para que estén todas definidas en inglés a efectos de que sea fácilmente modificable por alguien de la comunidad, de manera que pueda introducir mejoras y corregir errores.

Por último, sería deseable que la misma estuviese disponible en el Repositorio Central de Maven para una fácil integración con proyectos de terceras partes.

8.4. Resumen

Este Capítulo presenta la culminación del trabajo realizado para cumplir con los objetivos de la Tesina, específicamente el desarrollo de un *plugin* de Struts 2 para el soporte de conversaciones, así como ejemplificar su uso mediante el desarrollo de un sistema de reserva de habitaciones de hotel.

Los resultados obtenidos demuestran el valor de utilizar conversaciones en aplicaciones web. Por un lado se gana una mejora en la calidad que está directamente asociada a la utilización del framework con soporte de conversaciones, y por otro lado la mejora en la simplicidad y mantenibilidad obtenida al desarrollar basándose en una abstracción que captura con una gran fidelidad las funcionalidades básicas que componen las aplicaciones web.

Se enumeran bajo qué situaciones es conveniente utilizar conversaciones, así como los patrones de diseño de interacciones que mejor encajan en el modelo de programación, y los que no tienen una correspondencia tan directa.

Por último se realiza un listado de las posibles mejoras al framework desarrollado, se proponen estudios más extensos y rigurosos sobre la productividad y calidad generada, la escalabilidad de los sistemas que usan conversaciones, y la posibilidad de abrir el código fuente del proyecto para compartir con la comunidad mundial de desarrolladores de Struts 2.

Apéndice A

Código Fuente

A continuación se presenta el código fuente del plugin que implementa el soporte de conversaciones para Struts 2, así como de la aplicación de ejemplo demostrando la funcionalidad con un sistema de reserva de habitaciones de hotel. En ambos casos se omitió el código fuente con las pruebas unitarias (desarrolladas usando el framework JUnit [8]), sin embargo las mismas (así como el código fuente mostrado en este documento) pueden ser accedidas mediante el CD que acompaña la presente Tesina.

A.1. Plugin de conversaciones para Struts 2

A.1.1. pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.hexaid.struts2.conversations</groupId>
  <artifactId>struts2-conversations</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>struts2-conversations</name>
  <url>http://maven.apache.org</url>
  <description>Plugin de Struts 2 para utilizar conversaciones</description>
  <developers>
    <developer>
      <name>Gabriel Belingueres</name>
    </developer>
  </developers>
  <inceptionYear>2011</inceptionYear>

  <scm>
    <connection>scm:svn:https://riouxsvn.com/svn/conversations/trunk/proyectos/struts2-conversations</connection>
    <developerConnection>scm:svn:https://riouxsvn.com/svn/conversations/trunk/proyectos/struts2-conversations</
        developerConnection>
  </scm>

  <properties>
    <project.build.sourceEncoding>Cp1252</project.build.sourceEncoding>
    <struts2.version>2.2.3</struts2.version>
    <slf4j.version>1.6.4</slf4j.version>
    <spring.version>3.0.6.RELEASE</spring.version>

    <!-- Valores para el maven-compiler-plugin -->
    <maven.compiler.source>1.6</maven.compiler.source>
    <maven.compiler.target>1.6</maven.compiler.target>
    <maven.compiler.showWarnings>true</maven.compiler.showWarnings>
    <maven.compiler.showDeprecation>true</maven.compiler.showDeprecation>
    <maven.compiler.debuglevel>lines,vars,source</maven.compiler.debuglevel>
    <maven.compiler.verbose>true</maven.compiler.verbose>
  </properties>

  <build>
    <defaultGoal>test</defaultGoal>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>2.5.1</version>
        <configuration>
          <source>${maven.compiler.source}</source>
          <target>${maven.compiler.target}</target>
          <showWarnings>${maven.compiler.showWarnings}</showWarnings>
          <showDeprecation>${maven.compiler.showDeprecation}</showDeprecation>
          <debuglevel>${maven.compiler.debuglevel}</debuglevel>
          <verbose>${maven.compiler.verbose}</verbose>
          <!-- las siguiente propiedades actuan si fork es true -->
          <fork>true</fork>
          <compilerArgument>-Xlint:all</compilerArgument>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-eclipse-plugin</artifactId>
        <version>2.9</version>
        <configuration>
          <downloadSources>true</downloadSources>
          <downloadJavadocs>true</downloadJavadocs>
```

```
64          <workspaceCodeStylesURL>http://svn.apache.org/repos/asf/maven/plugins/trunk/maven-eclipse-plugin/src/optional/
                eclipse-config/maven-styles.xml</workspaceCodeStylesURL>
65          <useProjectReferences>false</useProjectReferences>
66        </configuration>
67      </plugin>
68      <plugin>
69        <groupId>org.apache.maven.plugins</groupId>
70        <artifactId>maven-source-plugin</artifactId>
71        <version>2.2</version>
72        <executions>
73          <execution>
74            <id>attach-sources</id>
75            <goals>
76              <goal>jar</goal>
77            </goals>
78          </execution>
79        </executions>
80      </plugin>
81      <plugin>
82        <groupId>org.apache.maven.plugins</groupId>
83        <artifactId>maven-javadoc-plugin</artifactId>
84        <version>2.8.1</version>
85        <configuration>
86          <minmemory>128m</minmemory>
87          <maxmemory>256m</maxmemory>
88          <quiet>true</quiet>
89          <links>
90            <link>http://download.oracle.com/javase/6/docs/api/</link>
91            <link>http://struts.apache.org/2.2.3/struts2-core/apidocs/</link>
92            <link>http://struts.apache.org/2.2.3/xwork-core/apidocs/</link>
93          </links>
94        </configuration>
95        <executions>
96          <execution>
97            <id>attach-javadocs</id>
98            <goals>
99              <goal>jar</goal>
100           </goals>
101         </execution>
102       </executions>
103     </plugin>
104   </plugins>
105
106   <!-- new plugin versions available from Super POM -->
107   <pluginManagement>
108     <plugins>
109       <plugin>
110         <groupId>org.apache.maven.plugins</groupId>
111         <artifactId>maven-jar-plugin</artifactId>
112         <version>2.4</version>
113       </plugin>
114       <plugin>
115         <groupId>org.apache.maven.plugins</groupId>
116         <artifactId>maven-dependency-plugin</artifactId>
117         <version>2.2</version>
118       </plugin>
119     </plugins>
120   </pluginManagement>
121 </build>
122
123 <dependencies>
124   <dependency>
125     <groupId>org.apache.struts</groupId>
126     <artifactId>struts2-core</artifactId>
127     <version>${struts2.version}</version>
128   </dependency>
129   <dependency>
130     <groupId>org.apache.struts.xwork</groupId>
131     <artifactId>xwork-core</artifactId>
132     <version>${struts2.version}</version>
133   </dependency>
134   <dependency>
135     <groupId>org.slf4j</groupId>
136     <artifactId>slf4j-api</artifactId>
137     <version>${slf4j.version}</version>
138   </dependency>
139   <dependency>
140     <groupId>javax.servlet</groupId>
141     <artifactId>servlet-api</artifactId>
142     <version>2.5</version>
143   </dependency>
144   <dependency>
145     <groupId>commons-lang</groupId>
146     <artifactId>commons-lang</artifactId>
147     <version>2.6</version>
148   </dependency>
149   <dependency>
150     <groupId>javax.servlet.jsp</groupId>
151     <artifactId>jsp-api</artifactId>
152     <version>2.0</version>
153     <scope>provided</scope>
154   </dependency>
155
156   <!-- For Tiles result type -->
157   <dependency>
158     <groupId>org.apache.struts</groupId>
159     <artifactId>struts2-tiles-plugin</artifactId>
160     <version>${struts2.version}</version>
161     <optional>true</optional>
162   </dependency>
163
164   <!-- Conversation Managed Persistence Context -->
165   <dependency>
166     <groupId>org.hibernate.javax.persistence</groupId>
167     <artifactId>hibernate-jpa-2.0-api</artifactId>
168     <version>1.0.1.Final</version>
169     <optional>true</optional>
170   </dependency>
171   <dependency>
172     <groupId>org.springframework</groupId>
173     <artifactId>spring-tx</artifactId>
174     <version>${spring.version}</version>
175     <optional>true</optional>
176   </dependency>
177   <dependency>
178     <groupId>org.springframework</groupId>
179     <artifactId>spring-orm</artifactId>
```

```
      <version>${spring.version}</version>
      <optional>true</optional>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-web</artifactId>
      <version>${spring.version}</version>
      <optional>true</optional>
    </dependency>

    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.10</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.apache.struts</groupId>
      <artifactId>struts2-junit-plugin</artifactId>
      <version>${struts2.version}</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-simple</artifactId>
      <version>${slf4j.version}</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>com.hexaid.struts2.junit</groupId>
      <artifactId>struts2-junit</artifactId>
      <version>0.2.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-test</artifactId>
      <version>${spring.version}</version>
      <scope>test</scope>
    </dependency>
    <!-- for testing PersistenceTransactionManagers -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-jdbc</artifactId>
      <version>${spring.version}</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.hsqldb</groupId>
      <artifactId>hsqldb</artifactId>
      <version>2.2.8</version>
      <classifier>jdk5</classifier>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.hibernate</groupId>
      <artifactId>hibernate-entitymanager</artifactId>
      <version>3.6.8.Final</version>
      <scope>test</scope>
    </dependency>
    <!-- for testing custom tags -->
    <dependency>
      <groupId>jstl</groupId>
      <artifactId>jstl</artifactId>
      <version>1.2</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Listado A.1: pom.xml

A.1.2. Begin.java

```
package com.hexaid.struts2.annotations;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
 * @author Gabriel Belingueres
 *
 */
@Retention(RetentionPolicy.RUNTIME)
@Target(value = { ElementType.METHOD })
public @interface Begin {

  /**
   * The name of an initialization method inside the Action class
   * that should be executed before the action method itself to
   * allow initialization of conversation related code.
   */
  String initialization() default "";

  /**
   * If not empty, it means the newly created conversation will
   * be a natural conversation, where the natural conversation
   * Id will be extracted from the OGNL specified.<br/>
   * If empty, the newly created conversation will have a
   * synthetic conversation Id.
   */
  String naturalIdExpression() default "";
}
```

Listado A.2: Begin.java

A.1.3. ConversationAttribute.java

```java
package com.hexaid.struts2.annotations;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

import com.hexaid.struts2.common.ConversationAttributeType;

/**
 * @author Gabriel Belingueres
 *
 */
@Retention(RetentionPolicy.RUNTIME)
@Target(value = { ElementType.METHOD })
public @interface ConversationAttribute {

  /**
   * Attribute type declared in ConversationAttributeType which states
   * in which way the conversation will be handled. Default is REQUIRED.
   */
  ConversationAttributeType value() default ConversationAttributeType.REQUIRED;

}
```

Listado A.3: ConversationAttribute.java

A.1.4. End.java

```java
package com.hexaid.struts2.annotations;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

import com.opensymphony.xwork2.ActionSupport;

/**
 * @author Gabriel Belingueres
 *
 */
@Retention(RetentionPolicy.RUNTIME)
@Target(value = { ElementType.METHOD })
public @interface End {

  /**
   * Set if the current conversation must retain its values
   * after an HTTP redirect (default) or not (false).
   */
  boolean beforeRedirect() default false;

  /**
   * Set if the current conversation must commit its
   * transaction (default) or must rollback all persistence
   * related operations (false).
   */
  boolean commit() default true;

  /**
   * Only ends the conversation if the Action result is
   * the specified here. Defaults to ActionSupport.SUCCESS.
   */
  String endResult() default ActionSupport.SUCCESS;
}
```

Listado A.4: End.java

A.1.5. In.java

```java
package com.hexaid.struts2.annotations;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

import com.hexaid.struts2.common.ScopeType;

/**
 *
 * @author Gabriel Belingueres
 *
 */
@Retention(RetentionPolicy.RUNTIME)
@Target(value = { ElementType.FIELD, ElementType.METHOD })
public @interface In {

  /**
   * The variable name used as a key to save the value
   * in the scope from which the value will be injected.
   * Defauls to the variable name if applied to a property,
   * or to the property name if applied to a setter method.
   */
  String value() default "";

  /**
   * The scope where the value should be injected from.
   * Defaults to CONVERSATION.
   * @see ScopeType
   */
  ScopeType scope() default ScopeType.CONVERSATION;

  /**
   * Specifies if the injected value is required to be
   * not null. Default to false.
   */
  boolean required() default false;

  /**
```

```
   * Specifies that the value should be created if not
   * found in the specified context. Default to false.
   *
   * Note: the class to instantiate should have a public
   * no-arg constructor.
   */
  boolean create() default false;
}
```

Listado A.5: In.java

A.1.6. Out.java

```
package com.hexaid.struts2.annotations;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

import com.hexaid.struts2.common.ScopeType;

/**
 * @author Gabriel Belingueres
 *
 */
@Retention(RetentionPolicy.RUNTIME)
@Target(value = { ElementType.FIELD, ElementType.METHOD })
public @interface Out {
  /**
   * The variable name used as a key to save the value
   * in the scope where the value will be outjected.
   * Defauls to the variable name if applied to a property,
   * or to the property name if applied to a setter method.
   */
  String value() default "";

  /**
   * The scope where the value should be outjected to.
   * Defaults to CONVERSATION.
   * @see ScopeType
   */
  ScopeType scope() default ScopeType.CONVERSATION;

  /**
   * Specifies if the outjected value is required to be
   * not null. Default to false.
   */
  boolean required() default false;
}
```

Listado A.6: Out.java

A.1.7. AbstractBijector.java

```
package com.hexaid.struts2.bijection;

import java.lang.annotation.Annotation;
import java.lang.reflect.AccessibleObject;
import java.lang.reflect.Member;
import java.util.List;

import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;

import org.apache.struts2.ServletActionContext;
import org.apache.struts2.StrutsException;

import com.hexaid.struts2.annotations.In;
import com.hexaid.struts2.annotations.Out;
import com.hexaid.struts2.common.ConversationUtils;
import com.hexaid.struts2.conversations.Conversation;

/**
 * @author Gabriel Belingueres
 *
 */
public abstract class AbstractBijector<T extends AccessibleObject & Member> implements Bijector {

  @Override
  public void doInjections(Object action, Class<?> actionClass) {
    List<T> elements = getElementsWithAnnotation(In.class, actionClass);

    for(final T element : elements) {
      // allow to set private and protected fields
      element.setAccessible(true);
      final In annotationIn = element.getAnnotation(In.class);

      final String name = resolveName(annotationIn.value(), element);

      final Class<?> classToInject = getClassToInject(element);

      Object valueToInject = injectSpecialObjectByType(classToInject);

      if (valueToInject == null) {
        valueToInject = annotationIn.scope().get(name);
      }

      if (valueToInject == null) {
        if (annotationIn.required()) {
          throw new IllegalArgumentException("The value to inject in the " + element.getClass() + " <" + element.getName() +
                "> can not be null");
        }
        else if (annotationIn.create()) {
          // create an instance of the type that must be injected
          try {
            valueToInject = classToInject.newInstance();
          } catch (InstantiationException e) {
```

```
54              throw new StrutsException("An instantiation exception happened when trying to instantiate an object of class <" +
                      classToInject.getName() +"> during injection", e);
55            } catch (IllegalAccessException e) {
56              throw new StrutsException("An illegal access happened when trying to instantiate an object of class <" +
                    classToInject.getName() +"> during injection", e);
57            }
58          }
59        }
60
61        inject(element, action, valueToInject);
62      }
63    }
64
65    @Override
66    public void doOutjections(Object action, Class<?> actionClass) {
67      final List<T> elements = getElementsWithAnnotation(Out.class, actionClass);
68      for(final T element : elements) {
69        // allow to set private and protected fields
70        element.setAccessible(true);
71        final Out annotationOut = element.getAnnotation(Out.class);
72
73        final String name = resolveName(annotationOut.value(), element);
74
75        final Object valueToOutject = getValueToOutject(element, action);
76
77        if (annotationOut.required() && valueToOutject == null) {
78          throw new IllegalArgumentException("The value to outject from the "+element.getClass().getSimpleName()+" <" + element
                .getName() + "> can not be null");
79        }
80
81        annotationOut.scope().put(name, valueToOutject);
82      }
83    }
84
85    protected Object injectSpecialObjectByType(final Class<?> type) {
86      if (type.isAssignableFrom(ServletContext.class)) {
87        return ServletActionContext.getServletContext();
88      }
89      else if (type.isAssignableFrom(HttpSession.class)) {
90        return ServletActionContext.getRequest().getSession(false);
91      }
92      else if (type.isAssignableFrom(Conversation.class)) {
93        return ConversationUtils.getCurrentConversation();
94      }
95      else if (type.isAssignableFrom(HttpServletRequest.class)) {
96        return ServletActionContext.getRequest();
97      }
98      return null;
99    }
100
101   // general
102   protected abstract List<T> getElementsWithAnnotation(final Class<? extends Annotation> annotationClass, final Class<?>
          actionClass);
103
104   protected abstract String resolveName(String valueDeclaredInAnnotation, T element);
105
106   // injection related
107   protected abstract Class<?> getClassToInject(T element);
108
109   protected abstract void inject(T element, Object action, Object valueToInject);
110
111   // outjection related
112   protected abstract Object getValueToOutject(T element, Object action);
113
114 }
```

Listado A.7: AbstractBijector.java

A.1.8. Bijector.java

```
1 package com.hexaid.struts2.bijection;
2
3 public interface Bijector {
4
5   public void doOutjections(Object action, Class<?> actionClass);
6
7   public void doInjections(Object action, Class<?> actionClass);
8
9 }
```

Listado A.8: Bijector.java

A.1.9. FieldBijector.java

```
1  package com.hexaid.struts2.bijection;
2
3  import java.lang.annotation.Annotation;
4  import java.lang.reflect.Field;
5  import java.util.ArrayList;
6  import java.util.List;
7
8  import org.apache.commons.lang.StringUtils;
9  import org.apache.struts2.StrutsException;
10
11 import com.opensymphony.xwork2.util.AnnotationUtils;
12
13 /**
14  * @author Gabriel Belingueres
15  *
16  */
17 public class FieldBijector extends AbstractBijector<Field> {
18
19   @Override
20   protected List<Field> getElementsWithAnnotation(Class<? extends Annotation> annotationClass,
21                         Class<?> actionClass) {
22     final List<Field> allInFields = new ArrayList<Field>();
23     AnnotationUtils.addAllFields(annotationClass, actionClass, allInFields);
24     return allInFields;
25   }
26
```

```
27   @Override
28   protected String resolveName(String valueDeclaredInAnnotation, Field element) {
29     return StringUtils.isEmpty(valueDeclaredInAnnotation) ? element.getName() : valueDeclaredInAnnotation;
30   }
31
32   @Override
33   protected Class<?> getClassToInject(Field element) {
34     return element.getType();
35   }
36
37   @Override
38   protected void inject(Field element, Object action, Object valueToInject) {
39     try {
40       element.set(action, valueToInject);
41     } catch (IllegalArgumentException e) {
42       throw new StrutsException("An illegal argument to the field <" + element.getName() +"> has been tried on injection", e)
             ;
43     } catch (IllegalAccessException e) {
44       throw new StrutsException("An illegal access to the field <" + element.getName() +"> has occurred on injection", e);
45     }
46   }
47
48   @Override
49   protected Object getValueToOutject(Field element, Object action) {
50     try {
51       return element.get(action);
52     } catch (IllegalArgumentException e) {
53       throw new StrutsException("An illegal argument to the field <" + element.getName() +"> has been tried on outjection", e
             );
54     } catch (IllegalAccessException e) {
55       throw new StrutsException("An illegal access to the field <" + element.getName() +"> has occurred on outjection", e);
56     }
57   }
58
59 }
```

Listado A.9: FieldBijector.java

A.1.10. MethodBijector.java

```
1  package com.hexaid.struts2.bijection;
2
3  import java.lang.annotation.Annotation;
4  import java.lang.reflect.InvocationTargetException;
5  import java.lang.reflect.Method;
6  import java.util.ArrayList;
7  import java.util.List;
8
9  import org.apache.commons.lang.StringUtils;
10 import org.apache.struts2.StrutsException;
11
12 import com.opensymphony.xwork2.util.AnnotationUtils;
13
14 /**
15  * @author Gabriel Belingueres
16  *
17  */
18 public class MethodBijector extends AbstractBijector<Method> {
19
20   @Override
21   protected List<Method> getElementsWithAnnotation( Class<? extends Annotation> annotationClass,
22                             Class<?> actionClass) {
23     final List<Method> allOutMethods = new ArrayList<Method>();
24     AnnotationUtils.addAllMethods(annotationClass, actionClass, allOutMethods);
25     return allOutMethods;
26   }
27
28   @Override
29   protected String resolveName(String valueDeclaredInAnnotation, Method element) {
30     String name = null;
31     if (StringUtils.isEmpty(valueDeclaredInAnnotation)) {
32       name = AnnotationUtils.resolvePropertyName(element);
33     }
34     else {
35       name = valueDeclaredInAnnotation;
36     }
37
38     if (name == null) {
39       throw new IllegalArgumentException("The method <"+ element.toGenericString() +"> can not be injected since there is no
             suitable name found");
40     }
41
42     return name;
43   }
44
45   @Override
46   protected Class<?> getClassToInject(Method element) {
47     return element.getParameterTypes()[0];
48   }
49
50   @Override
51   protected void inject(Method element, Object action, Object valueToInject) {
52     try {
53       element.invoke(action, valueToInject);
54     } catch (IllegalArgumentException e) {
55       throw new StrutsException("An illegal argument to the method <" + element.toGenericString() +"> has been tried on
             injection", e);
56     } catch (IllegalAccessException e) {
57       throw new StrutsException("An illegal access to the method <" + element.toGenericString() +"> has occurred on injection
             ", e);
58     } catch (InvocationTargetException e) {
59       throw new StrutsException("An exception has been throwed by the method <" + element.toGenericString() +"> on injection"
             , e);
60     }
61   }
62
63   @Override
64   protected Object getValueToOutject(Method element, Object action) {
65     try {
66       return element.invoke(action, (Object[])null);
67     } catch (IllegalArgumentException e) {
68       throw new StrutsException("An illegal argument to the method <" + element.getName() +"> has been tried on outjection",
             e);
69     } catch (IllegalAccessException e) {
```

```
      throw new StrutsException("An illegal access to the method <" + element.getName() +"> has occurred on outjection", e);
    } catch (InvocationTargetException e) {
      throw new StrutsException("An exception has been throwed by the method <" + element.getName() +"> on outjection", e);
    }
  }

}
```

Listado A.10: MethodBijector.java

A.1.11. ConversationAttributeType.java

```
package com.hexaid.struts2.common;

import java.util.Map;

import org.apache.commons.lang.StringUtils;

import com.hexaid.struts2.conversations.Conversation;
import com.hexaid.struts2.conversations.ConversationFactory;

/**
 * @author Gabriel Belingueres
 *
 */
public enum ConversationAttributeType {

  /**
   * Use REQUIRED when a new conversation needs to be created if none is present.
   */
  REQUIRED {
    @Override
    public Conversation process(final ConversationFactory conversationFactory,
                  final Map<String, Conversation> allConversationsMap,
                  final String conversationId,
                  final boolean hasNaturalIdExpression) {
      // check the conversationId parameter
      if (StringUtils.isBlank(conversationId)) {
        return REQUIRES_NEW.process(conversationFactory, allConversationsMap, conversationId, hasNaturalIdExpression);
      } else {
        // conversationID was present => get the conversation
        return allConversationsMap.get(conversationId);
      }
    }
  },

  /**
   * Use REQUIRES_NEW when a new conversation always needs to be created.
   */
  REQUIRES_NEW {
    @Override
    public Conversation process(final ConversationFactory conversationFactory,
                  final Map<String, Conversation> allConversationsMap,
                  final String conversationId,
                  final boolean hasNaturalIdExpression) {
      // creates a new conversation
      if (hasNaturalIdExpression) {
        // the id will be resolved LATER!
        return conversationFactory.createNaturalConversation();
      }
      else {
        return conversationFactory.createConversation();
      }
    }
  },

  /**
   * Use MANDATORY when a conversation must be already created.
   */
  MANDATORY {
    @Override
    public Conversation process(final ConversationFactory conversationFactory,
                  final Map<String, Conversation> allConversationsMap,
                  final String conversationId,
                  final boolean hasNaturalIdExpression) {
      // just get the conversation from the Map
      if (StringUtils.isBlank(conversationId)) {
        return null;
      }
      return allConversationsMap.get(conversationId);
    }
  },

  SUPPORTS {
        @Override
        public Conversation process(final ConversationFactory conversationFactory,
                                    final Map<String, Conversation> allConversationsMap,
                  final String conversationId,
                  final boolean hasNaturalIdExpression) {
            return MANDATORY.process(conversationFactory, allConversationsMap, conversationId, hasNaturalIdExpression);
        }
  },

  /**
   * Use NONE when a conversation must NOT be created.
   */
  NONE {
    @Override
    public Conversation process(final ConversationFactory conversationFactory,
        Map<String, Conversation> allConversationsMap,
        final String conversationId,
        final boolean hasNaturalIdExpression) {
      throw new IllegalStateException("Should not reach here");
    }
  };

  public abstract Conversation process(final ConversationFactory conversationFactory,
                    final Map<String, Conversation> allConversationsMap,
                    final String conversationId,
                    final boolean hasNaturalIdExpression);
}
```

Listado A.11: ConversationAttributeType.java

A.1.12. ConversationUtils.java

```java
package com.hexaid.struts2.common;

import java.util.Map;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;

import org.apache.commons.lang.StringUtils;
import org.apache.struts2.ServletActionContext;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.hexaid.struts2.conversations.Conversation;

/**
 * @author Gabriel Belingueres
 *
 */
public final class ConversationUtils {

  private static final Logger LOG = LoggerFactory.getLogger(ConversationUtils.class);

  private ConversationUtils() {
    // do not instantiate
  }

  public static Conversation getCurrentConversation(final HttpServletRequest request) {
    Conversation conversation = (Conversation)
        request.getAttribute(Conversation.CURRENT_CONVERSATION_KEY);
    if (conversation == null) {
      final String conversationId = request.getParameter(Conversation.CONVERSATION_ID_PARAM);
      if (StringUtils.isNotEmpty(conversationId)) {
        final HttpSession session = request.getSession();
        @SuppressWarnings("unchecked")
        final Map<String, Conversation> conversationMap =
            (Map<String, Conversation>) session.getAttribute(Conversation.CONVERSATIONS_MAP_KEY);
        if (conversationMap != null) {
          conversation = conversationMap.get(conversationId);
        }
      }
    }
    return conversation;
  }

  public static Conversation getCurrentConversation() {
    final HttpServletRequest request = ServletActionContext.getRequest();
    return ConversationUtils.getCurrentConversation(request);
  }

  public static void changeMaxInactiveInterval(final int maxInactiveInterval) {
    final Conversation currentConversation = getCurrentConversation();
    if (currentConversation != null) {
      LOG.debug(
          "Changed the maxInactiveInterval to {} seconds to conversation id [{}]",
          maxInactiveInterval, currentConversation.getId());
      currentConversation.setMaxInactiveInterval(maxInactiveInterval);
    }
  }

}
```

Listado A.12: ConversationUtils.java

A.1.13. ScopeType.java

```java
package com.hexaid.struts2.common;

import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts2.ServletActionContext;

import com.hexaid.struts2.conversations.Conversation;

/**
 * @author Gabriel Belingueres
 * @since 1.0
 */
public enum ScopeType {
  REQUEST {
    @Override
    public void put(final String name, final Object value) {
      HttpServletRequest request = ServletActionContext.getRequest();
      request.setAttribute(name, value);
    }

    @Override
    public Object get(String name) {
      HttpServletRequest request = ServletActionContext.getRequest();
      return request.getAttribute(name);
    }
  },
  CONVERSATION {
    @Override
    public void put(String name, Object value) {
      final HttpServletRequest request = ServletActionContext.getRequest();
      final Conversation conversation = ConversationUtils.getCurrentConversation(request);
      if (conversation != null && !conversation.isEnded()) {
        conversation.getMap().put(name, value);
      }
    }

    @Override
    public Object get(String name) {
      final HttpServletRequest request = ServletActionContext.getRequest();
      final Conversation conversation = ConversationUtils.getCurrentConversation(request);
      if (conversation == null || conversation.isEnded()) {
        return null;
      }
```

```
46      return conversation.getMap().get(name);
47    }
48  },
49  SESSION {
50    @Override
51    public void put(String name, Object value) {
52      ServletActionContext.getContext().getSession().put(name, value);
53    }
54
55    @Override
56    public Object get(String name) {
57      return ServletActionContext.getContext().getSession().get(name);
58    }
59  },
60  APPLICATION {
61    @Override
62    public void put(String name, Object value) {
63      ServletActionContext.getContext().getApplication().put(name, value);
64    }
65
66    @Override
67    public Object get(String name) {
68      return ServletActionContext.getContext().getApplication().get(name);
69    }
70  },
71  COOKIE {
72    @Override
73    public void put(String name, Object value) {
74      if (value == null) {
75        return;
76      }
77
78      final HttpServletResponse response = ServletActionContext.getResponse();
79      final Cookie cookie;
80      if (value instanceof Cookie) {
81        cookie = (Cookie) value;
82      }
83      else {
84        cookie = new Cookie(name, value.toString());
85      }
86      response.addCookie(cookie);
87    }
88
89    @Override
90    public Object get(String name) {
91      final HttpServletRequest request = ServletActionContext.getRequest();
92      final Cookie[] cookies = request.getCookies();
93      for(int i=0; i < cookies.length; ++i) {
94        if (cookies[i].getName().equals(name)) {
95          return cookies[i];
96        }
97      }
98      return null;
99    }
100 };
101
102 /**
103  * Saves a value into the scope, identified by its name.
104  * @param name the name of the object to save.
105  * @param value the state to save into the scope.
106  */
107 public abstract void put(final String name, final Object value);
108
109 /**
110  * Returns the value associated with name parameter stored in the scope.
111  * @param name the name of the object to return.
112  * @return the value of the object stored in the scope.
113  */
114 public abstract Object get(final String name);
115 }
```

Listado A.13: ScopeType.java

A.1.14. ServletConversationUrlRenderer.java

```
1  package com.hexaid.struts2.components;
2
3  import java.util.Map;
4
5  import org.apache.struts2.components.ServletUrlRenderer;
6  import org.apache.struts2.components.UrlProvider;
7
8  import com.hexaid.struts2.common.ConversationUtils;
9  import com.hexaid.struts2.conversations.Conversation;
10
11 /**
12  * @author Gabriel Belingueres
13  *
14  */
15 public class ServletConversationUrlRenderer extends ServletUrlRenderer {
16
17   private static final String CONVERSATION_PROPAGATION_PARAM = "conversationPropagation";
18
19   @Override
20   public void beforeRenderUrl(UrlProvider urlComponent) {
21     final String conversationPropagation = urlComponent.findString(CONVERSATION_PROPAGATION_PARAM);
22     if (conversationPropagation != null && "join".equalsIgnoreCase(conversationPropagation)) {
23       final Conversation currentConversation = ConversationUtils.getCurrentConversation();
24       if (currentConversation != null) {
25         @SuppressWarnings("unchecked")
26         Map<String, String> parameters = urlComponent.getParameters();
27         parameters.put(Conversation.CONVERSATION_ID_PARAM, currentConversation.getId());
28       }
29       // do not include the "conversationPropagation" parameter in the final URL
30       // a little bit disturbing it is not in the final URL, but since it is only
31       // a control parameter it is OK
32       //parameters.remove(CONVERSATION_PROPAGATION_PARAM);
33     }
34     super.beforeRenderUrl(urlComponent);
35   }
36
```

```
}
```

Listado A.14: ServletConversationUrlRenderer.java

A.1.15. Conversation.java

```
package com.hexaid.struts2.conversations;

import java.io.Serializable;
import java.util.HashMap;
import java.util.Map;

/**
 * Clase central del plugin, que representa el objeto Conversacin con el los
 * datos salvados en el contexto.
 *
 * @author Gabriel Belingueres
 * @version 1.0
 */
public class Conversation implements Serializable {

  private static final long serialVersionUID = 1L;

  public static final String CONVERSATIONS_MAP_KEY = "com.hexaid.struts2.conversations.conversationsMap";
  public static final String CONVERSATION_ID_PARAM = "conversationId";
  public static final String CURRENT_CONVERSATION_KEY = "com.hexaid.struts2.conversations.currentConversation";
  public static final String CURRENT_CONVERSATION_MESSAGES_KEY = "com.hexaid.struts2.conversations.currentConversation.
        messages";

  public static final String CONVERSATION_PROPAGATION_JOIN = "join";
  public static final String CONVERSATION_PROPAGATION_NONE = "none";

  private String id;
  private Map<String, Object> map;
  private boolean isANewConversation;
  private boolean ended;

  /**
   * The conversation timeout
   */
  private int maxInactiveInterval;

  /**
   * the last time the conversation was accessed (for timeout evaluation)
   */
  private long lastAccessTime;

  /**
   * maximum quantity of requests the conversation can be accessed after it was ended
   * defaults to 1 (that is, like a flash scope)
   */
  private int maxRequestCountAfterEnded = 1;
  private int requestCountAfterEnded = 0;

  /**
   * True if the conversation has a natural Id. False otherwise.
   * This is an immutable field.
   */
  final private boolean naturalId;

  public Conversation() {
    this(null);
  }

  public Conversation(final String id) {
    this(id, System.currentTimeMillis());
  }

  public Conversation(final String id, final long creationTime) {
    this(id, creationTime, false);
  }

  public Conversation(final String id, final long creationTime, final boolean naturalId) {
    this.id = id;
    this.map = new HashMap<String, Object>();
    this.isANewConversation = true;
    this.ended = false;
    this.lastAccessTime = creationTime;
    this.naturalId = naturalId;
  }

  public void setId(String id) {
    this.id = id;
  }

  public void setMap(Map<String, Object> map) {
    this.map = map;
  }

  public void setNew(boolean isNew) {
    this.isANewConversation = isNew;
  }

  /**
   * Don't use this method directly! Use {@link #end(boolean)} instead.
   * @param ended true or false
   */
  public void setEnded(boolean ended) {
    this.ended = ended;
  }

  public String getId() {
    return id;
  }

  public Map<String, Object> getMap() {
    return map;
  }

  public boolean isNew() {
    return isANewConversation;
  }

  public boolean isEnded() {
```

```
108     return ended;
109   }
110
111   public void end(final boolean beforeRedirect) {
112     this.ended = true;
113     if (beforeRedirect) {
114       // forces not to survive a single redirect
115       maxRequestCountAfterEnded = 0;
116     }
117   }
118
119   public boolean isMarkedForDeletion() {
120     // > for safety....
121     return requestCountAfterEnded >= maxRequestCountAfterEnded;
122   }
123
124   public int getMaxRequestCountAfterEnded() {
125     return maxRequestCountAfterEnded;
126   }
127
128   public void setMaxRequestCountAfterEnded(int maxRequestCountAfterEnded) {
129     this.maxRequestCountAfterEnded = maxRequestCountAfterEnded;
130   }
131
132   public void newRequestReceived() {
133     if (ended) {
134       ++requestCountAfterEnded;
135     }
136   }
137
138   public long getLastAccessTime() {
139     return lastAccessTime;
140   }
141
142   public void setLastAccessTime(long lastAccessTime) {
143     this.lastAccessTime = lastAccessTime;
144   }
145
146   public void updateLastAccessTime() {
147     this.lastAccessTime = System.currentTimeMillis();
148   }
149
150   public int getMaxInactiveInterval() {
151     return maxInactiveInterval;
152   }
153
154   public void setMaxInactiveInterval(int maxInactiveInterval) {
155     this.maxInactiveInterval = maxInactiveInterval;
156   }
157
158   public boolean isNaturalId() {
159     return naturalId;
160   }
161
162 }
```

Listado A.15: Conversation.java

A.1.16. ConversationFactory.java

```
1  package com.hexaid.struts2.conversations;
2
3  import java.io.Serializable;
4
5  /**
6   * @author Gabriel Belingueres
7   *
8   */
9  public interface ConversationFactory extends Serializable {
10
11   /**
12    * Obtain a new synthetic Id for a conversation.
13    * @return the String with the newly generated id.
14    */
15   String getNextConversationId();
16
17   /**
18    * Create a new conversation with synthetic id
19    * @return the newly created conversation
20    */
21   Conversation createConversation();
22
23   /**
24    * Create a new conversation with a natural id.
25    * Note that the id is supplied to the conversation in
26    * a later phase of processing.
27    * @return the newly created conversation with a
28    * natural id.
29    */
30   Conversation createNaturalConversation();
31 }
```

Listado A.16: ConversationFactory.java

A.1.17. ActionMessages.java

```
1  package com.hexaid.struts2.conversations.impl;
2
3  import java.io.Serializable;
4  import java.util.Collection;
5
6  /**
7   * @author Gabriel Belingueres
8   *
9   */
10 public class ActionMessages implements Serializable {
11
12   private static final long serialVersionUID = 1L;
13
14   private Collection<String> messages;
```

```

  public void setActionMessages(Collection<String> messages) {
    this.messages = messages;
  }

  public Collection<String> getActionMessages() {
    return messages;
  }

  public boolean hasActionMessages() {
    return messages != null && !messages.isEmpty();
  }

  public String toString() {
    return "ActionMessages:" + messages.toString();
  }

}
```

Listado A.17: ActionMessages.java

A.1.18. DefaultConversationFactory.java

```
package com.hexaid.struts2.conversations.impl;

import java.util.concurrent.atomic.AtomicInteger;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.hexaid.struts2.conversations.Conversation;
import com.hexaid.struts2.conversations.ConversationFactory;
import com.opensymphony.xwork2.inject.Inject;

/**
 * @author Gabriel Belingueres
 *
 */
public class DefaultConversationFactory implements ConversationFactory {

  private static final long serialVersionUID = 1L;

  private static final Logger LOG = LoggerFactory.getLogger(DefaultConversationFactory.class);

  final private AtomicInteger conversationIdCounter = new AtomicInteger();

  /**
   * Injected value
   */
  private int maxInactiveIntervalConstant;

  @Override
  public String getNextConversationId() {
    return Integer.toString(conversationIdCounter.incrementAndGet());
  }

  @Override
  public Conversation createConversation() {
    final String id = getNextConversationId();
    LOG.debug("Creating new conversation id [{}]", id);
    final Conversation conversation = new Conversation(id);
    conversation.setMaxInactiveInterval(maxInactiveIntervalConstant);
    return conversation;
  }

  @Override
  public Conversation createNaturalConversation() {
    LOG.debug("Creating new natural conversation");
    final Conversation conversation = new Conversation(null, System.currentTimeMillis(), true);
    conversation.setMaxInactiveInterval(maxInactiveIntervalConstant);
    return conversation;
  }

  public int getMaxInactiveIntervalConstant() {
    return maxInactiveIntervalConstant;
  }

  @Inject(value = "com.hexaid.struts2.conversation.expiration.maxInactiveInterval")
  public void setMaxInactiveIntervalConstant(String maxInactiveIntervalConstantStr) {
    this.maxInactiveIntervalConstant = Integer.parseInt(maxInactiveIntervalConstantStr);
  }

}
```

Listado A.18: DefaultConversationFactory.java

A.1.19. ConversationExpirationPolicy.java

```
package com.hexaid.struts2.expiration;

import java.io.Serializable;
import java.util.concurrent.ConcurrentMap;

import com.hexaid.struts2.conversations.Conversation;

/**
 * @author Gabriel Belingueres
 *
 */
public interface ConversationExpirationPolicy extends Serializable {

  public void checkExpirations( final ConcurrentMap<String, Conversation> allConversationsMap,
                 final String conversationId,
                 final long currentTime);

  public boolean isExpired( final Conversation conversation,
                final Conversation requestedConversation,
                final long currentTime);

```

```
}
```

Listado A.19: ConversationExpirationPolicy.java

A.1.20. AbstractConversationExpirationPolicy.java

```
package com.hexaid.struts2.expiration.impl;

import java.util.Map.Entry;
import java.util.concurrent.ConcurrentMap;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.hexaid.struts2.conversations.Conversation;
import com.hexaid.struts2.expiration.ConversationExpirationPolicy;

/**
 * @author Gabriel Belingueres
 *
 */
public abstract class AbstractConversationExpirationPolicy implements
    ConversationExpirationPolicy {

  private static final long serialVersionUID = 1L;

  private static final Logger LOG = LoggerFactory.getLogger(AbstractConversationExpirationPolicy.class);

  /**
   * Parameter:
   * Time in seconds between requests before the framework will expire
   * conversations.
   * If non positive, it will NOT explicitly expire the conversations,
   * instead they will all timeout when the user Session does.
   */
  protected int maxInactiveInterval = 0;

  protected AbstractConversationExpirationPolicy(final String maxInactiveIntervalStr) {
    this.maxInactiveInterval = Integer.parseInt(maxInactiveIntervalStr);
    LOG.info("maxInactiveInterval : {} seconds ({} minutes)",
        maxInactiveInterval, (maxInactiveInterval / 60));
  }

  @Override
  public void checkExpirations( final ConcurrentMap<String, Conversation> allConversationsMap,
                final String conversationId,
                final long currentTime) {
    final Conversation requestedConversation = (conversationId == null) ? null : allConversationsMap.get(conversationId);
    for(final Entry<String, Conversation> entry : allConversationsMap.entrySet()) {
      final Conversation conversation = entry.getValue();
      if (isExpired(conversation, requestedConversation, currentTime)) {
        LOG.debug("Conversation id [{}] expired", conversation.getId());
        // thread-safe removal
        allConversationsMap.remove(conversation.getId(), conversation);
      }
    }
  }

  public int getMaxInactiveInterval() {
    return maxInactiveInterval;
  }

  public void setMaxInactiveInterval(int maxInactiveInterval) {
    this.maxInactiveInterval = maxInactiveInterval;
  }

}
```

Listado A.20: AbstractConversationExpirationPolicy.java

A.1.21. FixedTimeConversationExpirationPolicy.java

```
package com.hexaid.struts2.expiration.impl;

import com.hexaid.struts2.conversations.Conversation;
import com.opensymphony.xwork2.inject.Inject;

/**
 * @author Gabriel Belingueres
 *
 */
public class FixedTimeConversationExpirationPolicy extends AbstractConversationExpirationPolicy {

  private static final long serialVersionUID = 1L;

  @Inject
  public FixedTimeConversationExpirationPolicy( @Inject(value = "com.hexaid.struts2.conversation.expiration.
        maxInactiveInterval", required = false) String maxInactiveIntervalStr) {
    super(maxInactiveIntervalStr);
  }

  @Override
  public boolean isExpired( final Conversation conversation,
                final Conversation requestedConversation,
                final long currentTime) {
    if (maxInactiveInterval > 0) {
      return (conversation.getLastAccessTime() + (maxInactiveInterval*1000)) < currentTime;
    }
    else {
      // never expires => expires only when the Session expires
      return false;
    }
  }
}
```

Listado A.21: FixedTimeConversationExpirationPolicy.java

A.1.22. ForeBackConversationExpirationPolicy.java

```java
package com.hexaid.struts2.expiration.impl;

import com.hexaid.struts2.conversations.Conversation;
import com.opensymphony.xwork2.inject.Inject;

/**
 * @author Gabriel Belingueres
 *
 */
public class ForeBackConversationExpirationPolicy extends AbstractConversationExpirationPolicy {

  private static final long serialVersionUID = 1L;

  @Inject
  public ForeBackConversationExpirationPolicy(@Inject(value = "com.hexaid.struts2.conversation.expiration.maxInactiveInterval
      ", required = false) String maxInactiveIntervalStr) {
    super(maxInactiveIntervalStr);
  }

  public boolean isExpired( final Conversation conversation,
                final Conversation requestedConversation,
                final long currentTime) {
    final boolean isBackground = isBackgroundConversation(requestedConversation, conversation);
    if (isBackground && maxInactiveInterval > 0) {
      return (conversation.getLastAccessTime() + (maxInactiveInterval*1000)) < currentTime;
    }
    else {
      // the conversation parameter will not expire in this opportunity
      return false;
    }
  }

  protected boolean isBackgroundConversation( final Conversation requestedConversation, final Conversation conversation) {
    return requestedConversation != null
        && !requestedConversation.isEnded()
        && conversation != requestedConversation;
  }

}
```

Listado A.22: ForeBackConversationExpirationPolicy.java

A.1.23. PerViewConversationExpirationPolicy.java

```java
package com.hexaid.struts2.expiration.impl;

import com.hexaid.struts2.conversations.Conversation;
import com.opensymphony.xwork2.inject.Inject;

/**
 * @author Gabriel Belingueres
 *
 */
public class PerViewConversationExpirationPolicy extends
    ForeBackConversationExpirationPolicy {

  private static final long serialVersionUID = 1L;

  @Inject
  public PerViewConversationExpirationPolicy(@Inject(value = "com.hexaid.struts2.conversation.expiration.maxInactiveInterval"
      , required = false) String maxInactiveIntervalStr) {
    super(maxInactiveIntervalStr);
  }

  @Override
  public boolean isExpired( final Conversation conversation,
                final Conversation requestedConversation,
                final long currentTime) {
    final boolean isBackground = isBackgroundConversation(requestedConversation, conversation);
    if (isBackground && conversation.getMaxInactiveInterval() > 0) {
      return (conversation.getLastAccessTime() + (conversation.getMaxInactiveInterval()*1000)) < currentTime;
    }
    else {
      // never expires => expires only when the Session expires
      return false;
    }
  }

}
```

Listado A.23: PerViewConversationExpirationPolicy.java

A.1.24. BijectionInterceptor.java

```java
package com.hexaid.struts2.interceptor;

import com.hexaid.struts2.bijection.Bijector;
import com.opensymphony.xwork2.ActionInvocation;
import com.opensymphony.xwork2.inject.Inject;
import com.opensymphony.xwork2.interceptor.AbstractInterceptor;
import com.opensymphony.xwork2.interceptor.PreResultListener;

/**
 * @author Gabriel Belingueres
 *
 */
public class BijectionInterceptor extends AbstractInterceptor {

  private static final long serialVersionUID = 1L;

  private transient Bijector fieldBijector;
  private transient Bijector methodBijector;

  @Override
  public String intercept(ActionInvocation invocation) throws Exception {
    final Object action = invocation.getAction();
    final Class<? extends Object> actionClass = action.getClass();

    fieldBijector.doInjections(action, actionClass);
    methodBijector.doInjections(action, actionClass);
```

```
27
28      invocation.addPreResultListener(new PreResultListener() {
29        @Override
30        public void beforeResult(ActionInvocation invocation, String resultCode) {
31          // perform Outjection
32          fieldBijector.doOutjections(action, actionClass);
33          methodBijector.doOutjections(action, actionClass);
34        }
35      });
36
37      return invocation.invoke();
38    }
39
40    @Inject("field")
41    public void setFieldBijector(Bijector fieldBijector) {
42      this.fieldBijector = fieldBijector;
43    }
44
45    @Inject("method")
46    public void setMethodBijector(Bijector methodBijector) {
47      this.methodBijector = methodBijector;
48    }
49
50  }
```

Listado A.24: BijectionInterceptor.java

A.1.25. ConversationInterceptor.java

```
1  package com.hexaid.struts2.interceptor;
2
3  import static com.hexaid.struts2.common.ConversationAttributeType.MANDATORY;
4  import static com.hexaid.struts2.common.ConversationAttributeType.NONE;
5  import static com.hexaid.struts2.common.ConversationAttributeType.REQUIRED;
6  import static com.hexaid.struts2.common.ConversationAttributeType.REQUIRES_NEW;
7  import static com.hexaid.struts2.common.ConversationAttributeType.SUPPORTS;
8
9  import java.lang.reflect.Method;
10 import java.util.Map;
11 import java.util.concurrent.ConcurrentHashMap;
12 import java.util.concurrent.ConcurrentMap;
13
14 import javax.servlet.http.HttpServletRequest;
15 import javax.servlet.http.HttpSession;
16
17 import org.apache.commons.lang.StringUtils;
18 import org.apache.struts2.ServletActionContext;
19 import org.slf4j.Logger;
20 import org.slf4j.LoggerFactory;
21 import org.slf4j.helpers.MessageFormatter;
22
23 import com.hexaid.struts2.annotations.Begin;
24 import com.hexaid.struts2.annotations.ConversationAttribute;
25 import com.hexaid.struts2.annotations.End;
26 import com.hexaid.struts2.common.ConversationAttributeType;
27 import com.hexaid.struts2.conversations.Conversation;
28 import com.hexaid.struts2.conversations.ConversationFactory;
29 import com.hexaid.struts2.conversations.impl.ActionMessages;
30 import com.hexaid.struts2.expiration.ConversationExpirationPolicy;
31 import com.hexaid.struts2.persistence.PersistenceTransactionManager;
32 import com.opensymphony.xwork2.ActionInvocation;
33 import com.opensymphony.xwork2.ValidationAware;
34 import com.opensymphony.xwork2.inject.Container;
35 import com.opensymphony.xwork2.inject.Inject;
36 import com.opensymphony.xwork2.interceptor.AbstractInterceptor;
37 import com.opensymphony.xwork2.interceptor.PreResultListener;
38 import com.opensymphony.xwork2.util.ValueStack;
39
40 public class ConversationInterceptor extends AbstractInterceptor {
41
42   private static final long serialVersionUID = 1L;
43
44   private static final Logger LOG = LoggerFactory.getLogger(ConversationInterceptor.class);
45
46   static final String DEFAULT_CONVERSATION_NOT_FOUND_RESULT = "conversation_not_found";
47
48   private ConversationFactory conversationFactory;
49
50   private transient PersistenceTransactionManager persistenceTransactionManager;
51
52   private ConversationExpirationPolicy expirationPolicyManager;
53
54   private Container container;
55
56   /**
57    * Interceptor parameter:
58    * Specifies the PersistenceTransactionManager to be used. Default is "default".
59    */
60   private String persistence = Container.DEFAULT_NAME;
61
62     /**
63      * Interceptor parameter:
64      * Specifies the action result to use when no conversation is found.
65      * Default is the constant DEFAULT_CONVERSATION_NOT_FOUND_RESULT.
66      */
67     private String conversationNotFoundResult = DEFAULT_CONVERSATION_NOT_FOUND_RESULT;
68
69   /**
70    * Injected constant
71    * Specifies the conversation expiration policy.
72    */
73   private String expirationPolicy = Container.DEFAULT_NAME;
74
75   /**
76    * Injected constant
77    * Specifies the ConversationFactory to use. Default is "default".
78    */
79   private String conversationFactoryStr = Container.DEFAULT_NAME;
80
81   @Override
82   public String intercept(final ActionInvocation invocation) throws Exception {
83     final Object action = invocation.getAction();
84     final HttpServletRequest request = ServletActionContext.getRequest();
85
```

```
    final String methodName = resolveMethodName(invocation);

    final Method actionMethod = getActionMethod(action, methodName);

    final String conversationId = request.getParameter(Conversation.CONVERSATION_ID_PARAM);

    final ConversationAttributeType conversationAttr = getConversationAttribute(actionMethod, conversationId);

    if (conversationAttr == NONE) {
      // process without conversation support
      LOG.debug("Invoking action method {}() without support for conversation", methodName);
      return invocation.invoke();
    }

    LOG.debug(
        "Invoking action method {}() with support for conversation. ConversationAttributeType: {}",
        methodName, conversationAttr);

    final HttpSession session = request.getSession();

    // there are no 2 requests on the same session which can operate from now on
    // and since the invocation.invoke() is inside the synchronized block too,
    // the succeeding interceptors will be synchronized too (ie bijection and others)
    // TODO: Implement something like Spring's WebUtils and HttpSessionMutexListener
    synchronized(session) {

      // Is there a ALL_CONVERSATIONS_MAP Map in the session? If not put one
      // there.
      final ConcurrentMap<String, Conversation> allConversationsMap = createAllConversationsMapIfNecessary(session);

      // remove expired conversations
      expirationPolicyManager.checkExpirations(allConversationsMap, conversationId, System.currentTimeMillis());

      final Begin beginAnnotation = actionMethod.getAnnotation(Begin.class);
      final boolean hasNaturalIdExpression =
          beginAnnotation != null && !beginAnnotation.naturalIdExpression().isEmpty();

      final Conversation conversation =
          conversationAttr.process(conversationFactory, allConversationsMap, conversationId, hasNaturalIdExpression);

      if (conversationAttr == MANDATORY && (conversation == null || conversation.isEnded())) {
        // was mandatory and conversation was not found =>
        // result in conversationNotFoundResult without calling the action
        return conversationNotFoundResult;
      }

      try {
        // if it is new => initialize conversation and put it in the map
        if (conversation != null && conversation.isNew()) {
          persistenceTransactionManager.conversationStarting(conversation);

          initializeNewConversation(action, beginAnnotation, conversation, invocation.getStack());

          // save the conversation
          allConversationsMap.put(conversation.getId(), conversation);
        }
        else if (conversation != null) {
          // not a new conversation
          if (!conversation.isEnded()) {
            // update last accessed time when is not ended (and when not new)
            conversation.updateLastAccessTime();
          }
          persistenceTransactionManager.conversationResumed(conversation);
        }

        // sets the current conversation in request scope to allow custom tags to find
        // it when it just begins (because no conversationId parameter is present)
        request.setAttribute(Conversation.CURRENT_CONVERSATION_KEY, conversation);

                invocation.addPreResultListener(new PreResultListener() {
                    @Override
                    public void beforeResult(ActionInvocation invocation, String resultCode) {
                        if (conversation != null) {
                            if (action instanceof ValidationAware) {
                                // set action messages inside the conversation, if any, but ONLY
                                // if the conversation is NOT ended, meaning that after ending
                                // the conversation it can not longer CHANGE its contents (which is
                                // very sound too)
                                saveActionMessages((ValidationAware) action, conversation);
                            }

                            // After action invocation
                            conversation.setNew(false);

                            try {
                                endConversation(actionMethod, conversation, resultCode);
                            }
                            catch (RuntimeException e) {
                                // erases the result code returned by the action
                                invocation.setResultCode(null);
                                throw e;
                            }
                        }
                    }
                });

        // Invoke the next interceptor in the interceptor stack.
        return invocation.invoke();
      }
      catch(Exception e) {
        persistenceTransactionManager.exceptionThrown(conversation, e);
        throw e;
      }
      finally {
        removeMarkedForDeletionConversations(allConversationsMap);

        // forces update on clustered environments??
        session.setAttribute(Conversation.CONVERSATIONS_MAP_KEY, allConversationsMap);
      }
    } // end synchronized block on the HttpSession object
  }

  private void initializeNewConversation(final Object action,
                      final Begin beginAnnotation,
                      final Conversation conversation,
                      final ValueStack valueStack)
      throws Exception {
```

```
203     final String initializationMethod = beginAnnotation.initialization();
204     if (!initializationMethod.isEmpty()) {
205       final Method initMethod = getActionMethod(action, initializationMethod);
206       if (initMethod != null) {
207         // the initialization method was found
208         initMethod.invoke(action, new Object[]{});
209       }
210     }
211
212     final String naturalIdExpression = beginAnnotation.naturalIdExpression();
213     if (!naturalIdExpression.isEmpty()) {
214       // a natural conversation => initialize its id
215       final String naturalId = valueStack.findString(naturalIdExpression);
216       if (StringUtils.isEmpty(naturalId)) {
217         throw new IllegalArgumentException(
218             "Illegal natural conversation Id [" + naturalId
219                 + "] using expression ["
220                 + naturalIdExpression
221                 + "]");
222       }
223       LOG.debug(
224           "Assigning natural conversation Id [{}] to conversation {} using expression [{}]",
225           new Object[] {naturalId, conversation, naturalIdExpression});
226       conversation.setId(naturalId);
227     }
228   }
229
230   private ConcurrentMap<String, Conversation> createAllConversationsMapIfNecessary(final HttpSession session) {
231     @SuppressWarnings("unchecked")
232     ConcurrentMap<String, Conversation> allConversationsMap =
233         (ConcurrentMap<String, Conversation>) session.getAttribute(Conversation.CONVERSATIONS_MAP_KEY);
234     if (allConversationsMap == null) {
235       allConversationsMap = new ConcurrentHashMap<String, Conversation>();
236       session.setAttribute(Conversation.CONVERSATIONS_MAP_KEY, allConversationsMap);
237     }
238     return allConversationsMap;
239   }
240
241   private void saveActionMessages(final ValidationAware action, final Conversation conversation) {
242     ActionMessages messages =
243         (ActionMessages) conversation.getMap().get(Conversation.CURRENT_CONVERSATION_MESSAGES_KEY);
244     if (conversation.isEnded()) {
245       if (messages.hasActionMessages()) {
246         // copy messages to current action BEFORE rendering the page
247         for(final String msg : messages.getActionMessages()) {
248           action.addActionMessage(msg);
249         }
250       }
251     }
252     else {
253       // if NOT ended
254       if  (messages == null) {
255         // salva este objeto que tiene SOLO mensajes (para que no se "cuelen" datos de un request previo)
256         messages = new ActionMessages();
257       }
258
259       // whether or not they have messages (so previous messages stored in conversation are deleted)
260       messages.setActionMessages(action.getActionMessages());
261
262       LOG.debug("Putting action messages {} in conversation id [{}]", messages, conversation.getId());
263
264       conversation.getMap().put(Conversation.CURRENT_CONVERSATION_MESSAGES_KEY, messages);
265     }
266   }
267
268   private void removeMarkedForDeletionConversations(final Map<String, Conversation> allConversationsMap) {
269     for(final Conversation conversation : allConversationsMap.values()) {
270       if (conversation.isMarkedForDeletion()) {
271         LOG.debug("Removing conversation id [{}] from allConversationsMap", conversation.getId());
272         allConversationsMap.remove(conversation.getId());
273       }
274     }
275   }
276
277   private void endConversation(final Method actionMethod,
278                 final Conversation conversation,
279                 final String result) {
280     if (conversation.isEnded()) {
281       LOG.debug("The Ended conversation id [{}] has received a new request", conversation.getId());
282       conversation.newRequestReceived();
283     }
284     else {
285       // has an @End annotation?
286       final End endAnnotation = actionMethod.getAnnotation(End.class);
287
288       if (endAnnotation != null) {
289         if (endAnnotation.endResult().equals(result)) {
290           // may throw an exception on COMMIT of the transaction
291           persistenceTransactionManager.conversationEnding(conversation, endAnnotation.commit());
292
293           // end the conversation
294           LOG.debug("The conversation id [{}] has Ended ({})", conversation.getId(), endAnnotation);
295           conversation.end(endAnnotation.beforeRedirect());
296         }
297       }
298       else {
299         persistenceTransactionManager.conversationPaused(conversation);
300       }
301     }
302   }
303
304     private ConversationAttributeType getConversationAttribute(final Method actionMethod, final String conversationId) {
305     // si tiene anotacin @Begin, @End or @ConversationAttribute
306     final Begin begin = actionMethod.getAnnotation(Begin.class);
307     final End end = actionMethod.getAnnotation(End.class);
308     final ConversationAttribute conversationControl = actionMethod.getAnnotation(ConversationAttribute.class);
309
310     if (begin != null && conversationControl != null) {
311       // @Begin and @ConversationAttribute, with or without @End
312       final ConversationAttributeType conversationAttribute = conversationControl.value();
313       if (conversationAttribute == NONE || conversationAttribute == SUPPORTS || conversationAttribute == MANDATORY) {
314         final String message =
315             MessageFormatter.format("The action method {}() has declared the @Begin with @ConversationAttribute({})",
316                 actionMethod.getName(), conversationAttribute.toString()).getMessage();
317         throw new IllegalStateException(message);
318       }
319
```

```
320       return conversationAttribute;
321     }
322     else if (begin != null && end != null && conversationControl == null) {
323       // @Begin, @End without @ConversationAttribute
324       return REQUIRES_NEW;
325     }
326     else if (begin == null && end != null && conversationControl != null) {
327       // @End and @ConversationAttribute, without @Begin
328       final ConversationAttributeType conversationAttribute = conversationControl.value();
329       if (conversationAttribute != MANDATORY) {
330         final String message =
331             MessageFormatter.format("The action method {}() has declared the @End annotation, but with @ConversationAttribute
                  ({})",
332                 actionMethod.getName(), conversationAttribute.toString()).getMessage();
333         throw new IllegalStateException(message);
334       }
335
336       return MANDATORY;
337     }
338     else if (begin != null && end == null && conversationControl == null) {
339       // @Begin only
340       return REQUIRED;
341     }
342     else if (begin == null && end != null && conversationControl == null) {
343       // @End only
344       return MANDATORY;
345     }
346     else if (begin == null && end == null && conversationControl != null) {
347       // @ConversationAttribute only
348       return conversationControl.value();
349     }
350     else if (!StringUtils.isEmpty(conversationId)) {
351       // conversationId parameter is present ONLY => assume SUPPORTS
352       return SUPPORTS;
353     }
354
355     // anything else is NONE
356     return NONE;
357   }
358
359   private Method getActionMethod(final Object action, String methodName) throws NoSuchMethodException {
360     LOG.debug("clase: " + action.getClass().getName() + "  metodo: " + methodName);
361     return action.getClass().getMethod(methodName, new Class<?>[]{});
362   }
363
364   private static String resolveMethodName(final ActionInvocation invocation) {
365     String method = invocation.getProxy().getMethod();
366     if (method == null) {
367       return "execute";
368     }
369     return method;
370   }
371
372   @Override
373   public void init() {
374     LOG.info("=== Initializing ConversationInterceptor ===");
375     conversationFactory = container.getInstance(ConversationFactory.class, conversationFactoryStr);
376     LOG.info(
377         "Configured Conversation Factory \"{}\" with class: {}", conversationFactoryStr, conversationFactory.getClass().
                getName());
378
379     persistenceTransactionManager = container.getInstance(PersistenceTransactionManager.class, persistence);
380     LOG.info(
381         "Configured persistence transaction manager \"{}\" with class: {}",
382         persistence, persistenceTransactionManager.getClass().getName());
383     persistenceTransactionManager.init();
384
385     expirationPolicyManager = container.getInstance(ConversationExpirationPolicy.class, expirationPolicy);
386     LOG.info(
387         "Configured conversation expiration policy \"{}\" with class: {}",
388         expirationPolicy, expirationPolicyManager.getClass().getName());
389   }
390
391   @Override
392   public void destroy() {
393     conversationFactory = null;
394     persistenceTransactionManager = null;
395     expirationPolicyManager = null;
396     container = null;
397   }
398
399   public ConversationFactory getConversationFactory() {
400     return conversationFactory;
401   }
402
403   public void setConversationFactory(ConversationFactory conversationFactory) {
404     this.conversationFactory = conversationFactory;
405   }
406
407   public PersistenceTransactionManager getPersistenceTransactionManager() {
408     return persistenceTransactionManager;
409   }
410
411   public void setPersistenceTransactionManager( PersistenceTransactionManager persistenceTransactionManager) {
412     this.persistenceTransactionManager = persistenceTransactionManager;
413   }
414
415   public String getPersistence() {
416     return persistence;
417   }
418
419   public void setPersistence(String persistence) {
420     this.persistence = persistence;
421     LOG.debug("set Persistence to: {}", persistence);
422   }
423
424   @Inject
425   public void setContainer(Container container) {
426     this.container = container;
427   }
428
429   public String getExpirationPolicy() {
430     return expirationPolicy;
431   }
432
433   @Inject(value="com.hexaid.struts2.conversation.expiration.policy")
434   public void setExpirationPolicy(String expirationPolicy) {
```

```
435     this.expirationPolicy = expirationPolicy;
436   }
437
438   public ConversationExpirationPolicy getExpirationPolicyManager() {
439     return expirationPolicyManager;
440   }
441
442   public void setExpirationPolicyManager(ConversationExpirationPolicy expirationPolicyManager) {
443     this.expirationPolicyManager = expirationPolicyManager;
444   }
445
446   public String getConversationFactoryStr() {
447     return conversationFactoryStr;
448   }
449
450   @Inject(value="com.hexaid.struts2.conversation.factory")
451   public void setConversationFactoryStr(String conversationFactoryStr) {
452     this.conversationFactoryStr = conversationFactoryStr;
453   }
454
455     public String getConversationNotFoundResult() {
456         return conversationNotFoundResult;
457     }
458
459     public void setConversationNotFoundResult(String conversationNotFoundResult) {
460         this.conversationNotFoundResult = conversationNotFoundResult;
461     }
462
463 }
```

Listado A.25: ConversationInterceptor.java

A.1.26. JPASpringPersistenceTransactionManager.java

```
1  package com.hexaid.struts2.persistence.impl;
2
3  import javax.persistence.EntityManager;
4  import javax.persistence.EntityManagerFactory;
5  import javax.persistence.FlushModeType;
6
7  import org.slf4j.Logger;
8  import org.slf4j.LoggerFactory;
9  import org.springframework.orm.jpa.EntityManagerHolder;
10 import org.springframework.orm.jpa.JpaTransactionManager;
11 import org.springframework.transaction.TransactionStatus;
12 import org.springframework.transaction.support.TransactionCallbackWithoutResult;
13 import org.springframework.transaction.support.TransactionSynchronizationManager;
14 import org.springframework.transaction.support.TransactionTemplate;
15
16 import com.hexaid.struts2.conversations.Conversation;
17 import com.hexaid.struts2.persistence.PersistenceTransactionManager;
18 import com.opensymphony.xwork2.ObjectFactory;
19 import com.opensymphony.xwork2.inject.Inject;
20
21 /**
22  * Implements JPA specific code for Conversation Managed Persistence Contexts,
23  * where both JPA and Transactions are configured using Spring (requires using
24  * the Struts Spring plugin).
25  *
26  * @author Gabriel Belingueres
27  *
28  */
29 public class JPASpringPersistenceTransactionManager implements
30     PersistenceTransactionManager {
31
32   private final static Logger LOG = LoggerFactory.getLogger(JPASpringPersistenceTransactionManager.class);
33
34   public static final String PERSISTENCE_CONTEXT_KEY = "com.hexaid.struts2.conversations.persistence.context";
35
36   private EntityManagerFactory entityManagerFactory;
37   private TransactionTemplate transactionTemplate;
38
39   private final ObjectFactory objectFactory;
40
41   @Inject
42   public JPASpringPersistenceTransactionManager(@Inject ObjectFactory objectFactory) {
43     this.objectFactory = objectFactory;
44   }
45
46   @Override
47   public void init() {
48     try {
49       // inject the JPA Transaction Manager
50       final JpaTransactionManager transactionManager = (JpaTransactionManager) objectFactory.buildBean(JpaTransactionManager.
             class, null);
51
52       this.entityManagerFactory = transactionManager.getEntityManagerFactory();
53       this.transactionTemplate = new TransactionTemplate(transactionManager);
54     } catch (Exception e) {
55       throw new IllegalArgumentException("Could not find a Spring's JpaTransactionManager!", e);
56     }
57   }
58
59   @Override
60   public void conversationStarting(final Conversation conversation) {
61     final EntityManager em = entityManagerFactory.createEntityManager();
62     em.setFlushMode(FlushModeType.COMMIT);
63     putEntityManager(conversation, em);
64     bind(em);
65   }
66
67   @Override
68   public void conversationPaused(final Conversation conversation) {
69     final EntityManager em = getEntityManager(conversation);
70     LOG.debug("Conversation paused with entityManager: {}", em);
71     unbind(em);
72   }
73
74   @Override
75   public void conversationResumed(Conversation conversation) {
76     if (!conversation.isEnded()) {
77       final EntityManager em = getEntityManager(conversation);
78       LOG.debug("Conversation resumed with entityManager: {}", em);
79       bind(em);
```

```
80      }
81    }
82
83    @Override
84    public void conversationEnding(final Conversation conversation, final boolean commit) {
85      final EntityManager em = getEntityManager(conversation);
86      if (commit) {
87        LOG.debug("Committing transaction with entityManager: {}", em);
88        transactionTemplate.execute(new TransactionCallbackWithoutResult() {
89          protected void doInTransactionWithoutResult(TransactionStatus status) {
90            em.joinTransaction();
91          }
92        });
93      }
94      else {
95        LOG.debug("NOT Committing transaction with entityManager: {}", em);
96      }
97      unbind(em);
98      removeEntityManager(conversation, em);
99      em.close();
100   }
101
102   @Override
103   public void exceptionThrown(Conversation conversation, Exception exception) {
104     final EntityManager em = getEntityManager(conversation);
105     unbind(em);
106   }
107
108   protected void putEntityManager(final Conversation conversation, final EntityManager em) {
109     conversation.getMap().put(PERSISTENCE_CONTEXT_KEY, em);
110   }
111
112   protected EntityManager getEntityManager(final Conversation conversation) {
113     return (EntityManager) conversation.getMap().get(PERSISTENCE_CONTEXT_KEY);
114   }
115
116   protected void removeEntityManager(final Conversation conversation, final EntityManager em) {
117     conversation.getMap().remove(PERSISTENCE_CONTEXT_KEY);
118   }
119
120   protected void bind(final EntityManager em) {
121     LOG.debug("Binding entityManager: {}", em);
122     TransactionSynchronizationManager.bindResource(entityManagerFactory, new EntityManagerHolder(em));
123   }
124
125   protected void unbind(final EntityManager em) {
126     LOG.debug("Unbinding entityManager: {}", em);
127     if (TransactionSynchronizationManager.hasResource(entityManagerFactory)) {
128       TransactionSynchronizationManager.unbindResource(entityManagerFactory);
129     }
130   }
131
132   protected boolean isEntityManagerFactoryBound() {
133     return TransactionSynchronizationManager.hasResource(entityManagerFactory);
134   }
135
136 }
```

Listado A.26: JPASpringPersistenceTransactionManager.java

A.1.27. PersistenceTransactionManagerAdapter.java

```
1  package com.hexaid.struts2.persistence.impl;
2
3  import com.hexaid.struts2.conversations.Conversation;
4  import com.hexaid.struts2.persistence.PersistenceTransactionManager;
5
6  /**
7   * @author Gabriel Belingueres
8   *
9   */
10 public class PersistenceTransactionManagerAdapter implements
11     PersistenceTransactionManager {
12
13   @Override
14   public void init() {
15     // nothing
16   }
17
18   /* (non-Javadoc)
19    * @see com.hexaid.struts2.persistence.PersistenceTransactionManager#conversationStarting(com.hexaid.struts2.conversations.
          Conversation)
20    */
21   @Override
22   public void conversationStarting(Conversation conversation) {
23     // nothing
24   }
25
26   /* (non-Javadoc)
27    * @see com.hexaid.struts2.persistence.PersistenceTransactionManager#conversationPaused(com.hexaid.struts2.conversations.
          Conversation)
28    */
29   @Override
30   public void conversationPaused(Conversation conversation) {
31     // nothing
32   }
33
34   /* (non-Javadoc)
35    * @see com.hexaid.struts2.persistence.PersistenceTransactionManager#conversationResumed(com.hexaid.struts2.conversations.
          Conversation)
36    */
37   @Override
38   public void conversationResumed(Conversation conversation) {
39     // nothing
40   }
41
42   /* (non-Javadoc)
43    * @see com.hexaid.struts2.persistence.PersistenceTransactionManager#conversationEnding(com.hexaid.struts2.conversations.
          Conversation, boolean)
44    */
45   @Override
46   public void conversationEnding(Conversation conversation, boolean commit) {
47     // nothing
48   }
```

```
49
50   /* (non-Javadoc)
51    * @see com.hexaid.struts2.persistence.PersistenceTransactionManager#exceptionThrown(com.hexaid.struts2.conversations.
          Conversation, java.lang.Exception)
52    */
53   @Override
54   public void exceptionThrown(Conversation conversation, Exception exception) {
55     // nothing
56   }
57
58 }
```

Listado A.27: PersistenceTransactionManagerAdapter.java

A.1.28. PersistenceTransactionManager.java

```
1  package com.hexaid.struts2.persistence;
2
3  import com.hexaid.struts2.conversations.Conversation;
4
5  /**
6   * @author Gabriel Belingueres
7   *
8   */
9  public interface PersistenceTransactionManager {
10
11   public void init();
12
13   public void conversationStarting(final Conversation conversation);
14
15   public void conversationPaused(final Conversation conversation);
16
17   public void conversationResumed(final Conversation conversation);
18
19   public void conversationEnding(final Conversation conversation, final boolean commit);
20
21   public void exceptionThrown(final Conversation conversation, final Exception exception);
22
23 }
```

Listado A.28: PersistenceTransactionManager.java

A.1.29. ServletActionRedirectConversationResult.java

```
1  package com.hexaid.struts2.result;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6  import org.apache.struts2.dispatcher.ServletActionRedirectResult;
7
8  import com.hexaid.struts2.common.ConversationUtils;
9  import com.hexaid.struts2.conversations.Conversation;
10 import com.opensymphony.xwork2.ActionInvocation;
11
12 /**
13  * @author Gabriel Belingueres
14  *
15  */
16 public class ServletActionRedirectConversationResult extends
17     ServletActionRedirectResult {
18
19   private static final long serialVersionUID = 1L;
20
21   /**
22    * default propagation is "join"
23    */
24   private String conversationPropagation = Conversation.CONVERSATION_PROPAGATION_JOIN;
25
26   /**
27    * Result parameter that changes the conversation timeout to the specified
28    * value (measured in Seconds). (If the expiration policy is not "perview"
29    * the parameter will be ignored).
30    */
31   private int maxInactiveInterval;
32
33   public ServletActionRedirectConversationResult() {
34     super();
35   }
36
37   public ServletActionRedirectConversationResult(String namespace, String actionName, String method, String anchor) {
38     super(namespace, actionName, method, anchor);
39   }
40
41   public ServletActionRedirectConversationResult(String namespace, String actionName, String method) {
42     super(namespace, actionName, method);
43   }
44
45   public ServletActionRedirectConversationResult(String actionName, String method) {
46     super(actionName, method);
47   }
48
49   public ServletActionRedirectConversationResult(String actionName) {
50     super(actionName);
51   }
52
53   @Override
54   public void execute(ActionInvocation invocation) throws Exception {
55     if (Conversation.CONVERSATION_PROPAGATION_JOIN.equalsIgnoreCase(conversationPropagation)) {
56       final Conversation currentConversation = ConversationUtils.getCurrentConversation();
57       if (currentConversation != null) {
58         addParameter(Conversation.CONVERSATION_ID_PARAM, currentConversation.getId());
59       }
60     }
61     else if (!Conversation.CONVERSATION_PROPAGATION_NONE.equalsIgnoreCase(conversationPropagation)) {
62       throw new IllegalArgumentException("Unrecognized conversationPropagation. Valid values are: join, none");
63     }
64
65     super.execute(invocation);
66
```

```
67      if (getMaxInactiveInterval() > 0) {
68        ConversationUtils.changeMaxInactiveInterval(getMaxInactiveInterval());
69      }
70    }
71
72    public void setConversationPropagation(String conversationPropagation) {
73      this.conversationPropagation = conversationPropagation;
74    }
75
76    @Override
77    protected List<String> getProhibitedResultParams() {
78      final List<String> prohibitedResultParams = super.getProhibitedResultParams();
79      List<String> newList = new ArrayList<String>(prohibitedResultParams);
80      newList.add("conversationPropagation");
81      return newList;
82    }
83
84    public int getMaxInactiveInterval() {
85      return maxInactiveInterval;
86    }
87
88    public void setMaxInactiveInterval(int maxInactiveInterval) {
89      this.maxInactiveInterval = maxInactiveInterval;
90    }
91
92  }
```

Listado A.29: ServletActionRedirectConversationResult.java

A.1.30. ServletDispatcherConversationResult.java

```
1  package com.hexaid.struts2.result;
2
3  import org.apache.struts2.dispatcher.ServletDispatcherResult;
4
5  import com.hexaid.struts2.common.ConversationUtils;
6  import com.opensymphony.xwork2.ActionInvocation;
7
8  /**
9   * @author Gabriel Belingueres
10  *
11  */
12 public class ServletDispatcherConversationResult extends
13      ServletDispatcherResult {
14
15    private static final long serialVersionUID = 1L;
16
17    /**
18     * Result parameter that changes the conversation timeout to the specified
19     * value (measured in Seconds). (If the expiration policy is not "perview"
20     * the parameter will be ignored).
21     */
22    private int maxInactiveInterval;
23
24    public ServletDispatcherConversationResult() {
25      super();
26    }
27
28    public ServletDispatcherConversationResult(String location) {
29      super(location);
30    }
31
32    @Override
33    public void execute(ActionInvocation invocation) throws Exception {
34      if (maxInactiveInterval > 0) {
35        ConversationUtils.changeMaxInactiveInterval(maxInactiveInterval);
36      }
37
38      super.execute(invocation);
39    }
40
41    public int getMaxInactiveInterval() {
42      return maxInactiveInterval;
43    }
44
45    public void setMaxInactiveInterval(int maxInactiveInterval) {
46      this.maxInactiveInterval = maxInactiveInterval;
47    }
48
49 }
```

Listado A.30: ServletDispatcherConversationResult.java

A.1.31. ServletRedirectConversationResult.java

```
1  package com.hexaid.struts2.result;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6  import org.apache.struts2.dispatcher.ServletRedirectResult;
7
8  import com.hexaid.struts2.common.ConversationUtils;
9  import com.hexaid.struts2.conversations.Conversation;
10 import com.opensymphony.xwork2.ActionInvocation;
11
12 /**
13  * @author Gabriel Belingueres
14  *
15  */
16 public class ServletRedirectConversationResult extends ServletRedirectResult {
17
18    private static final long serialVersionUID = 1L;
19
20    /**
21     * default propagation is "join"
22     */
23    private String conversationPropagation = Conversation.CONVERSATION_PROPAGATION_JOIN;
24
25    /**
```

```
26    * Result parameter that changes the conversation timeout to the specified
27    * value (measured in Seconds). (If the expiration policy is not "perview"
28    * the parameter will be ignored).
29    */
30   private String maxInactiveInterval;
31
32   public ServletRedirectConversationResult() {
33   }
34
35   public ServletRedirectConversationResult(String location) {
36     super(location);
37   }
38
39   public ServletRedirectConversationResult(String location, String anchor) {
40     super(location, anchor);
41   }
42
43   public ServletRedirectConversationResult(String location, String anchor, String maxInactiveInterval) {
44     super(location, anchor);
45     this.maxInactiveInterval = maxInactiveInterval;
46   }
47
48   @Override
49   public void execute(ActionInvocation invocation) throws Exception {
50     if (Conversation.CONVERSATION_PROPAGATION_JOIN.equalsIgnoreCase(conversationPropagation)) {
51       final Conversation currentConversation = ConversationUtils.getCurrentConversation();
52       if (currentConversation != null) {
53         addParameter(Conversation.CONVERSATION_ID_PARAM, currentConversation.getId());
54       }
55     }
56     else if (!Conversation.CONVERSATION_PROPAGATION_NONE.equalsIgnoreCase(conversationPropagation)) {
57       throw new IllegalArgumentException("Unrecognized conversationPropagation. Valid values are: join, none");
58     }
59
60     super.execute(invocation);
61
62     int timeout = Integer.parseInt(maxInactiveInterval);
63     if (timeout > 0) {
64       ConversationUtils.changeMaxInactiveInterval(timeout);
65     }
66   }
67
68   public void setConversationPropagation(String conversationPropagation) {
69     this.conversationPropagation = conversationPropagation;
70   }
71
72   @Override
73   protected List<String> getProhibitedResultParams() {
74     final List<String> prohibitedResultParams = super.getProhibitedResultParams();
75     List<String> newList = new ArrayList<String>(prohibitedResultParams);
76     newList.add("conversationPropagation");
77     newList.add("maxInactiveInterval");
78     return newList;
79   }
80
81   public String getMaxInactiveInterval() {
82     return maxInactiveInterval;
83   }
84
85   public void setMaxInactiveInterval(String maxInactiveInterval) {
86     this.maxInactiveInterval = maxInactiveInterval;
87   }
88
89 }
```

Listado A.31: ServletRedirectConversationResult.java

A.1.32. TilesConversationResult.java

```
1  package com.hexaid.struts2.result;
2
3  import org.apache.struts2.views.tiles.TilesResult;
4
5  import com.hexaid.struts2.common.ConversationUtils;
6  import com.opensymphony.xwork2.ActionInvocation;
7
8  /**
9   * @author Gabriel Belingueres
10  *
11  */
12 public class TilesConversationResult extends TilesResult {
13
14   private static final long serialVersionUID = 1L;
15
16   /**
17    * Result parameter that changes the conversation timeout to the specified
18    * value (measured in Seconds). (If the expiration policy is not "perview"
19    * the parameter will be ignored).
20    */
21   private int maxInactiveInterval;
22
23   public TilesConversationResult() {
24     super();
25   }
26
27   public TilesConversationResult(String location) {
28     super(location);
29   }
30
31   @Override
32   public void execute(ActionInvocation invocation) throws Exception {
33     if (maxInactiveInterval > 0) {
34       ConversationUtils.changeMaxInactiveInterval(maxInactiveInterval);
35     }
36
37     super.execute(invocation);
38   }
39
40   public int getMaxInactiveInterval() {
41     return maxInactiveInterval;
42   }
43
44   public void setMaxInactiveInterval(int maxInactiveInterval) {
45     this.maxInactiveInterval = maxInactiveInterval;
```

```
46   }
47
48
49 }
```

Listado A.32: TilesConversationResult.java

A.1.33. ConversationPropagationTag.java

```
1  package com.hexaid.struts2.ui.jsp;
2
3  import java.io.IOException;
4
5  import javax.servlet.http.HttpServletRequest;
6  import javax.servlet.jsp.JspException;
7  import javax.servlet.jsp.PageContext;
8  import javax.servlet.jsp.tagext.SimpleTagSupport;
9
10 import com.hexaid.struts2.common.ConversationUtils;
11 import com.hexaid.struts2.conversations.Conversation;
12
13 /**
14  * @author Gabriel Belingueres
15  *
16  */
17 public class ConversationPropagationTag extends SimpleTagSupport {
18
19   private String mode;
20
21   @Override
22   public void doTag() throws JspException, IOException {
23     if (Conversation.CONVERSATION_PROPAGATION_JOIN.equalsIgnoreCase(mode)) {
24       final PageContext jspContext = (PageContext) getJspContext();
25
26       final HttpServletRequest request = (HttpServletRequest) jspContext.getRequest();
27       final Conversation conversation = ConversationUtils.getCurrentConversation(request);
28
29       if (conversation != null) {
30         jspContext.getOut()
31           .append("<input type=\"hidden\" name=\"")
32           .append(Conversation.CONVERSATION_ID_PARAM)
33           .append("\" value=\"")
34           .append(conversation.getId())
35           .append("\"/>\n");
36       }
37     }
38   }
39
40   public String getMode() {
41     return mode;
42   }
43
44   public void setMode(String mode) {
45     this.mode = mode;
46   }
47
48 }
```

Listado A.33: ConversationPropagationTag.java

A.1.34. FormConversationTag.java

```
1  package com.hexaid.struts2.ui.jsp;
2
3  import org.apache.struts2.components.Component;
4  import org.slf4j.Logger;
5  import org.slf4j.LoggerFactory;
6
7  import com.hexaid.struts2.common.ConversationUtils;
8  import com.hexaid.struts2.conversations.Conversation;
9
10 /**
11  * @author Gabriel Belingueres
12  *
13  */
14 public class FormConversationTag extends org.apache.struts2.views.jsp.ui.FormTag {
15
16   private static final long serialVersionUID = 1L;
17
18   private static final Logger LOG = LoggerFactory.getLogger(FormConversationTag.class);
19
20   /**
21    * defaults to "join"
22    */
23   private String conversationPropagation = Conversation.CONVERSATION_PROPAGATION_JOIN;
24
25
26   @Override
27   protected void populateParams() {
28     super.populateParams();
29
30     final Component form = getComponent();
31
32         final String convPropagValue = findString(conversationPropagation);
33
34     if (Conversation.CONVERSATION_PROPAGATION_JOIN.equalsIgnoreCase(convPropagValue)) {
35       final Conversation currentConversation = ConversationUtils.getCurrentConversation();
36       if (currentConversation != null) {
37         form.addParameter("conversationPropagation", convPropagValue);
38         form.addParameter("CONVERSATION_ID_PARAM", Conversation.CONVERSATION_ID_PARAM);
39         form.addParameter(Conversation.CONVERSATION_ID_PARAM, currentConversation.getId());
40       }
41     }
42     else if (!Conversation.CONVERSATION_PROPAGATION_NONE.equalsIgnoreCase(convPropagValue)) {
43       LOG.warn("Unrecognized conversationPropagation attribute value '{}' in URL tag", convPropagValue);
44     }
45   }
46
47   public void setConversationPropagation(String conversationPropagation) {
48     this.conversationPropagation = conversationPropagation;
```

```
  }
}
```

Listado A.34: FormConversationTag.java

A.1.35. URLConversationTag.java

```
package com.hexaid.struts2.ui.jsp;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.hexaid.struts2.common.ConversationUtils;
import com.hexaid.struts2.conversations.Conversation;

/**
 * @author Gabriel Belingueres
 *
 */
public class URLConversationTag extends org.apache.struts2.views.jsp.URLTag {

  private static final long serialVersionUID = 1L;

  private static final Logger LOG = LoggerFactory.getLogger(URLConversationTag.class);

  /**
   * defaults to "join"
   */
  private String conversationPropagation = Conversation.CONVERSATION_PROPAGATION_JOIN;

  @Override
  protected void populateParams() {
    super.populateParams();

    final String convPropagValue = findString(conversationPropagation);

    if (Conversation.CONVERSATION_PROPAGATION_JOIN.equalsIgnoreCase(convPropagValue)) {
      final Conversation currentConversation = ConversationUtils.getCurrentConversation();
      if (currentConversation != null) {
        getComponent().addParameter(Conversation.CONVERSATION_ID_PARAM, currentConversation.getId());
      }
    }
    else if (!Conversation.CONVERSATION_PROPAGATION_NONE.equalsIgnoreCase(convPropagValue)) {
      LOG.warn("Unrecognized conversationPropagation attribute value '{}' in URL tag", convPropagValue);
    }
  }

  public void setConversationPropagation(String conversationPropagation) {
    this.conversationPropagation = conversationPropagation;
  }

}
```

Listado A.35: URLConversationTag.java

A.1.36. UseConversationTag.java

```
package com.hexaid.struts2.ui.jsp;

import java.io.IOException;
import java.util.Map;
import java.util.Map.Entry;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.tagext.SimpleTagSupport;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.hexaid.struts2.common.ConversationUtils;
import com.hexaid.struts2.conversations.Conversation;

/**
 * @author Gabriel Belingueres
 *
 */
public class UseConversationTag extends SimpleTagSupport {

  final static private Logger LOG = LoggerFactory.getLogger(UseConversationTag.class);

  /* (non-Javadoc)
   * @see javax.servlet.jsp.tagext.SimpleTagSupport#doTag()
   */
  @Override
  public void doTag() throws JspException, IOException {
    final PageContext jspContext = (PageContext) getJspContext();

    final HttpServletRequest request = (HttpServletRequest) jspContext.getRequest();
    final Conversation conversation = ConversationUtils.getCurrentConversation(request);

    if (conversation != null) {
      putConversationAttributesInPageContext(jspContext, conversation);
    }
  }

  private void putConversationAttributesInPageContext(final PageContext jspContext,
                              final Conversation conversation) {
    final Map<String, Object> internalConversationMap = conversation.getMap();
    for(final Entry<String, Object> entry : internalConversationMap.entrySet()) {
      final String key = entry.getKey();
      final Object value = entry.getValue();
      // si encuentra un valor con el mismo nombre emite un WARNING
      if (jspContext.findAttribute(key) != null) {
        LOG.warn(
            "Page attribute '{}' in current page will be shadowed by same variable name from conversation id=[{}]",
            key, conversation.getId());
      }
```

```
53
54        LOG.debug("adding to pageContext {}={}", key, value);
55
56        jspContext.setAttribute(key, value, PageContext.PAGE_SCOPE);
57      }
58    }
59
60  }
```

Listado A.36: UseConversationTag.java

A.1.37. struts2-conversation-tags.tld

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <taglib xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd"
4    version="2.0">
5    <description><![CDATA[Custom tag library for using conversations with Struts 2]]></description>
6    <display-name>"Conversation Struts 2 Tags"</display-name>
7    <tlib-version>2.2</tlib-version>
8    <short-name>sconv</short-name>
9    <uri>/conversation-struts2-tags</uri>
10   <tag>
11     <description><![CDATA[Set current conversation's attributes as page-scoped attributes]]></description>
12     <name>useConversation</name>
13     <tag-class>com.hexaid.struts2.ui.jsp.UseConversationTag</tag-class>
14     <body-content>empty</body-content>
15     <dynamic-attributes>false</dynamic-attributes>
16   </tag>
17   <tag>
18     <description><![CDATA[Set current conversation's as a hidden form parameter]]></description>
19     <name>conversationPropagation</name>
20     <tag-class>com.hexaid.struts2.ui.jsp.ConversationPropagationTag</tag-class>
21     <body-content>empty</body-content>
22     <attribute>
23       <description><![CDATA[The mode: Use 'join' for continuing the current conversation]]></description>
24       <name>mode</name>
25       <required>true</required>
26       <rtexprvalue>false</rtexprvalue>
27       <type>java.lang.String</type>
28     </attribute>
29     <dynamic-attributes>false</dynamic-attributes>
30   </tag>
31   <tag>
32     <description><![CDATA[This tag is used to create Conversation aware a URL]]></description>
33     <name>url</name>
34     <tag-class>com.hexaid.struts2.ui.jsp.URLConversationTag</tag-class>
35     <body-content>JSP</body-content>
36     <!-- conversation related attributes -->
37     <attribute>
38       <description><![CDATA[The mode: Use 'join' for continuing the current conversation. Defaults to 'join']]></description>
39       <name>conversationPropagation</name>
40       <required>false</required>
41       <rtexprvalue>false</rtexprvalue>
42     </attribute>
43     <!-- original Struts 2 URL tag attributes -->
44     <attribute>
45       <description><![CDATA[The action to generate the URL for, if not using value]]></description>
46       <name>action</name>
47       <required>false</required>
48       <rtexprvalue>false</rtexprvalue>
49     </attribute>
50     <attribute>
51       <description><![CDATA[The anchor for this URL]]></description>
52       <name>anchor</name>
53       <required>false</required>
54       <rtexprvalue>false</rtexprvalue>
55     </attribute>
56     <attribute>
57       <description><![CDATA[Whether to encode parameters]]></description>
58       <name>encode</name>
59       <required>false</required>
60       <rtexprvalue>false</rtexprvalue>
61     </attribute>
62     <attribute>
63       <description><![CDATA[Specifies whether to escape ampersand (&amp;) to (&amp;amp;) or not]]></description>
64       <name>escapeAmp</name>
65       <required>false</required>
66       <rtexprvalue>false</rtexprvalue>
67     </attribute>
68     <attribute>
69       <description><![CDATA[Specifies whether to force the addition of scheme, host and port or not]]></description>
70       <name>forceAddSchemeHostAndPort</name>
71       <required>false</required>
72       <rtexprvalue>false</rtexprvalue>
73     </attribute>
74     <attribute>
75       <description><![CDATA[Deprecated. Use 'var' instead]]></description>
76       <name>id</name>
77       <required>false</required>
78       <rtexprvalue>false</rtexprvalue>
79     </attribute>
80     <attribute>
81       <description><![CDATA[Whether actual context should be included in URL]]></description>
82       <name>includeContext</name>
83       <required>false</required>
84       <rtexprvalue>false</rtexprvalue>
85     </attribute>
86     <attribute>
87       <description><![CDATA[The includeParams attribute may have the value 'none', 'get' or 'all']]></description>
88       <name>includeParams</name>
89       <required>false</required>
90       <rtexprvalue>false</rtexprvalue>
91     </attribute>
92     <attribute>
93       <description><![CDATA[The method of action to use]]></description>
94       <name>method</name>
95       <required>false</required>
96       <rtexprvalue>false</rtexprvalue>
97     </attribute>
98     <attribute>
99       <description><![CDATA[The namespace to use]]></description>
100      <name>namespace</name>
101      <required>false</required>
```

```
102       <rtexprvalue>false</rtexprvalue>
103     </attribute>
104     <attribute>
105       <description><![CDATA[The resulting portlet mode]]></description>
106       <name>portletMode</name>
107       <required>false</required>
108       <rtexprvalue>false</rtexprvalue>
109     </attribute>
110     <attribute>
111       <description><![CDATA[Specifies if this should be a portlet render or action URL. Default is "render". To create an
              action URL, use "action".]]></description>
112       <name>portletUrlType</name>
113       <required>false</required>
114       <rtexprvalue>false</rtexprvalue>
115     </attribute>
116     <attribute>
117       <description><![CDATA[Set scheme attribute]]></description>
118       <name>scheme</name>
119       <required>false</required>
120       <rtexprvalue>false</rtexprvalue>
121     </attribute>
122     <attribute>
123       <description><![CDATA[The target value to use, if not using action]]></description>
124       <name>value</name>
125       <required>false</required>
126       <rtexprvalue>false</rtexprvalue>
127     </attribute>
128     <attribute>
129       <description><![CDATA[Name used to reference the value pushed into the Value Stack]]></description>
130       <name>var</name>
131       <required>false</required>
132       <rtexprvalue>false</rtexprvalue>
133     </attribute>
134     <attribute>
135       <description><![CDATA[The resulting portlet window state]]></description>
136       <name>windowState</name>
137       <required>false</required>
138       <rtexprvalue>false</rtexprvalue>
139     </attribute>
140     <dynamic-attributes>false</dynamic-attributes>
141   </tag>
142   <tag>
143     <description><![CDATA[Renders an input form with conversation support]]></description>
144     <name>form</name>
145     <tag-class>com.hexaid.struts2.ui.jsp.FormConversationTag</tag-class>
146     <body-content>JSP</body-content>
147     <!-- conversation related attributes -->
148     <attribute>
149       <description><![CDATA[The mode: Use 'join' for continuing the current conversation. Defaults to 'join']]></description>
150       <name>conversationPropagation</name>
151       <required>false</required>
152       <rtexprvalue>false</rtexprvalue>
153     </attribute>
154     <!-- original Struts 2 URL tag attributes -->
155     <attribute>
156       <description><![CDATA[The accepted charsets for this form. The values may be comma or blank delimited.]]></description>
157       <name>acceptcharset</name>
158       <required>false</required>
159       <rtexprvalue>false</rtexprvalue>
160     </attribute>
161     <attribute>
162       <description><![CDATA[Set the html accesskey attribute on rendered html element]]></description>
163       <name>accesskey</name>
164       <required>false</required>
165       <rtexprvalue>false</rtexprvalue>
166     </attribute>
167     <attribute>
168       <description><![CDATA[Set action name to submit to, without .action suffix]]></description>
169       <name>action</name>
170       <required>false</required>
171       <rtexprvalue>false</rtexprvalue>
172     </attribute>
173     <attribute>
174       <description><![CDATA[The css class to use for element]]></description>
175       <name>cssClass</name>
176       <required>false</required>
177       <rtexprvalue>false</rtexprvalue>
178     </attribute>
179     <attribute>
180       <description><![CDATA[The css error class to use for element]]></description>
181       <name>cssErrorClass</name>
182       <required>false</required>
183       <rtexprvalue>false</rtexprvalue>
184     </attribute>
185     <attribute>
186       <description><![CDATA[The css error style definitions for element to use]]></description>
187       <name>cssErrorStyle</name>
188       <required>false</required>
189       <rtexprvalue>false</rtexprvalue>
190     </attribute>
191     <attribute>
192       <description><![CDATA[The css style definitions for element to use]]></description>
193       <name>cssStyle</name>
194       <required>false</required>
195       <rtexprvalue>false</rtexprvalue>
196     </attribute>
197     <attribute>
198       <description><![CDATA[Set the html disabled attribute on rendered html element]]></description>
199       <name>disabled</name>
200       <required>false</required>
201       <rtexprvalue>false</rtexprvalue>
202     </attribute>
203     <attribute>
204       <description><![CDATA[HTML form enctype attribute]]></description>
205       <name>enctype</name>
206       <required>false</required>
207       <rtexprvalue>false</rtexprvalue>
208     </attribute>
209     <attribute>
210       <description><![CDATA[Id of element that will receive the focus when page loads.]]></description>
211       <name>focusElement</name>
212       <required>false</required>
213       <rtexprvalue>false</rtexprvalue>
214     </attribute>
215     <attribute>
216       <description><![CDATA[HTML id attribute]]></description>
217       <name>id</name>
```

```
      <required>false</required>
      <rtexprvalue>false</rtexprvalue>
    </attribute>
    <attribute>
      <description><![CDATA[Whether actual context should be included in URL]]></description>
      <name>includeContext</name>
      <required>false</required>
      <rtexprvalue>false</rtexprvalue>
    </attribute>
    <attribute>
      <description><![CDATA[Use JavaScript to generate tooltips]]></description>
      <name>javascriptTooltip</name>
      <required>false</required>
      <rtexprvalue>false</rtexprvalue>
    </attribute>
    <attribute>
      <description><![CDATA[Set the key (name, value, label) for this particular component]]></description>
      <name>key</name>
      <required>false</required>
      <rtexprvalue>false</rtexprvalue>
    </attribute>
    <attribute>
      <description><![CDATA[Label expression used for rendering an element specific label]]></description>
      <name>label</name>
      <required>false</required>
      <rtexprvalue>false</rtexprvalue>
    </attribute>
    <attribute>
      <description><![CDATA[String that will be appended to the label]]></description>
      <name>labelSeparator</name>
      <required>false</required>
      <rtexprvalue>false</rtexprvalue>
    </attribute>
    <attribute>
      <description><![CDATA[Define label position of form element (top/left)]]></description>
      <name>labelposition</name>
      <required>false</required>
      <rtexprvalue>false</rtexprvalue>
    </attribute>
    <attribute>
      <description><![CDATA[HTML form method attribute]]></description>
      <name>method</name>
      <required>false</required>
      <rtexprvalue>false</rtexprvalue>
    </attribute>
    <attribute>
      <description><![CDATA[The name to set for element]]></description>
      <name>name</name>
      <required>false</required>
      <rtexprvalue>false</rtexprvalue>
    </attribute>
    <attribute>
      <description><![CDATA[Namespace for action to submit to]]></description>
      <name>namespace</name>
      <required>false</required>
      <rtexprvalue>false</rtexprvalue>
    </attribute>
    <attribute>
      <description><![CDATA[ Set the html onblur attribute on rendered html element]]></description>
      <name>onblur</name>
      <required>false</required>
      <rtexprvalue>false</rtexprvalue>
    </attribute>
    <attribute>
      <description><![CDATA[Set the html onchange attribute on rendered html element]]></description>
      <name>onchange</name>
      <required>false</required>
      <rtexprvalue>false</rtexprvalue>
    </attribute>
    <attribute>
      <description><![CDATA[Set the html onclick attribute on rendered html element]]></description>
      <name>onclick</name>
      <required>false</required>
      <rtexprvalue>false</rtexprvalue>
    </attribute>
    <attribute>
      <description><![CDATA[Set the html ondblclick attribute on rendered html element]]></description>
      <name>ondblclick</name>
      <required>false</required>
      <rtexprvalue>false</rtexprvalue>
    </attribute>
    <attribute>
      <description><![CDATA[Set the html onfocus attribute on rendered html element]]></description>
      <name>onfocus</name>
      <required>false</required>
      <rtexprvalue>false</rtexprvalue>
    </attribute>
    <attribute>
      <description><![CDATA[Set the html onkeydown attribute on rendered html element]]></description>
      <name>onkeydown</name>
      <required>false</required>
      <rtexprvalue>false</rtexprvalue>
    </attribute>
    <attribute>
      <description><![CDATA[Set the html onkeypress attribute on rendered html element]]></description>
      <name>onkeypress</name>
      <required>false</required>
      <rtexprvalue>false</rtexprvalue>
    </attribute>
    <attribute>
      <description><![CDATA[Set the html onkeyup attribute on rendered html element]]></description>
      <name>onkeyup</name>
      <required>false</required>
      <rtexprvalue>false</rtexprvalue>
    </attribute>
    <attribute>
      <description><![CDATA[Set the html onmousedown attribute on rendered html element]]></description>
      <name>onmousedown</name>
      <required>false</required>
      <rtexprvalue>false</rtexprvalue>
    </attribute>
    <attribute>
      <description><![CDATA[Set the html onmousemove attribute on rendered html element]]></description>
      <name>onmousemove</name>
      <required>false</required>
      <rtexprvalue>false</rtexprvalue>
    </attribute>
```

```
335     <attribute>
336       <description><![CDATA[Set the html onmouseout attribute on rendered html element]]></description>
337       <name>onmouseout</name>
338       <required>false</required>
339       <rtexprvalue>false</rtexprvalue>
340     </attribute>
341     <attribute>
342       <description><![CDATA[Set the html onmouseover attribute on rendered html element]]></description>
343       <name>onmouseover</name>
344       <required>false</required>
345       <rtexprvalue>false</rtexprvalue>
346     </attribute>
347     <attribute>
348       <description><![CDATA[Set the html onmouseup attribute on rendered html element]]></description>
349       <name>onmouseup</name>
350       <required>false</required>
351       <rtexprvalue>false</rtexprvalue>
352     </attribute>
353     <attribute>
354       <description><![CDATA[HTML onreset attribute]]></description>
355       <name>onreset</name>
356       <required>false</required>
357       <rtexprvalue>false</rtexprvalue>
358     </attribute>
359     <attribute>
360       <description><![CDATA[Set the html onselect attribute on rendered html element]]></description>
361       <name>onselect</name>
362       <required>false</required>
363       <rtexprvalue>false</rtexprvalue>
364     </attribute>
365     <attribute>
366       <description><![CDATA[HTML onsubmit attribute]]></description>
367       <name>onsubmit</name>
368       <required>false</required>
369       <rtexprvalue>false</rtexprvalue>
370     </attribute>
371     <attribute>
372       <description><![CDATA[Set template to use for opening the rendered html.]]></description>
373       <name>openTemplate</name>
374       <required>false</required>
375       <rtexprvalue>false</rtexprvalue>
376     </attribute>
377     <attribute>
378       <description><![CDATA[The portlet mode to display after the form submit]]></description>
379       <name>portletMode</name>
380       <required>false</required>
381       <rtexprvalue>false</rtexprvalue>
382     </attribute>
383     <attribute>
384       <description><![CDATA[If set to true, the rendered element will indicate that input is required]]></description>
385       <name>required</name>
386       <required>false</required>
387       <rtexprvalue>false</rtexprvalue>
388     </attribute>
389     <attribute>
390       <description><![CDATA[Define required position of required form element (left|right)]]></description>
391       <name>requiredposition</name>
392       <required>false</required>
393       <rtexprvalue>false</rtexprvalue>
394     </attribute>
395     <attribute>
396       <description><![CDATA[Set the html tabindex attribute on rendered html element]]></description>
397       <name>tabindex</name>
398       <required>false</required>
399       <rtexprvalue>false</rtexprvalue>
400     </attribute>
401     <attribute>
402       <description><![CDATA[HTML form target attribute]]></description>
403       <name>target</name>
404       <required>false</required>
405       <rtexprvalue>false</rtexprvalue>
406     </attribute>
407     <attribute>
408       <description><![CDATA[The template (other than default) to use for rendering the element]]></description>
409       <name>template</name>
410       <required>false</required>
411       <rtexprvalue>false</rtexprvalue>
412     </attribute>
413     <attribute>
414       <description><![CDATA[The template directory.]]></description>
415       <name>templateDir</name>
416       <required>false</required>
417       <rtexprvalue>false</rtexprvalue>
418     </attribute>
419     <attribute>
420       <description><![CDATA[The theme (other than default) to use for rendering the element]]></description>
421       <name>theme</name>
422       <required>false</required>
423       <rtexprvalue>false</rtexprvalue>
424     </attribute>
425     <attribute>
426       <description><![CDATA[Set the html title attribute on rendered html element]]></description>
427       <name>title</name>
428       <required>false</required>
429       <rtexprvalue>false</rtexprvalue>
430     </attribute>
431     <attribute>
432       <description><![CDATA[Set the tooltip of this particular component]]></description>
433       <name>tooltip</name>
434       <required>false</required>
435       <rtexprvalue>false</rtexprvalue>
436     </attribute>
437     <attribute>
438       <description><![CDATA[Deprecated. Use individual tooltip configuration attributes instead.]]></description>
439       <name>tooltipConfig</name>
440       <required>false</required>
441       <rtexprvalue>false</rtexprvalue>
442     </attribute>
443     <attribute>
444       <description><![CDATA[CSS class applied to JavaScrip tooltips]]></description>
445       <name>tooltipCssClass</name>
446       <required>false</required>
447       <rtexprvalue>false</rtexprvalue>
448     </attribute>
449     <attribute>
450       <description><![CDATA[Delay in milliseconds, before showing JavaScript tooltips ]]></description>
451       <name>tooltipDelay</name>
```

```
452       <required>false</required>
453       <rtexprvalue>false</rtexprvalue>
454     </attribute>
455     <attribute>
456       <description><![CDATA[Icon path used for image that will have the tooltip]]></description>
457       <name>tooltipIconPath</name>
458       <required>false</required>
459       <rtexprvalue>false</rtexprvalue>
460     </attribute>
461     <attribute>
462       <description><![CDATA[Whether client side/remote validation should be performed. Only useful with theme xhtml/ajax]]></
            description>
463       <name>validate</name>
464       <required>false</required>
465       <rtexprvalue>false</rtexprvalue>
466     </attribute>
467     <attribute>
468       <description><![CDATA[Preset the value of input element.]]></description>
469       <name>value</name>
470       <required>false</required>
471       <rtexprvalue>false</rtexprvalue>
472     </attribute>
473     <attribute>
474       <description><![CDATA[The window state to display after the form submit]]></description>
475       <name>windowState</name>
476       <required>false</required>
477       <rtexprvalue>false</rtexprvalue>
478     </attribute>
479     <dynamic-attributes>true</dynamic-attributes>
480   </tag>
481   <tag>
482     <description><![CDATA[Render a custom ui widget]]></description>
483     <name>component</name>
484     <tag-class>org.apache.struts2.views.jsp.ui.ComponentTag</tag-class>
485     <body-content>JSP</body-content>
486     <attribute>
487       <description><![CDATA[Set the html accesskey attribute on rendered html element]]></description>
488       <name>accesskey</name>
489       <required>false</required>
490       <rtexprvalue>false</rtexprvalue>
491     </attribute>
492     <attribute>
493       <description><![CDATA[The css class to use for element]]></description>
494       <name>cssClass</name>
495       <required>false</required>
496       <rtexprvalue>false</rtexprvalue>
497     </attribute>
498     <attribute>
499       <description><![CDATA[The css error class to use for element]]></description>
500       <name>cssErrorClass</name>
501       <required>false</required>
502       <rtexprvalue>false</rtexprvalue>
503     </attribute>
504     <attribute>
505       <description><![CDATA[The css error style definitions for element to use]]></description>
506       <name>cssErrorStyle</name>
507       <required>false</required>
508       <rtexprvalue>false</rtexprvalue>
509     </attribute>
510     <attribute>
511       <description><![CDATA[The css style definitions for element to use]]></description>
512       <name>cssStyle</name>
513       <required>false</required>
514       <rtexprvalue>false</rtexprvalue>
515     </attribute>
516     <attribute>
517       <description><![CDATA[Set the html disabled attribute on rendered html element]]></description>
518       <name>disabled</name>
519       <required>false</required>
520       <rtexprvalue>false</rtexprvalue>
521     </attribute>
522     <attribute>
523       <description><![CDATA[HTML id attribute]]></description>
524       <name>id</name>
525       <required>false</required>
526       <rtexprvalue>false</rtexprvalue>
527     </attribute>
528     <attribute>
529       <description><![CDATA[Use JavaScript to generate tooltips]]></description>
530       <name>javascriptTooltip</name>
531       <required>false</required>
532       <rtexprvalue>false</rtexprvalue>
533     </attribute>
534     <attribute>
535       <description><![CDATA[Set the key (name, value, label) for this particular component]]></description>
536       <name>key</name>
537       <required>false</required>
538       <rtexprvalue>false</rtexprvalue>
539     </attribute>
540     <attribute>
541       <description><![CDATA[Label expression used for rendering an element specific label]]></description>
542       <name>label</name>
543       <required>false</required>
544       <rtexprvalue>false</rtexprvalue>
545     </attribute>
546     <attribute>
547       <description><![CDATA[String that will be appended to the label]]></description>
548       <name>labelSeparator</name>
549       <required>false</required>
550       <rtexprvalue>false</rtexprvalue>
551     </attribute>
552     <attribute>
553       <description><![CDATA[Define label position of form element (top/left)]]></description>
554       <name>labelposition</name>
555       <required>false</required>
556       <rtexprvalue>false</rtexprvalue>
557     </attribute>
558     <attribute>
559       <description><![CDATA[The name to set for element]]></description>
560       <name>name</name>
561       <required>false</required>
562       <rtexprvalue>false</rtexprvalue>
563     </attribute>
564     <attribute>
565       <description><![CDATA[ Set the html onblur attribute on rendered html element]]></description>
566       <name>onblur</name>
567       <required>false</required>
```

```
568       <rtexprvalue>false</rtexprvalue>
569     </attribute>
570     <attribute>
571       <description><![CDATA[Set the html onchange attribute on rendered html element]]></description>
572       <name>onchange</name>
573       <required>false</required>
574       <rtexprvalue>false</rtexprvalue>
575     </attribute>
576     <attribute>
577       <description><![CDATA[Set the html onclick attribute on rendered html element]]></description>
578       <name>onclick</name>
579       <required>false</required>
580       <rtexprvalue>false</rtexprvalue>
581     </attribute>
582     <attribute>
583       <description><![CDATA[Set the html ondblclick attribute on rendered html element]]></description>
584       <name>ondblclick</name>
585       <required>false</required>
586       <rtexprvalue>false</rtexprvalue>
587     </attribute>
588     <attribute>
589       <description><![CDATA[Set the html onfocus attribute on rendered html element]]></description>
590       <name>onfocus</name>
591       <required>false</required>
592       <rtexprvalue>false</rtexprvalue>
593     </attribute>
594     <attribute>
595       <description><![CDATA[Set the html onkeydown attribute on rendered html element]]></description>
596       <name>onkeydown</name>
597       <required>false</required>
598       <rtexprvalue>false</rtexprvalue>
599     </attribute>
600     <attribute>
601       <description><![CDATA[Set the html onkeypress attribute on rendered html element]]></description>
602       <name>onkeypress</name>
603       <required>false</required>
604       <rtexprvalue>false</rtexprvalue>
605     </attribute>
606     <attribute>
607       <description><![CDATA[Set the html onkeyup attribute on rendered html element]]></description>
608       <name>onkeyup</name>
609       <required>false</required>
610       <rtexprvalue>false</rtexprvalue>
611     </attribute>
612     <attribute>
613       <description><![CDATA[Set the html onmousedown attribute on rendered html element]]></description>
614       <name>onmousedown</name>
615       <required>false</required>
616       <rtexprvalue>false</rtexprvalue>
617     </attribute>
618     <attribute>
619       <description><![CDATA[Set the html onmousemove attribute on rendered html element]]></description>
620       <name>onmousemove</name>
621       <required>false</required>
622       <rtexprvalue>false</rtexprvalue>
623     </attribute>
624     <attribute>
625       <description><![CDATA[Set the html onmouseout attribute on rendered html element]]></description>
626       <name>onmouseout</name>
627       <required>false</required>
628       <rtexprvalue>false</rtexprvalue>
629     </attribute>
630     <attribute>
631       <description><![CDATA[Set the html onmouseover attribute on rendered html element]]></description>
632       <name>onmouseover</name>
633       <required>false</required>
634       <rtexprvalue>false</rtexprvalue>
635     </attribute>
636     <attribute>
637       <description><![CDATA[Set the html onmouseup attribute on rendered html element]]></description>
638       <name>onmouseup</name>
639       <required>false</required>
640       <rtexprvalue>false</rtexprvalue>
641     </attribute>
642     <attribute>
643       <description><![CDATA[Set the html onselect attribute on rendered html element]]></description>
644       <name>onselect</name>
645       <required>false</required>
646       <rtexprvalue>false</rtexprvalue>
647     </attribute>
648     <attribute>
649       <description><![CDATA[If set to true, the rendered element will indicate that input is required]]></description>
650       <name>required</name>
651       <required>false</required>
652       <rtexprvalue>false</rtexprvalue>
653     </attribute>
654     <attribute>
655       <description><![CDATA[Define required position of required form element (left|right)]]></description>
656       <name>requiredposition</name>
657       <required>false</required>
658       <rtexprvalue>false</rtexprvalue>
659     </attribute>
660     <attribute>
661       <description><![CDATA[Set the html tabindex attribute on rendered html element]]></description>
662       <name>tabindex</name>
663       <required>false</required>
664       <rtexprvalue>false</rtexprvalue>
665     </attribute>
666     <attribute>
667       <description><![CDATA[The template (other than default) to use for rendering the element]]></description>
668       <name>template</name>
669       <required>false</required>
670       <rtexprvalue>false</rtexprvalue>
671     </attribute>
672     <attribute>
673       <description><![CDATA[The template directory.]]></description>
674       <name>templateDir</name>
675       <required>false</required>
676       <rtexprvalue>false</rtexprvalue>
677     </attribute>
678     <attribute>
679       <description><![CDATA[The theme (other than default) to use for rendering the element]]></description>
680       <name>theme</name>
681       <required>false</required>
682       <rtexprvalue>false</rtexprvalue>
683     </attribute>
684     <attribute>
```

```
      <description><![CDATA[Set the html title attribute on rendered html element]]></description>
      <name>title</name>
      <required>false</required>
      <rtexprvalue>false</rtexprvalue>
    </attribute>
    <attribute>
      <description><![CDATA[Set the tooltip of this particular component]]></description>
      <name>tooltip</name>
      <required>false</required>
      <rtexprvalue>false</rtexprvalue>
    </attribute>
    <attribute>
      <description><![CDATA[Deprecated. Use individual tooltip configuration attributes instead.]]></description>
      <name>tooltipConfig</name>
      <required>false</required>
      <rtexprvalue>false</rtexprvalue>
    </attribute>
    <attribute>
      <description><![CDATA[CSS class applied to JavaScrip tooltips]]></description>
      <name>tooltipCssClass</name>
      <required>false</required>
      <rtexprvalue>false</rtexprvalue>
    </attribute>
    <attribute>
      <description><![CDATA[Delay in milliseconds, before showing JavaScript tooltips ]]></description>
      <name>tooltipDelay</name>
      <required>false</required>
      <rtexprvalue>false</rtexprvalue>
    </attribute>
    <attribute>
      <description><![CDATA[Icon path used for image that will have the tooltip]]></description>
      <name>tooltipIconPath</name>
      <required>false</required>
      <rtexprvalue>false</rtexprvalue>
    </attribute>
    <attribute>
      <description><![CDATA[Preset the value of input element.]]></description>
      <name>value</name>
      <required>false</required>
      <rtexprvalue>false</rtexprvalue>
    </attribute>
    <dynamic-attributes>false</dynamic-attributes>
  </tag>
</taglib>
```

Listado A.37: struts2-conversation-tags.tld

A.1.38. struts-plugin.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 2.1//EN" "http://struts.apache.org/dtds/
      struts-2.1.dtd" >
<struts>
  <bean type="com.hexaid.struts2.conversations.ConversationFactory" name="default"
    class="com.hexaid.struts2.conversations.impl.DefaultConversationFactory" />

  <!-- Standard Persistent Transaction Managers -->
  <bean type="com.hexaid.struts2.persistence.PersistenceTransactionManager" name="jpa-spring"
    class="com.hexaid.struts2.persistence.impl.JPASpringPersistenceTransactionManager" />
  <bean type="com.hexaid.struts2.persistence.PersistenceTransactionManager" name="default"
    class="com.hexaid.struts2.persistence.impl.PersistenceTransactionManagerAdapter" />

  <!-- Standard Conversation Expiration Policies -->
  <bean type="com.hexaid.struts2.expiration.ConversationExpirationPolicy" name="fixedtime"
    class="com.hexaid.struts2.expiration.impl.FixedTimeConversationExpirationPolicy" />
  <bean type="com.hexaid.struts2.expiration.ConversationExpirationPolicy" name="foreback"
    class="com.hexaid.struts2.expiration.impl.ForeBackConversationExpirationPolicy" />
  <bean type="com.hexaid.struts2.expiration.ConversationExpirationPolicy" name="perview"
    class="com.hexaid.struts2.expiration.impl.PerViewConversationExpirationPolicy" />

  <!-- Standard bijectors -->
  <bean type="com.hexaid.struts2.bijection.Bijector" name="field"
    class="com.hexaid.struts2.bijection.FieldBijector" />
  <bean type="com.hexaid.struts2.bijection.Bijector" name="method"
    class="com.hexaid.struts2.bijection.MethodBijector" />

  <!-- default conversation factory -->
  <constant name="com.hexaid.struts2.conversation.factory" value="default" />

  <!-- default conversation expiration policy -->
  <constant name="com.hexaid.struts2.conversation.expiration.policy" value="foreback" />

  <!-- default conversation timeout of 300 seconds (5 minutes) -->
  <constant name="com.hexaid.struts2.conversation.expiration.maxInactiveInterval" value="300" />

  <package name="conversationPackage" extends="struts-default">

    <!-- Redefine Conversation-aware result types -->
    <result-types>
      <result-type name="dispatcher" class="com.hexaid.struts2.result.ServletDispatcherConversationResult" default="true" />
      <result-type name="redirect" class="com.hexaid.struts2.result.ServletRedirectConversationResult" />
      <result-type name="redirectAction" class="com.hexaid.struts2.result.ServletActionRedirectConversationResult" />
      <!-- Declare it in your application if you are using the struts-tiles-plugin -->
      <!--
      <result-type name="tiles" class="com.hexaid.struts2.result.TilesConversationResult" />
      -->
    </result-types>

    <interceptors>
      <interceptor name="conversation"
        class="com.hexaid.struts2.interceptor.ConversationInterceptor" />
      <interceptor name="bijection"
        class="com.hexaid.struts2.interceptor.BijectionInterceptor" />

      <!--
      Generic interceptor stack supporting conversations and bijection.
      It is a paramsPrepareParamsStack with the addition of conversation and
      bijection interceptors BEFORE validation and BEFORE params
      -->
      <interceptor-stack name="abstractConversationSupportStack">
        <interceptor-ref name="exception"/>
        <interceptor-ref name="alias"/>
        <interceptor-ref name="i18n"/>
        <interceptor-ref name="checkbox"/>
```

```
65         <interceptor-ref name="multiselect"/>
66         <interceptor-ref name="params">
67             <param name="excludeParams">dojo\..*,^struts\..*</param>
68         </interceptor-ref>
69         <interceptor-ref name="servletConfig"/>
70         <interceptor-ref name="prepare"/>
71         <interceptor-ref name="chain"/>
72         <interceptor-ref name="modelDriven"/>
73         <interceptor-ref name="fileUpload"/>
74         <interceptor-ref name="staticParams"/>
75         <interceptor-ref name="actionMappingParams"/>
76
77         <interceptor-ref name="conversation" />
78         <interceptor-ref name="bijection" />
79
80         <interceptor-ref name="params">
81             <param name="excludeParams">dojo\..*,^struts\..*</param>
82         </interceptor-ref>
83         <interceptor-ref name="conversionError"/>
84         <interceptor-ref name="validation">
85             <param name="excludeMethods">input,back,cancel,browse</param>
86         </interceptor-ref>
87         <interceptor-ref name="workflow">
88             <param name="excludeMethods">input,back,cancel,browse</param>
89         </interceptor-ref>
90       </interceptor-stack>
91     </interceptors>
92
93     <default-interceptor-ref name="abstractConversationSupportStack" />
94   </package>
95 </struts>
```

Listado A.38: struts-plugin.xml

A.1.39. form.ftl

```
1 <#--
2 /*
3  * $Id: form.ftl 720258 2008-11-24 19:05:16Z musachy $
4  *
5  * Licensed to the Apache Software Foundation (ASF) under one
6  * or more contributor license agreements.  See the NOTICE file
7  * distributed with this work for additional information
8  * regarding copyright ownership.  The ASF licenses this file
9  * to you under the Apache License, Version 2.0 (the
10 * "License"); you may not use this file except in compliance
11 * with the License.  You may obtain a copy of the License at
12 *
13 *  http://www.apache.org/licenses/LICENSE-2.0
14 *
15 * Unless required by applicable law or agreed to in writing,
16 * software distributed under the License is distributed on an
17 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
18 * KIND, either express or implied.  See the License for the
19 * specific language governing permissions and limitations
20 * under the License.
21 */
22 -->
23 <#include "/${parameters.templateDir}/simple/form-common.ftl" />
24 <#if parameters.onreset??>
25  onreset="${parameters.onreset?html}"<#rt/>
26 </#if>
27 >
28 <#if parameters.conversationPropagation?? && parameters.conversationPropagation == "join">
29     <#assign tmpCurrentConversationInputName = parameters.CONVERSATION_ID_PARAM />
30     <input type="hidden" name="${tmpCurrentConversationInputName}" value="${parameters[tmpCurrentConversationInputName]}"/><#
          rt/>
31 </#if>
```

Listado A.39: form.ftl

A.1.40. theme.properties

```
1 #
2 # $Id: theme.properties 651946 2008-04-27 13:41:38Z apetrelli $
3 #
4 # Licensed to the Apache Software Foundation (ASF) under one
5 # or more contributor license agreements.  See the NOTICE file
6 # distributed with this work for additional information
7 # regarding copyright ownership.  The ASF licenses this file
8 # to you under the Apache License, Version 2.0 (the
9 # "License"); you may not use this file except in compliance
10 # with the License.  You may obtain a copy of the License at
11 #
12 #  http://www.apache.org/licenses/LICENSE-2.0
13 #
14 # Unless required by applicable law or agreed to in writing,
15 # software distributed under the License is distributed on an
16 # "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
17 # KIND, either express or implied.  See the License for the
18 # specific language governing permissions and limitations
19 # under the License.
20 #
21 parent = simple
```

Listado A.40: theme.properties

A.2. Reserva de hoteles

A.2.1. pom.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5   <groupId>com.hexaid.examples</groupId>
6   <artifactId>hotel</artifactId>
7   <version>1.0-SNAPSHOT</version>
8   <packaging>war</packaging>
9   <name>hotel</name>
10  <description>Example application showing the use of conversations</description>
11  <developers>
12    <developer>
13      <name>Gabriel Belingueres</name>
14    </developer>
15  </developers>
16  <inceptionYear>2011</inceptionYear>
17
18  <scm>
19    <connection>scm:svn:https://riouxsvn.com/svn/conversations/trunk/proyectos/hotel</connection>
20    <developerConnection>scm:svn:https://riouxsvn.com/svn/conversations/trunk/proyectos/hotel</developerConnection>
21  </scm>
22
23  <properties>
24    <struts2.version>2.2.3.1</struts2.version>
25    <spring.version>3.0.6.RELEASE</spring.version>
26    <spring.security.version>3.0.7.RELEASE</spring.security.version>
27    <slf4j.version>1.6.6</slf4j.version>
28    <project.build.sourceEncoding>Cp1252</project.build.sourceEncoding>
29
30    <!-- Valores para el maven-compiler-plugin -->
31    <maven.compiler.source>1.6</maven.compiler.source>
32    <maven.compiler.target>1.6</maven.compiler.target>
33    <maven.compiler.showWarnings>true</maven.compiler.showWarnings>
34    <maven.compiler.showDeprecation>true</maven.compiler.showDeprecation>
35    <maven.compiler.debuglevel>lines,vars,source</maven.compiler.debuglevel>
36    <maven.compiler.verbose>true</maven.compiler.verbose>
37  </properties>
38
39  <dependencies>
40
41    <dependency>
42      <groupId>org.apache.struts</groupId>
43      <artifactId>struts2-core</artifactId>
44      <version>${struts2.version}</version>
45    </dependency>
46
47    <dependency>
48      <groupId>org.apache.struts.xwork</groupId>
49      <artifactId>xwork-core</artifactId>
50      <version>${struts2.version}</version>
51    </dependency>
52
53    <dependency>
54      <groupId>org.apache.struts</groupId>
55      <artifactId>struts2-spring-plugin</artifactId>
56      <version>${struts2.version}</version>
57      <scope>runtime</scope>
58    </dependency>
59
60    <dependency>
61      <groupId>org.apache.struts</groupId>
62      <artifactId>struts2-config-browser-plugin</artifactId>
63      <version>${struts2.version}</version>
64      <scope>runtime</scope>
65    </dependency>
66
67    <dependency>
68      <groupId>org.apache.struts</groupId>
69      <artifactId>struts2-tiles-plugin</artifactId>
70      <version>${struts2.version}</version>
71      <scope>runtime</scope>
72      <exclusions>
73        <exclusion>
74          <groupId>org.apache.tiles</groupId>
75          <artifactId>tiles-core</artifactId>
76        </exclusion>
77        <exclusion>
78          <groupId>org.apache.tiles</groupId>
79          <artifactId>tiles-api</artifactId>
80        </exclusion>
81        <exclusion>
82          <groupId>org.apache.tiles</groupId>
83          <artifactId>tiles-jsp</artifactId>
84        </exclusion>
85      </exclusions>
86    </dependency>
87
88    <dependency>
89      <groupId>org.apache.tiles</groupId>
90      <artifactId>tiles-jsp</artifactId>
91      <version>2.2.2</version>
92      <scope>runtime</scope>
93    </dependency>
94
95    <dependency>
96      <groupId>displaytag</groupId>
97      <artifactId>displaytag</artifactId>
98      <version>1.2</version>
99      <scope>runtime</scope>
100     <exclusions>
101       <exclusion>
102         <groupId>com.lowagie</groupId>
103         <artifactId>itext</artifactId>
104       </exclusion>
105       <exclusion>
106         <groupId>org.slf4j</groupId>
107         <artifactId>jcl104-over-slf4j</artifactId>
108       </exclusion>
109       <exclusion>
110         <artifactId>slf4j-log4j12</artifactId>
111         <groupId>org.slf4j</groupId>
112       </exclusion>
113     </exclusions>
114   </dependency>
115
116   <dependency>
117     <groupId>javax.servlet</groupId>
118     <artifactId>jstl</artifactId>
119     <version>1.2</version>
```

```
      <scope>runtime</scope>
    </dependency>

    <dependency>
      <groupId>commons-lang</groupId>
      <artifactId>commons-lang</artifactId>
      <version>2.6</version>
    </dependency>

    <dependency>
      <groupId>commons-beanutils</groupId>
      <artifactId>commons-beanutils</artifactId>
      <version>1.8.3</version>
      <scope>runtime</scope>
      <exclusions>
        <exclusion>
          <artifactId>commons-logging</artifactId>
          <groupId>commons-logging</groupId>
        </exclusion>
      </exclusions>
    </dependency>

    <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-api</artifactId>
      <version>${slf4j.version}</version>
    </dependency>
    <dependency>
      <groupId>ch.qos.logback</groupId>
      <artifactId>logback-classic</artifactId>
      <version>1.0.6</version>
      <scope>runtime</scope>
    </dependency>
    <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>jcl-over-slf4j</artifactId>
      <version>${slf4j.version}</version>
      <scope>runtime</scope>
    </dependency>

    <dependency>
      <groupId>org.springframework.security</groupId>
      <artifactId>spring-security-web</artifactId>
      <version>${spring.security.version}</version>
      <scope>runtime</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework.security</groupId>
      <artifactId>spring-security-config</artifactId>
      <version>${spring.security.version}</version>
      <scope>runtime</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework.security</groupId>
      <artifactId>spring-security-taglibs</artifactId>
      <version>${spring.security.version}</version>
      <scope>runtime</scope>
    </dependency>

    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-core</artifactId>
      <version>${spring.version}</version>
      <scope>runtime</scope>
      <exclusions>
        <exclusion>
          <artifactId>commons-logging</artifactId>
          <groupId>commons-logging</groupId>
        </exclusion>
      </exclusions>
    </dependency>

    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-beans</artifactId>
      <version>${spring.version}</version>
    </dependency>

    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>${spring.version}</version>
    </dependency>

    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-web</artifactId>
      <version>${spring.version}</version>
      <scope>runtime</scope>
    </dependency>

    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-aop</artifactId>
      <version>${spring.version}</version>
      <scope>runtime</scope>
    </dependency>

    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-jdbc</artifactId>
      <version>${spring.version}</version>
      <scope>runtime</scope>
    </dependency>

    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-orm</artifactId>
      <version>${spring.version}</version>
      <scope>runtime</scope>
    </dependency>


    <dependency>
      <groupId>com.hexaid.struts2.conversations</groupId>
      <artifactId>struts2-conversations</artifactId>
      <version>1.0-SNAPSHOT</version>
```

```
237     </dependency>
238
239     <dependency>
240       <groupId>org.hibernate</groupId>
241       <artifactId>hibernate-entitymanager</artifactId>
242       <version>3.6.8.Final</version>
243       <scope>runtime</scope>
244     </dependency>
245
246     <dependency>
247       <groupId>org.hibernate.javax.persistence</groupId>
248       <artifactId>hibernate-jpa-2.0-api</artifactId>
249       <version>1.0.1.Final</version>
250     </dependency>
251
252     <dependency>
253       <groupId>org.hibernate</groupId>
254       <artifactId>hibernate-validator</artifactId>
255       <version>4.2.0.Final</version>
256     </dependency>
257
258     <dependency>
259       <groupId>org.hsqldb</groupId>
260       <artifactId>hsqldb</artifactId>
261       <version>2.2.8</version>
262       <classifier>jdk5</classifier>
263       <scope>runtime</scope>
264     </dependency>
265
266     <dependency>
267       <groupId>javax.servlet</groupId>
268       <artifactId>servlet-api</artifactId>
269       <version>2.5</version>
270       <scope>provided</scope>
271     </dependency>
272
273     <dependency>
274       <groupId>javax.servlet.jsp</groupId>
275       <artifactId>jsp-api</artifactId>
276       <version>2.1-rev-1</version>
277       <scope>provided</scope>
278     </dependency>
279
280     <dependency>
281       <groupId>org.apache.struts</groupId>
282       <artifactId>struts2-junit-plugin</artifactId>
283       <version>${struts2.version}</version>
284       <scope>test</scope>
285     </dependency>
286
287     <dependency>
288       <groupId>junit</groupId>
289       <artifactId>junit</artifactId>
290       <version>4.10</version>
291       <scope>test</scope>
292     </dependency>
293
294     <dependency>
295       <groupId>org.springframework</groupId>
296       <artifactId>spring-test</artifactId>
297       <version>${spring.version}</version>
298       <scope>test</scope>
299     </dependency>
300
301   </dependencies>
302
303   <build>
304     <defaultGoal>test</defaultGoal>
305     <finalName>${project.artifactId}</finalName>
306     <plugins>
307       <plugin>
308         <groupId>org.apache.maven.plugins</groupId>
309         <artifactId>maven-compiler-plugin</artifactId>
310         <version>2.5.1</version>
311         <configuration>
312           <source>${maven.compiler.source}</source>
313           <target>${maven.compiler.target}</target>
314           <showWarnings>${maven.compiler.showWarnings}</showWarnings>
315           <showDeprecation>${maven.compiler.showDeprecation}</showDeprecation>
316           <debuglevel>${maven.compiler.debuglevel}</debuglevel>
317           <verbose>${maven.compiler.verbose}</verbose>
318           <!-- las siguiente propiedades actuan si fork es true -->
319           <fork>true</fork>
320           <compilerArgument>-Xlint:all</compilerArgument>
321         </configuration>
322       </plugin>
323       <plugin>
324         <groupId>org.apache.maven.plugins</groupId>
325         <artifactId>maven-source-plugin</artifactId>
326         <version>2.2</version>
327         <executions>
328           <execution>
329             <id>attach-sources</id>
330             <goals>
331               <goal>jar</goal>
332             </goals>
333           </execution>
334         </executions>
335       </plugin>
336       <plugin>
337         <groupId>org.apache.maven.plugins</groupId>
338         <artifactId>maven-eclipse-plugin</artifactId>
339         <version>2.9</version>
340         <configuration>
341           <downloadSources>true</downloadSources>
342           <downloadJavadocs>true</downloadJavadocs>
343           <workspaceCodeStylesURL>http://svn.apache.org/repos/asf/maven/plugins/trunk/maven-eclipse-plugin/src/optional/
                eclipse-config/maven-styles.xml</workspaceCodeStylesURL>
344           <wtpversion>2.0</wtpversion>
345           <useProjectReferences>false</useProjectReferences>
346           <additionalProjectnatures>
347             <projectnature>org.springframework.ide.eclipse.core.springnature</projectnature>
348           </additionalProjectnatures>
349           <additionalBuildcommands>
350             <buildCommand>
351               <name>org.springframework.ide.eclipse.core.springbuilder</name>
352             </buildCommand>
```

```
          </additionalBuildcommands>
        </configuration>
      </plugin>
      <!-- modifica el web.xml agregandole el nombre del artefacto y version en el display-name -->
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-war-plugin</artifactId>
        <version>2.2</version>
        <configuration>
          <webResources>
            <webResource>
              <directory>${basedir}/src/main/webapp/WEB-INF</directory>
              <includes>
                <include>web.xml</include>
              </includes>
              <targetPath>WEB-INF</targetPath>
              <filtering>true</filtering>
            </webResource>
          </webResources>
        </configuration>
      </plugin>
    </plugins>

    <pluginManagement>
      <plugins>
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-dependency-plugin</artifactId>
          <version>2.4</version>
        </plugin>
      </plugins>
    </pluginManagement>
  </build>
</project>
```

Listado A.41: pom.xml

A.2.2. BookingDAO.java

```
package com.hexaid.examples.hotel.dao;

import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.Query;

import com.hexaid.examples.hotel.domain.Booking;

/**
 * @author Gabriel Belingueres
 *
 */
public class BookingDAO {

  @PersistenceContext
  private EntityManager em;

  public void saveBooking(Booking reserva) {
    em.persist(reserva);
  }

  @SuppressWarnings("unchecked")
  public List<Booking> getBookingsByUserId(String username) {
    final String sql =
        "from Booking as b join fetch b.hotel where b.user.username = :username order by b.checkinDate";
    Query query = em.createQuery(sql);
    query.setParameter("username", username);
    return query.getResultList();
  }

  public void remove(final Booking booking) {
    em.remove(booking);
  }

  public Booking getBookingById(final Long bookingId) {
    return em.find(Booking.class, bookingId);
  }

}
```

Listado A.42: BookingDAO.java

A.2.3. HotelDAO.java

```
package com.hexaid.examples.hotel.dao;

import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.Query;

import com.hexaid.examples.hotel.domain.Hotel;

/**
 * @author Gabriel Belingueres
 *
 */
public class HotelDAO {

  @PersistenceContext
  private EntityManager em;

  @SuppressWarnings("unchecked")
  public List<Hotel> search(String criteria) {
    String sql =
        "from Hotel as h where " +
        "lower(h.name) like :criteria or " +
        "lower(h.address) like :criteria or " +
```

```
26         "lower(h.city) like :criteria or " +
27         "lower(h.state) like :criteria or " +
28         "lower(h.zip) like :criteria " +
29         "order by h.name";
30     Query query = em.createQuery(sql);
31     query.setParameter("criteria", "%" + criteria.toLowerCase() + "%");
32     return query.getResultList();
33   }
34
35   public Hotel getHotelById(Long hotelId) {
36     return em.find(Hotel.class, hotelId);
37   }
38
39 }
```

Listado A.43: HotelDAO.java

A.2.4. UserDAO.java

```
1 package com.hexaid.examples.hotel.dao;
2
3 import javax.persistence.EntityManager;
4 import javax.persistence.PersistenceContext;
5 import javax.persistence.Query;
6
7 import com.hexaid.examples.hotel.domain.User;
8
9 /**
10  * @author Gabriel Belingueres
11  *
12  */
13 public class UserDAO {
14
15   @PersistenceContext
16   EntityManager em;
17
18   public User getUserByUsername(final String username) {
19     String sql = "from User as u where u.username = :username";
20     Query query = em.createQuery(sql);
21     query.setParameter("username", username);
22     return (User) query.getSingleResult();
23   }
24 }
```

Listado A.44: UserDAO.java

A.2.5. Amenity.java

```
1 package com.hexaid.examples.hotel.domain;
2
3 /**
4  * @author Gabriel Belingueres
5  *
6  */
7 public enum Amenity {
8
9   OCEAN_VIEW {
10    public String getDescription() {
11      return "Vista al ocano";
12    }
13  },
14  LATE_CHECKOUT {
15    public String getDescription() {
16      return "Salida a la tarde";
17    }
18  },
19  MINIBAR {
20    public String getDescription() {
21      return "Mini Bar";
22    }
23  };
24
25  public abstract String getDescription();
26 }
```

Listado A.45: Amenity.java

A.2.6. Booking.java

```
1 package com.hexaid.examples.hotel.domain;
2
3 import java.io.Serializable;
4 import java.math.BigDecimal;
5 import java.text.DateFormat;
6 import java.util.Arrays;
7 import java.util.Calendar;
8 import java.util.Date;
9 import java.util.HashSet;
10 import java.util.List;
11 import java.util.Set;
12
13 import javax.persistence.Basic;
14 import javax.persistence.Entity;
15 import javax.persistence.GeneratedValue;
16 import javax.persistence.GenerationType;
17 import javax.persistence.Id;
18 import javax.persistence.ManyToOne;
19 import javax.persistence.Temporal;
20 import javax.persistence.TemporalType;
21 import javax.persistence.Transient;
22 import javax.validation.constraints.Future;
23 import javax.validation.constraints.NotNull;
24
25 import org.apache.commons.lang.StringUtils;
26 import org.hibernate.validator.constraints.NotEmpty;
27 import org.springframework.format.annotation.DateTimeFormat;
```

```
28
29 import com.hexaid.examples.hotel.validation.BookingDateRange;
30
31 /**
32  * A Hotel Booking made by a User.
33  */
34 @Entity
35 @BookingDateRange
36 public class Booking implements Serializable {
37
38   private static final long serialVersionUID = 1L;
39
40   private Long id;
41
42   private User user;
43
44   private Hotel hotel;
45
46   @DateTimeFormat(pattern = "dd/MM/yyyy")
47   private Date checkinDate;
48
49   @DateTimeFormat(pattern = "dd/MM/yyyy")
50   private Date checkoutDate;
51
52   private String creditCard;
53
54   private String creditCardName;
55
56   private int creditCardExpiryMonth;
57
58   private int creditCardExpiryYear;
59
60   private boolean smoking;
61
62   private int beds;
63
64   private Set<Amenity> amenities = new HashSet<Amenity>();
65
66   public Booking() {
67     Calendar calendar = Calendar.getInstance();
68     calendar.add(Calendar.DAY_OF_MONTH, 1);
69     setCheckinDate(calendar.getTime());
70     calendar.add(Calendar.DAY_OF_MONTH, 1);
71     setCheckoutDate(calendar.getTime());
72   }
73
74   public Booking(Hotel hotel, User user) {
75     this();
76     this.hotel = hotel;
77     this.user = user;
78   }
79
80   @Transient
81   public BigDecimal getTotal() {
82     return hotel.getPrice().multiply(new BigDecimal(getNights()));
83   }
84
85   @Transient
86   public int getNights() {
87     if (checkinDate == null || checkoutDate == null) {
88       return 0;
89     } else {
90       return (int) ((checkoutDate.getTime() - checkinDate.getTime()) / 1000 / 60 / 60 / 24);
91     }
92   }
93
94   @Id
95   @GeneratedValue(strategy = GenerationType.TABLE)
96   public Long getId() {
97     return id;
98   }
99
100  public void setId(Long id) {
101    this.id = id;
102  }
103
104  @Basic
105  @Temporal(TemporalType.DATE)
106  @NotNull
107  @Future
108  public Date getCheckinDate() {
109    return checkinDate;
110  }
111
112  public void setCheckinDate(Date datetime) {
113    this.checkinDate = datetime;
114  }
115
116  @ManyToOne
117  public Hotel getHotel() {
118    return hotel;
119  }
120
121  public void setHotel(Hotel hotel) {
122    this.hotel = hotel;
123  }
124
125  @ManyToOne
126  public User getUser() {
127    return user;
128  }
129
130  public void setUser(User user) {
131    this.user = user;
132  }
133
134  @Basic
135  @Temporal(TemporalType.DATE)
136  @NotNull
137  @Future
138  public Date getCheckoutDate() {
139    return checkoutDate;
140  }
141
142  public void setCheckoutDate(Date checkoutDate) {
143    this.checkoutDate = checkoutDate;
144  }
```

```
145
146   @NotEmpty
147   public String getCreditCard() {
148     return creditCard;
149   }
150
151   public void setCreditCard(String creditCard) {
152     this.creditCard = creditCard;
153   }
154
155   @Transient
156   public String getCreditCardForShow() {
157     if (creditCard == null) {
158       return "";
159     }
160
161     final int qtyFill = creditCard.length() - 4;
162     final String repeat = StringUtils.repeat("*", qtyFill);
163     return repeat + creditCard.substring(qtyFill);
164   }
165
166   @Transient
167   public String getDescription() {
168     DateFormat df = DateFormat.getDateInstance(DateFormat.MEDIUM);
169     return hotel == null ? null : hotel.getName() + ", "
170         + df.format(getCheckinDate()) + " to "
171         + df.format(getCheckoutDate());
172   }
173
174   public boolean isSmoking() {
175     return smoking;
176   }
177
178   public void setSmoking(boolean smoking) {
179     this.smoking = smoking;
180   }
181
182   public int getBeds() {
183     return beds;
184   }
185
186   public void setBeds(int beds) {
187     this.beds = beds;
188   }
189
190   @NotEmpty
191   public String getCreditCardName() {
192     return creditCardName;
193   }
194
195   public void setCreditCardName(String creditCardName) {
196     this.creditCardName = creditCardName;
197   }
198
199   public int getCreditCardExpiryMonth() {
200     return creditCardExpiryMonth;
201   }
202
203   public void setCreditCardExpiryMonth(int creditCardExpiryMonth) {
204     this.creditCardExpiryMonth = creditCardExpiryMonth;
205   }
206
207   public int getCreditCardExpiryYear() {
208     return creditCardExpiryYear;
209   }
210
211   public void setCreditCardExpiryYear(int creditCardExpiryYear) {
212     this.creditCardExpiryYear = creditCardExpiryYear;
213   }
214
215   @Transient
216   public Set<Amenity> getAmenities() {
217     return amenities;
218   }
219
220   public void setAmenities(Set<Amenity> amenities) {
221     this.amenities = amenities;
222   }
223
224 //  private Date today() {
225 //    Calendar calendar = Calendar.getInstance();
226 //    calendar.add(Calendar.DAY_OF_MONTH, -1);
227 //    return calendar.getTime();
228 //  }
229
230   @Override
231   public String toString() {
232     return "Booking(" + user + "," + hotel + "," + amenities + ")";
233   }
234
235   @Transient
236   public List<Amenity> getAvailableAmenities() {
237     return Arrays.asList(Amenity.values());
238   }
239
240 }
```

Listado A.46: Booking.java

A.2.7. Hotel.java

```
1 package com.hexaid.examples.hotel.domain;
2
3 import java.io.Serializable;
4 import java.math.BigDecimal;
5
6 import javax.persistence.Column;
7 import javax.persistence.Entity;
8 import javax.persistence.GeneratedValue;
9 import javax.persistence.Id;
10
11 /**
12  * A hotel where users may book stays.
13  */
```

```java
@Entity
public class Hotel implements Serializable {
  private static final long serialVersionUID = 1L;

  private Long id;

  private String name;

  private String address;

  private String city;

  private String state;

  private String zip;

  private String country;

  private BigDecimal price;

  @Id
  @GeneratedValue
  public Long getId() {
    return id;
  }

  public void setId(Long id) {
    this.id = id;
  }

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }

  public String getAddress() {
    return address;
  }

  public void setAddress(String address) {
    this.address = address;
  }

  public String getCity() {
    return city;
  }

  public void setCity(String city) {
    this.city = city;
  }

  public String getZip() {
    return zip;
  }

  public void setZip(String zip) {
    this.zip = zip;
  }

  public String getState() {
    return state;
  }

  public void setState(String state) {
    this.state = state;
  }

  public String getCountry() {
    return country;
  }

  public void setCountry(String country) {
    this.country = country;
  }

  @Column(precision = 6, scale = 2)
  public BigDecimal getPrice() {
    return price;
  }

  public void setPrice(BigDecimal price) {
    this.price = price;
  }

  public Booking createBooking(User user) {
    return new Booking(this, user);
  }

  @Override
  public String toString() {
    return "Hotel(" + name + "," + address + "," + city + "," + zip + ")";
  }
}
```

Listado A.47: Hotel.java

A.2.8. User.java

```java
package com.hexaid.examples.hotel.domain;

import java.io.Serializable;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "Customer")
public class User implements Serializable {

  private static final long serialVersionUID = 1L;
```

```

  private String username;

  private String password;

  private String name;

  public User() {
  }

  public User(String username, String password, String name) {
    this.username = username;
    this.password = password;
    this.name = name;
  }

  @Id
  public String getUsername() {
    return username;
  }

  public void setUsername(String username) {
    this.username = username;
  }

  public String getPassword() {
    return password;
  }

  public void setPassword(String password) {
    this.password = password;
  }

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }

  @Override
  public String toString() {
    return "User(" + username + ")";
  }
}
```

Listado A.48: User.java

A.2.9. BookingService.java

```
package com.hexaid.examples.hotel.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import com.hexaid.examples.hotel.dao.BookingDAO;
import com.hexaid.examples.hotel.domain.Booking;

/**
 * @author Gabriel Belingueres
 *
 */
public class BookingService {

  @Autowired
  private BookingDAO bookingDAO;

  public void saveBooking(Booking reserva) {
    bookingDAO.saveBooking(reserva);
  }

  public List<Booking> getBookingsByUserId(final String username) {
    return bookingDAO.getBookingsByUserId(username);
  }

  public void cancel(final Booking booking) {
    bookingDAO.remove(booking);
  }

  public Booking getBookingById(final Long bookingId) {
    return bookingDAO.getBookingById(bookingId);
  }

}
```

Listado A.49: BookingService.java

A.2.10. HotelService.java

```
package com.hexaid.examples.hotel.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import com.hexaid.examples.hotel.dao.HotelDAO;
import com.hexaid.examples.hotel.domain.Hotel;

/**
 * @author Gabriel Belingueres
 *
 */
public class HotelService {

  @Autowired
  private HotelDAO hotelDAO;

  public List<Hotel> search(String criteria) {
```

```
20     return hotelDAO.search(criteria);
21   }
22
23   public Hotel getHotelById(Long hotelId) {
24     return hotelDAO.getHotelById(hotelId);
25   }
26
27 }
```

Listado A.50: HotelService.java

A.2.11. UserService.java

```
1 /**
2  *
3  */
4 package com.hexaid.examples.hotel.service;
5
6 import org.springframework.beans.factory.annotation.Autowired;
7
8 import com.hexaid.examples.hotel.dao.UserDAO;
9 import com.hexaid.examples.hotel.domain.User;
10
11 /**
12  * @author Gabriel Belingueres
13  *
14  */
15 public class UserService {
16
17   @Autowired
18   private UserDAO userDAO;
19
20   public User getUserByUsername(final String username) {
21     return userDAO.getUserByUsername(username);
22   }
23 }
```

Listado A.51: UserService.java

A.2.12. BookingDateRange.java

```
1 package com.hexaid.examples.hotel.validation;
2
3 import java.lang.annotation.ElementType;
4 import java.lang.annotation.Retention;
5 import java.lang.annotation.RetentionPolicy;
6 import java.lang.annotation.Target;
7
8 import javax.validation.Constraint;
9 import javax.validation.Payload;
10
11 /**
12  * @author Gabriel Belingueres
13  *
14  */
15 @Constraint(validatedBy = BookingDateRangeValidator.class)
16 @Target({ ElementType.TYPE, ElementType.METHOD, ElementType.FIELD, ElementType.CONSTRUCTOR, ElementType.PARAMETER })
17 @Retention(RetentionPolicy.RUNTIME)
18 public @interface BookingDateRange {
19
20     String message() default "Invalid check-in and check-out date range";
21
22     Class<?>[] groups() default {};
23
24     Class<? extends Payload>[] payload() default {};
25
26 }
```

Listado A.52: BookingDateRange.java

A.2.13. BookingDateRangeValidator.java

```
1 package com.hexaid.examples.hotel.validation;
2
3 import javax.validation.ConstraintValidator;
4 import javax.validation.ConstraintValidatorContext;
5
6 import com.hexaid.examples.hotel.domain.Booking;
7
8 /**
9  * @author Gabriel Belingueres
10  *
11  */
12 public class BookingDateRangeValidator implements ConstraintValidator<BookingDateRange, Booking> {
13
14     public void initialize(BookingDateRange bookingDateRange) {
15     }
16
17     public boolean isValid(Booking booking, ConstraintValidatorContext context) {
18   if ((booking.getCheckinDate() != null) && (booking.getCheckoutDate() != null)
19     && booking.getCheckoutDate().before(booking.getCheckinDate())) {
20       return false;
21   }
22   return true;
23     }
24
25 }
```

Listado A.53: BookingDateRangeValidator.java

A.2.14. CancelarReservaAction.java

```java
package com.hexaid.examples.hotel.web.action;

import org.springframework.beans.factory.annotation.Autowired;

import com.hexaid.examples.hotel.domain.Booking;
import com.hexaid.examples.hotel.service.BookingService;
import com.hexaid.struts2.annotations.Begin;
import com.hexaid.struts2.annotations.ConversationAttribute;
import com.hexaid.struts2.annotations.End;
import com.hexaid.struts2.common.ConversationAttributeType;
import com.opensymphony.xwork2.ActionSupport;

/**
 * @author Gabriel Belingueres
 *
 */
public class CancelarReservaAction extends ActionSupport {

  private static final long serialVersionUID = 1L;

  // servicios inyectados
  //
  @Autowired
  private BookingService bookingService;

  // parametros
  //
  private Long bookingId;

  @Override
  public void validate() {
    if (bookingId == null || bookingId <= 0) {
      addActionError("El identificador de Reserva es invlido");
    }
  }

  @Override
  @Begin
  @End
  @ConversationAttribute(ConversationAttributeType.REQUIRES_NEW)
  public String execute() throws Exception {
    final Booking booking = bookingService.getBookingById(bookingId);
    if (booking == null) {
      addActionError("La reserva con id " + bookingId + " no existe!");
      return INPUT;
    }

    bookingService.cancel(booking);
    addActionMessage("La Reserva id " + booking.getId() + " en el Hotel <b>"
        + booking.getHotel().getName() + "</b> ha sido eliminada!");
    return SUCCESS;
  }

  public Long getBookingId() {
    return bookingId;
  }

  public void setBookingId(Long bookingId) {
    this.bookingId = bookingId;
  }

}
```

Listado A.54: CancelarReservaAction.java

A.2.15. ReservaHotelAction.java

```java
package com.hexaid.examples.hotel.web.action;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.List;

import org.apache.struts2.interceptor.PrincipalAware;
import org.apache.struts2.interceptor.PrincipalProxy;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.validation.BindingResult;
import org.springframework.validation.DataBinder;
import org.springframework.validation.FieldError;
import org.springframework.validation.ObjectError;
import org.springframework.validation.Validator;

import com.hexaid.examples.hotel.domain.Booking;
import com.hexaid.examples.hotel.domain.Hotel;
import com.hexaid.examples.hotel.domain.User;
import com.hexaid.examples.hotel.service.BookingService;
import com.hexaid.examples.hotel.service.HotelService;
import com.hexaid.examples.hotel.service.UserService;
import com.hexaid.struts2.annotations.Begin;
import com.hexaid.struts2.annotations.ConversationAttribute;
import com.hexaid.struts2.annotations.End;
import com.hexaid.struts2.annotations.In;
import com.hexaid.struts2.annotations.Out;
import com.hexaid.struts2.common.ConversationAttributeType;
import com.opensymphony.xwork2.ActionSupport;

/**
 * @author Gabriel Belingueres
 *
 */
public class ReservaHotelAction extends ActionSupport implements PrincipalAware {

    private static final long serialVersionUID = 1L;

    // servicios
  @Autowired
  private HotelService hotelService;

  @Autowired
  private UserService userService;

  @Autowired
```

```
48  private BookingService bookingService;
49
50  @Autowired
51  private Validator validator;
52
53  // valores inyectados
54  private PrincipalProxy principalProxy;
55
56  // parametros
57  private Long hotelId;
58
59  // variables de contexto
60  @In
61  @Out
62  private Hotel currentHotel;
63
64  @In
65  @Out
66  private Booking reserva;
67
68  public void validateShowForm() {
69    if (currentHotel == null) {
70      if (hotelId == null) {
71        addActionError("El identificador de hotel no est presente!");
72      }
73      else if (hotelId <= 0) {
74        addActionError("El identificador de hotel es invlido!");
75      }
76    }
77  }
78
79  @Begin
80  public String showForm() {
81    if (currentHotel == null) {
82      currentHotel = hotelService.getHotelById(hotelId);
83      if (currentHotel == null) {
84        addActionError("El hotel buscado no existe en nuestra base de datos!");
85        return INPUT;
86      }
87    }
88
89    if (reserva == null) {
90      // crear nueva reserva:
91      // obtener el usuario logueado
92      final String username = principalProxy.getRemoteUser();
93      final User currentUser = userService.getUserByUsername(username);
94      if (currentUser == null) {
95        addActionError("El usuario '" + username + "' ya no existe en nuestra base de datos!");
96        return INPUT;
97      }
98
99      reserva = createBooking(currentUser);
100   }
101
102   return SUCCESS;
103 }
104
105 public void validateShowConfirm() {
106     DataBinder binder = new DataBinder(reserva, "reserva");
107     binder.setValidator(validator);
108     // validate the target object
109     binder.validate();
110
111     // get BindingResult that includes any validation errors
112     BindingResult results = binder.getBindingResult();
113     for(ObjectError error : results.getAllErrors()) {
114         if (error instanceof FieldError) {
115             FieldError fieldError = (FieldError) error;
116             String fieldName = "reserva." + fieldError.getField();
117             String messageCode = reserva.getClass().getSimpleName().toLowerCase() + '.' + fieldError.getField() + '.' +
                    error.getCode();
118               String message = getText(messageCode);
119               addFieldError(fieldName, message);
120         }
121         else {
122               String message = getText(error.getCode());
123               addActionError(message);
124         }
125     }
126
127   // validar fecha de expiracin tarjeta
128   final Calendar calendar = Calendar.getInstance();
129   // mes con base 1 (1 a 12)
130   final int currentMonth = calendar.get(Calendar.MONTH) + 1;
131   final int currentYear = calendar.get(Calendar.YEAR);
132
133   if(reserva.getCreditCardExpiryYear() < currentYear ||
134       (reserva.getCreditCardExpiryYear() == currentYear && reserva.getCreditCardExpiryMonth() < currentMonth)) {
135     addFieldError(
136         "reserva.creditCardExpiryMonth",
137         "La fecha de vencimiento de su tarjeta de crdito es invlida");
138   }
139
140   //TODO: Validar tarjeta de crdito
141 }
142
143 @ConversationAttribute(ConversationAttributeType.MANDATORY)
144 public String showConfirm() {
145   return SUCCESS;
146 }
147
148 @ConversationAttribute(ConversationAttributeType.MANDATORY)
149 @End
150 public String confirm() {
151   // salva la reserva en la base de datos
152   bookingService.saveBooking(reserva);
153
154   DateFormat df = new SimpleDateFormat("dd/MM/yyyy");
155
156   addActionMessage(
157       "La reserva en el hotel <b>" + currentHotel.getName()
158       + "</b> desde el da " + df.format(reserva.getCheckinDate()) + " al "
159       + df.format(reserva.getCheckoutDate())
160       + " ha sido salvada exitosamente.");
161
162   // retornando SUCCESS se termina la conversacin
163   return SUCCESS;
```

```
  }

  @ConversationAttribute(ConversationAttributeType.MANDATORY)
  @End(beforeRedirect=true, commit=false)
  public String cancel() {
    return SUCCESS;
  }

  private Booking createBooking(final User currentUser) {
    final Booking booking = new Booking(currentHotel, currentUser);

    Calendar cal = Calendar.getInstance();
    // a partir de maana
    cal.add(Calendar.DAY_OF_MONTH, 1);
    booking.setCheckinDate(cal.getTime());

    // hasta pasado maana
    cal.add(Calendar.DAY_OF_MONTH, 1);
    booking.setCheckoutDate(cal.getTime());

    booking.setSmoking(false);

    return booking;
  }

  public Long getHotelId() {
    return hotelId;
  }

  public void setHotelId(Long hotelId) {
    this.hotelId = hotelId;
  }

  @Override
  public void setPrincipalProxy(PrincipalProxy principalProxy) {
    this.principalProxy = principalProxy;
  }

  public Booking getReserva() {
    return reserva;
  }

  public void setReserva(Booking reserva) {
    this.reserva = reserva;
  }

  public List<Integer> getAvailableYearsCreditCard() {
    final List<Integer> list = new ArrayList<Integer>();
    final int currentYear = Calendar.getInstance().get(Calendar.YEAR);
    for(int i=currentYear; i < currentYear+20; ++i) {
      list.add(i);
    }
    return list;
  }

}
```

Listado A.55: ReservaHotelAction.java

A.2.16. SearchAction.java

```
package com.hexaid.examples.hotel.web.action;

import java.security.Principal;
import java.util.List;

import org.apache.struts2.interceptor.PrincipalAware;
import org.apache.struts2.interceptor.PrincipalProxy;
import org.springframework.beans.factory.annotation.Autowired;

import com.hexaid.examples.hotel.domain.Booking;
import com.hexaid.examples.hotel.domain.Hotel;
import com.hexaid.examples.hotel.service.BookingService;
import com.hexaid.examples.hotel.service.HotelService;
import com.opensymphony.xwork2.ActionSupport;

/**
 * @author Gabriel Belingueres
 *
 */
public class SearchAction extends ActionSupport implements PrincipalAware {

  private static final long serialVersionUID = 1L;

  @Autowired
  private HotelService hotelService;

  @Autowired
  private BookingService bookingService;

  // inyectado por Struts
  private Principal principal;

  // parametros
  private String criteria;

  // resultado
  private List<Hotel> hotelList;
  private List<Booking> bookingList;

  public String show() throws Exception {
    if (principal != null) {
      bookingList = bookingService.getBookingsByUserId(principal.getName());
    }
    return SUCCESS;
  }

  @Override
  public String execute() throws Exception {
    setHotelList(hotelService.search(criteria));
    return "success_result";
  }

  public String getCriteria() {
```

```
    return criteria;
  }

  public void setCriteria(String criteria) {
    this.criteria = criteria;
  }

  public List<Hotel> getHotelList() {
    return hotelList;
  }

  public void setHotelList(List<Hotel> hotelList) {
    this.hotelList = hotelList;
  }

  @Override
  public void setPrincipalProxy(PrincipalProxy principalProxy) {
    this.principal = principalProxy.getUserPrincipal();
  }

  public List<Booking> getBookingList() {
    return bookingList;
  }

  public void setBookingList(List<Booking> bookingList) {
    this.bookingList = bookingList;
  }

}
```

Listado A.56: SearchAction.java

A.2.17. ViewHotelAction.java

```
package com.hexaid.examples.hotel.web.action;

import org.springframework.beans.factory.annotation.Autowired;

import com.hexaid.examples.hotel.domain.Hotel;
import com.hexaid.examples.hotel.service.HotelService;
import com.opensymphony.xwork2.ActionSupport;

/**
 * @author Gabriel Belingueres
 *
 */
public class ViewHotelAction extends ActionSupport {

  private static final long serialVersionUID = 1L;

  // servicios
  @Autowired
  private HotelService hotelService;

  // paramatros
  private Long hotelId;

  // resultados
  private Hotel hotel;

  @Override
  public String execute() throws Exception {
    hotel = hotelService.getHotelById(hotelId);
    if (hotel == null) {
      addActionError("El hotel buscado no existe en nuestra base de datos!");
      return INPUT;
    }
    return SUCCESS;
  }

  public Long getHotelId() {
    return hotelId;
  }

  public void setHotelId(Long hotelId) {
    this.hotelId = hotelId;
  }

  public Hotel getHotel() {
    return hotel;
  }

  public void setHotel(Hotel hotel) {
    this.hotel = hotel;
  }

}
```

Listado A.57: ViewHotelAction.java

A.2.18. ReservaHotelAction-reservaHotel_showConfirm-validation.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator 1.0.3//EN" "http://www.opensymphony.com/xwork/xwork-
    validator-1.0.3.dtd" >
<validators>
<!--
  <field name="reserva.checkinDate">
    <field-validator type="required">
      <message>La fecha de Ingreso es requerida</message>
    </field-validator>
  </field>
  <field name="reserva.checkoutDate">
    <field-validator type="required">
      <message>La fecha de Egreso es requerida</message>
    </field-validator>
  </field>
-->
  <field name="reserva.beds">
    <field-validator type="required">
```

```
      <message>Debe seleccionar las camas</message>
    </field-validator>
  </field>
  <field name="reserva.smoking">
    <field-validator type="required">
      <message>Debe seleccionar si el cuarto es para Fumador o no</message>
    </field-validator>
  </field>
<!--
  <field name="reserva.creditCardName">
    <field-validator type="requiredstring">
      <message>Debe seleccionar el medio de pago</message>
    </field-validator>
  </field>
  <field name="reserva.creditCard">
    <field-validator type="requiredstring">
      <message>El número de tarjeta de crédito es requerido</message>
    </field-validator>
    <field-validator type="stringlength">
      <param name="minLength">14</param>
      <param name="maxLength">16</param>
      <message>El número de tarjeta debe tener entre ${minLength} y ${maxLength} dígitos</message>
    </field-validator>
    <field-validator type="int">
      <message>El número de tarjeta debe contener sólo dígitos</message>
    </field-validator>
  </field>
-->
  <field name="reserva.creditCardExpiryMonth">
    <field-validator type="int">
      <param name="min">1</param>
      <param name="max">12</param>
      <message>El mes debe ser un número entre 1 (Enero) y 12 (Diciembre)</message>
    </field-validator>
  </field>
  <field name="reserva.creditCardExpiryYear">
    <field-validator type="int">
      <param name="min">2011</param>
      <param name="max">2099</param>
      <message>El año de vencimiento es inválido</message>
    </field-validator>
  </field>
</validators>
```

Listado A.58: ReservaHotelAction-reservaHotel_showConfirm-validation.xml

A.2.19. import.sql

```
insert into Customer (username, name) values ('javier', 'Javier')
insert into Customer (username, name) values ('claudia', 'Claudia')
insert into Customer (username, name) values ('gabriel', 'Gabriel')
insert into Hotel (id, price, name, address, city, state, zip, country) values (1, 199, 'Westin Diplomat', '3555 S. Ocean Drive', 'Hollywood', 'FL', '33019', 'USA')
insert into Hotel (id, price, name, address, city, state, zip, country) values (2, 60, 'Jameson Inn', '890 Palm Bay Rd NE', 'Palm Bay', 'FL', '32905', 'USA')
insert into Hotel (id, price, name, address, city, state, zip, country) values (3, 199, 'Chilworth Manor', 'The Cottage, Southampton Business Park', 'Southampton', 'Hants', 'SO16 7JF', 'UK')
insert into Hotel (id, price, name, address, city, state, zip, country) values (4, 120, 'Marriott Courtyard', 'Tower Place, Buckhead', 'Atlanta', 'GA', '30305', 'USA')
insert into Hotel (id, price, name, address, city, state, zip, country) values (5, 180, 'Doubletree', 'Tower Place, Buckhead', 'Atlanta', 'GA', '30305', 'USA')
insert into Hotel (id, price, name, address, city, state, zip, country) values (6, 450, 'W Hotel', 'Union Square, Manhattan', 'NY', 'NY', '10011', 'USA')
insert into Hotel (id, price, name, address, city, state, zip, country) values (7, 450, 'W Hotel', 'Lexington Ave, Manhattan', 'NY', 'NY', '10011', 'USA')
insert into Hotel (id, price, name, address, city, state, zip, country) values (8, 250, 'Hotel Rouge', '1315 16th Street NW', 'Washington', 'DC', '20036', 'USA')
insert into Hotel (id, price, name, address, city, state, zip, country) values (9, 300, '70 Park Avenue Hotel', '70 Park Avenue', 'NY', 'NY', '10011', 'USA')
insert into Hotel (id, price, name, address, city, state, zip, country) values (10, 300, 'Conrad Miami', '1395 Brickell Ave', 'Miami', 'FL', '33131', 'USA')
insert into Hotel (id, price, name, address, city, state, zip, country) values (11, 80, 'Sea Horse Inn', '2106 N Clairemont Ave', 'Eau Claire', 'WI', '54703', 'USA')
insert into Hotel (id, price, name, address, city, state, zip, country) values (12, 90, 'Super 8 Eau Claire Campus Area', '1151 W Macarthur Ave', 'Eau Claire', 'WI', '54701', 'USA')
insert into Hotel (id, price, name, address, city, state, zip, country) values (13, 160, 'Marriot Downtown', '55 Fourth Street', 'San Francisco', 'CA', '94103', 'USA')
insert into Hotel (id, price, name, address, city, state, zip, country) values (14, 200, 'Hilton Diagonal Mar', 'Passeig del Taulat 262-264', 'Barcelona', 'Catalunya', '08019', 'Spain')
insert into Hotel (id, price, name, address, city, state, zip, country) values (15, 210, 'Hilton Tel Aviv', 'Independence Park', 'Tel Aviv', '', '63405', 'Israel')
insert into Hotel (id, price, name, address, city, state, zip, country) values (16, 240, 'InterContinental Tokyo Bay', 'Takeshiba Pier', 'Tokyo', '', '105', 'Japan')
insert into Hotel (id, price, name, address, city, state, zip, country) values (17, 130, 'Hotel Beaulac', ' Esplanade Lopold-Robert 2', 'Neuchatel', '', '2000', 'Switzerland')
insert into Hotel (id, price, name, address, city, state, zip, country) values (18, 140, 'Conrad Treasury Place', 'William & George Streets', 'Brisbane', 'QLD', '4001', 'Australia')
insert into Hotel (id, price, name, address, city, state, zip, country) values (19, 230, 'Ritz Carlton', '1228 Sherbrooke St', 'West Montreal', 'Quebec', 'H3G1H6', 'Canada')
insert into Hotel (id, price, name, address, city, state, zip, country) values (20, 460, 'Ritz Carlton', 'Peachtree Rd, Buckhead', 'Atlanta', 'GA', '30326', 'USA')
insert into Hotel (id, price, name, address, city, state, zip, country) values (21, 220, 'Swissotel', '68 Market Street', 'Sydney', 'NSW', '2000', 'Australia')
insert into Hotel (id, price, name, address, city, state, zip, country) values (22, 250, 'Meli White House', 'Albany Street', 'Regents Park London', '', 'NW13UP', 'Great Britain')
insert into Hotel (id, price, name, address, city, state, zip, country) values (23, 210, 'Hotel Allegro', '171 West Randolph Street', 'Chicago', 'IL', '60601', 'USA')
```

Listado A.59: import.sql

A.2.20. logback.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- For assistance related to logback-translator or configuration  -->
<!-- files in general, please contact the logback user mailing list -->
<!-- at http://www.qos.ch/mailman/listinfo/logback-user             -->
<!--                                                                -->
<!-- For professional support please see                            -->
<!--    http://www.qos.ch/shop/products/professionalSupport         -->
```

```
<!--                                                                -->
<configuration>
    <!--See http://logback.qos.ch/manual/appenders.html#RollingFileAppender-->
    <!--and http://logback.qos.ch/manual/appenders.html#TimeBasedRollingPolicy-->
    <!--for further documentation-->
  <appender name="archivo" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <file>${catalina.home}/logs/hotel.log</file>
    <encoder>
      <pattern>%d{dd/MM/yyyy HH:mm:ss} %5p %C:%L - %msg%n</pattern>
    </encoder>
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
      <fileNamePattern>${catalina.home}/logs/hotel.log.%d</fileNamePattern>
    </rollingPolicy>
  </appender>
  <appender name="consola" class="ch.qos.logback.core.ConsoleAppender">
    <encoder>
      <pattern>%d{dd/MM/yyyy HH:mm:ss} %5p %C:%L - %msg%n</pattern>
    </encoder>
  </appender>
  <logger name="com.hexaid" level="DEBUG"/>
  <logger name="org.springframework" level="INFO"/>
  <!-- STRUTS 2.2.x PARA QUE NO TIRE WARNINGS DE VARIABLES QUE NO ENCUENTRA, PONER EN ERROR o FATAL -->
  <logger name="com.opensymphony.xwork2.ognl" level="ERROR"/>
  <root level="WARN">
    <appender-ref ref="consola"/>
  </root>
</configuration>
```

Listado A.60: logback.xml

A.2.21. persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/
                 persistence_2_0.xsd"
             version="2.0">
   <persistence-unit name="hotelDatabase" transaction-type="RESOURCE_LOCAL">
      <provider>org.hibernate.ejb.HibernatePersistence</provider>
      <class>com.hexaid.examples.hotel.domain.User</class>
      <class>com.hexaid.examples.hotel.domain.Booking</class>
      <class>com.hexaid.examples.hotel.domain.Hotel</class>
      <properties>
         <property name="hibernate.dialect" value="org.hibernate.dialect.HSQLDialect"/>
         <property name="hibernate.hbm2ddl.auto" value="create-drop" />
         <property name="hibernate.show_sql" value="true"/>
         <property name="hibernate.cache.provider_class" value="org.hibernate.cache.HashtableCacheProvider"/>
      </properties>
   </persistence-unit>
</persistence>
```

Listado A.61: persistence.xml

A.2.22. struts.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC
          "-//Apache Software Foundation//DTD Struts Configuration 2.1//EN"
          "http://struts.apache.org/dtds/struts-2.1.dtd">
<struts>
  <constant name="struts.enable.DynamicMethodInvocation" value="false" />
  <constant name="struts.devMode" value="true" />
  <constant name="struts.i18n.encoding" value="ISO-8859-1" />
  <constant name="struts.objectFactory" value="spring" />
  <constant name="struts.ui.theme" value="simpleconv" />
  <constant name="struts.custom.i18n.resources" value="ValidationMessages" />

  <!-- timeout de 20 segundos para demostracion
  <constant name="com.hexaid.struts2.conversation.expiration.maxInactiveInterval" value="20" />
  -->

  <!-- timeout por pagina para demostracion
  <constant name="com.hexaid.struts2.conversation.expiration.policy" value="perview" />
  -->

  <package name="abstractpackage" extends="conversationPackage" abstract="true">
    <result-types>
      <!-- Result type para Tiles con soporte de conversaciones -->
      <result-type name="tiles"
        class="com.hexaid.struts2.result.TilesConversationResult" />
    </result-types>

    <interceptors>
      <!--
      JPA specific interceptor stack supporting conversations and bijection.
      It is a paramsPrepareParamsStack with the addition of conversation and
      bijection interceptors BEFORE validation and BEFORE params
      -->
      <interceptor-stack name="jpaConversationStack">
        <interceptor-ref name="abstractConversationSupportStack">
          <param name="exception.logEnabled">true</param>
          <param name="exception.logLevel">ERROR</param>
          <param name="exception.logCategory">com.hexaid</param>

          <param name="conversation.persistence">jpa-spring</param>
        </interceptor-ref>
      </interceptor-stack>
    </interceptors>

    <default-interceptor-ref name="jpaConversationStack" />

    <global-results>
      <result name="exception" type="tiles">/exception.tiles</result>

      <result name="conversation_not_found" type="redirectAction">
        <param name="actionName">index</param>
        <param name="namespace">/</param>
      </result>
    </global-results>
```

```
55
56      <global-exception-mappings>
57        <exception-mapping exception="java.lang.Exception" result="exception" />
58      </global-exception-mappings>
59
60    </package>
61
62    <package name="publicPackage" extends="abstractpackage" namespace="/">
63
64      <action name="index">
65        <result name="success" type="tiles">/index.tiles</result>
66      </action>
67
68      <action name="login">
69        <result name="success" type="tiles">/login.tiles</result>
70      </action>
71
72      <action name="search_*"
73        class="com.hexaid.examples.hotel.web.action.SearchAction" method="{1}">
74        <result name="success" type="tiles">/search.tiles</result>
75        <result name="success_result" type="tiles">/search_result.tiles
76        </result>
77      </action>
78
79      <action name="viewHotel"
80        class="com.hexaid.examples.hotel.web.action.ViewHotelAction">
81        <result name="success" type="tiles">/viewHotel.tiles</result>
82        <result name="input" type="tiles">/search.tiles</result>
83      </action>
84
85      <action name="infoAutor">
86        <result type="tiles">/infoAutor.tiles</result>
87      </action>
88    </package>
89
90    <package name="protectedPackage" extends="abstractpackage" namespace="/protected">
91
92      <action name="reservaHotel_showForm"
93        class="com.hexaid.examples.hotel.web.action.ReservaHotelAction"
94        method="showForm">
95        <result name="success" type="tiles">
96          <param name="location">/protected/reservaHotel_showForm.tiles</param>
97          <param name="maxInactiveInterval">10</param>
98        </result>
99        <result name="input" type="tiles">/search.tiles</result>
100     </action>
101
102     <action name="reservaHotel_showConfirm"
103       class="com.hexaid.examples.hotel.web.action.ReservaHotelAction"
104       method="showConfirm">
105       <result name="success" type="tiles">
106         <param name="location">/protected/reservaHotel_showConfirm.tiles</param>
107         <param name="maxInactiveInterval">25</param>
108       </result>
109       <result name="input" type="tiles">
110         <param name="location">/protected/reservaHotel_showForm.tiles</param>
111         <param name="maxInactiveInterval">10</param>
112       </result>
113     </action>
114
115     <action name="reservaHotel_confirm"
116       class="com.hexaid.examples.hotel.web.action.ReservaHotelAction"
117       method="confirm">
118       <result name="success" type="redirectAction">
119         <param name="actionName">search_show</param>
120         <param name="namespace">/</param>
121       </result>
122       <result name="input" type="tiles">/protected/reservaHotel_showConfirm.tiles</result>
123     </action>
124
125     <action name="reservaHotel_cancel"
126       class="com.hexaid.examples.hotel.web.action.ReservaHotelAction"
127       method="cancel">
128       <result name="success" type="redirectAction">
129         <param name="actionName">search_show</param>
130         <param name="namespace">/</param>
131       </result>
132     </action>
133
134     <action name="cancelBooking"
135       class="com.hexaid.examples.hotel.web.action.CancelarReservaAction">
136       <result name="success" type="redirectAction">
137         <param name="actionName">search_show</param>
138         <param name="namespace">/</param>
139       </result>
140       <result name="input" type="tiles">/exception.tiles</result>
141     </action>
142   </package>
143
144 </struts>
```

Listado A.62: struts.xml

A.2.23. checkboxlist.ftl

```
1  <#--
2  /*
3   * Copia modificada del tema "simple"
4   *
5   */
6  -->
7  <#assign itemCount = 0/>
8  <#if parameters.list??>
9      <@s.iterator value="parameters.list">
10         <#assign itemCount = itemCount + 1/>
11         <#if parameters.listKey??>
12             <#assign itemKey = stack.findValue(parameters.listKey)/>
13         <#else>
14             <#assign itemKey = stack.findValue('top')/>
15         </#if>
16         <#if parameters.listValue??>
17             <#assign itemValue = stack.findString(parameters.listValue)?default("")/>
18         <#else>
19             <#assign itemValue = stack.findString('top')/>
```

```
        </#if>
<#assign itemKeyStr=itemKey.toString() />
<input type="checkbox" name="${parameters.name?html}" value="${itemKeyStr?html}" id="${parameters.name?html}-${itemCount}"<#
      rt/>
        <#if tag.contains(parameters.nameValue, itemKey)>
 checked="checked"<#rt/>
        </#if>
        <#if parameters.disabled?default(false)>
 disabled="disabled"<#rt/>
        </#if>
        <#if parameters.title??>
 title="${parameters.title?html}"<#rt/>
        </#if>
        <#include "/${parameters.templateDir}/simple/css.ftl" />
        <#include "/${parameters.templateDir}/simple/scripting-events.ftl" />
        <#include "/${parameters.templateDir}/simple/common-attributes.ftl" />
/>
<#--
<label for="${parameters.name?html}-${itemCount}" class="checkboxLabel">${itemValue?html}</label>
-->
${itemValue?html}
    </@s.iterator>
<#else>
   
</#if>
<input type="hidden" id="__multiselect_${parameters.id?html}" name="__multiselect_${parameters.name?html}" value=""<#rt/>
<#if parameters.disabled?default(false)>
 disabled="disabled"<#rt/>
</#if>
 />
```

Listado A.63: checkboxlist.ftl

A.2.24. ValidationMessages_es.properties

```
# Spring security related messages
SPRING_SECURITY_LAST_EXCEPTION=Las credenciales ingresadas son invlidas.

# JSR 303 Validation related messages
booking.checkinDate.NotNull=La fecha de check in es requerida
booking.checkinDate.Future=La fecha de check in debe estar en el futuro
booking.checkinDate.beforeToday=La fecha de check in debe ser una fecha en el futuro
booking.checkinDate.typeMismatch=La fecha de check in debe estar en el formato dd/mm/aaaa

booking.checkoutDate.NotNull=La fecha de check out es requerida
booking.checkoutDate.Future=La fecha de check out debe estar en el futuro
booking.checkoutDate.typeMismatch=La fecha de check out debe estar en el formato dd/mm/aaaa
booking.BookingDateRange=La fecha de check out debe ser posterior a la de check in

booking.creditCard.NotEmpty=La tarjeta de crdito es requerida
booking.creditCardName.NotEmpty=El nombre de la tarjeta de crdito es requerido

NotEmpty=El {0} campo es requerido
NotNull=El {0} campo es requerido
Future=La fecha {0} debe estar en el futuro

BookingDateRange=La fecha de salida no puede ser menor a la de entrada

# Struts conversion errors
invalid.fieldvalue.reserva.checkinDate=La fecha debe estar en formato dd/mm/aaaa
invalid.fieldvalue.reserva.checkoutDate=La fecha debe estar en formato dd/mm/aaaa
xwork.default.invalid.fieldvalue=El campo tiene un formato invlido
```

Listado A.64: ValidationMessages_es.properties

A.2.25. BaseLayout.jsp

```
<%--
response.addHeader("Cache-Control","no-cache, no-store, max-age=0");
response.addHeader("Pragma","no-cache");
response.addDateHeader("Expires", 1);
--%>
<%@ taglib uri="http://tiles.apache.org/tags-tiles" prefix="tiles"%>
<%@ taglib uri="/struts-tags" prefix="s" %>
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>UNLP Viajes <tiles:insertAttribute name="title" ignore="true" /></title>
    <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
    <link rel="stylesheet" type="text/css" href="<s:url includeContext="true" includeParams="none" namespace="/" value="/css/
          reset.css" />" media="screen" />
    <link rel="stylesheet" type="text/css" href="<s:url includeContext="true" includeParams="none" namespace="/" value="/css/
          text.css" />" media="screen" />
    <link rel="stylesheet" type="text/css" href="<s:url includeContext="true" includeParams="none" namespace="/" value="/css
          /960.css" />" media="screen" />
    <link rel="stylesheet" type="text/css" href="<s:url includeContext="true" includeParams="none" namespace="/" value="/css/
          layout.css" />" media="screen" />
    <link rel="stylesheet" type="text/css" href="<s:url includeContext="true" includeParams="none" namespace="/" value="/css/
          nav.css" />" media="screen" />
    <!--[if IE 6]><link rel="stylesheet" type="text/css" href="<s:url includeContext="true" includeParams="none" namespace="/
          " value="/css/ie6.css" />" media="screen" /><![endif]-->
    <!--[if IE 7]><link rel="stylesheet" type="text/css" href="<s:url includeContext="true" includeParams="none" namespace="/
          " value="/css/ie.css" />" media="screen" /><![endif]-->
    <link rel="stylesheet" type="text/css" href="<s:url includeContext="true" includeParams="none" namespace="/" value="/css/
          hotel.css" />" media="screen" />
    <link rel="icon" type="image/x-icon" href="http://www.info.unlp.edu.ar/images/frontend/favicon.ico" />
  </head>
  <body>
    <div class="container_12">

      <tiles:insertAttribute name="header" />

      <div class="clear"></div>

      <tiles:insertAttribute name="menu" />

      <tiles:insertAttribute name="body" />

```

```
35      <tiles:insertAttribute name="footer" />
```

Listado A.65: BaseLayout.jsp

A.2.26. exception.jsp

```
1 <%@ taglib uri="/struts-tags" prefix="s"%>
2 <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
3   pageEncoding="ISO-8859-1"%>
4 <div class="box">
5   <h2>Ha ocurrido un error inesperado:</h2>
6   <div class="block">
7     <s:if test="hasActionErrors()">
8       <s:actionerror />
9     </s:if>
10    <s:if test="exception != null">
11      <p><s:property value="exception" /></p>
12    </s:if>
13  </div>
14 </div>
15
16 <div class="clear"></div>
17
18 <div class="box">
19   <h2 class="errorMessage">
20     <a href="#" id="toggle-grid" class="visible">Ms informacin</a>
21   </h2>
22   <div class="block errorMessage hidden" id="grid">
23     <s:if test="exception != null">
24       <s:property value="exceptionStack" />
25     </s:if>
26   </div>
27 </div>
```

Listado A.66: exception.jsp

A.2.27. Footer.jsp

```
1 <%@ taglib uri="/struts-tags" prefix="s" %>
2 <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
3     pageEncoding="ISO-8859-1"%>
4       </div>
5
6       <div class="grid_12">
7         <h2></h2>
8       </div>
9
10      <div class="clear"></div>
11      <div class="grid_12" id="site_info">
12        <div class="box">
13          <p>Fluid 960 Grid System, created by <a href="http://www.domain7.com/WhoWeAre/StephenBau.html">Stephen Bau</a>,
                based on the <a href="http://960.gs/">960 Grid System</a> by <a href="http://sonspring.com/journal/960-grid-
                system">Nathan Smith</a>. Released under the
14    <a href="licenses/GPL_license.txt">GPL</a> / <a href="licenses/MIT_license.txt">MIT</a> <a href="README.txt">Licenses</a>
            .</p>
15        </div>
16      </div>
17      <div class="clear"></div>
18    </div>
19    <script type="text/javascript" src="<s:url includeContext="true" includeParams="none" namespace="/" value="/js/jquery
          -1.3.2.min.js" />"></script>
20    <script type="text/javascript" src="<s:url includeContext="true" includeParams="none" namespace="/" value="/js/jquery-ui.
          js" />"></script>
21    <script type="text/javascript" src="<s:url includeContext="true" includeParams="none" namespace="/" value="/js/jquery-
          fluid16.js" />"></script>
22  </body>
23 </html>
```

Listado A.67: Footer.jsp

A.2.28. fotoAutor.jsp

```
1 <%@ taglib uri="/struts-tags" prefix="s" %>
2 <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
3     pageEncoding="ISO-8859-1"%>
4       <div class="grid_3">
5         <div class="box">
6           <h2>Sobre UNLP Viajes</h2>
7           <div class="block">
8             <p>Esta aplicacin demuestra el uso de conversaciones en Struts 2.</p>
9           </div>
10        </div>
11        <div class="box">
12          <img alt="Hotel" width="100%" height="100%" src="<s:url includeContext="true" includeParams="none" namespace="/"
                value="/img/gabriel-belingueres.jpg" />">
13        </div>
14      </div>
15      <div class="grid_9">
```

Listado A.68: fotoAutor.jsp

A.2.29. Header.jsp

```
1 <%@ taglib uri="/struts-tags" prefix="s" %>
2 <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
3 <%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
4       <div class="grid_8">
5         <h1 id="branding">
6 <s:url var="indexUrl" includeContext="true" includeParams="none" namespace="/" action="index" />
7           <a href="${indexUrl}">UNLP Viajes</a>
8         </h1>
9       </div>
```

```
      <div class="grid_4" style="text-align: center;">
<s:url var="S2ImageUrl" includeContext="true" includeParams="none" namespace="/" value="/img/struts2.png" />
            <img src="${S2ImageUrl}" alt="Powered by Struts 2" title="Powered by Struts 2" />
      </div>
      <div class="clear"></div>
      <div class="grid_12">
        <ul class="nav main">
          <li>
<s:url var="autorUrl" includeContext="true" includeParams="none" namespace="/" action="infoAutor" />
            <a href="${autorUrl}">Tesina / Autor</a>
          </li>
          <li class="secondary">
<c:choose>
  <c:when test="${empty pageContext.request.remoteUser}">
    <s:url var="loginUrl" includeContext="true" includeParams="none" namespace="/" action="login" />
            <a href="${loginUrl}" title="Presione aqu para loguearse al sitio y completar su reserva">Ingresar</a>
  </c:when>
  <c:otherwise>
    <s:url var="logoutUrl" includeContext="true" includeParams="none" namespace="/" value="/j_spring_security_logout" />
            <a href="${logoutUrl}" title="Cerrar Sesin">Cerrar Sesin</a>
  </c:otherwise>
</c:choose>
          </li>
        </ul>
      </div>
      <div class="clear"></div>
      <div class="grid_12">
        <h2 id="page-heading">
<c:choose>
  <c:when test="${empty pageContext.request.remoteUser}">
          Bienvenidos a Struts Viajes
  </c:when>
  <c:otherwise>
          Bienvenido a Struts Viajes, ${pageContext.request.remoteUser}!!!
  </c:otherwise>
</c:choose>

        </h2>
      </div>
```

Listado A.69: Header.jsp

A.2.30. index.html

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>UNLP Viajes</title>
    <meta http-equiv="refresh" content="0; url=index.action" />
  </head>
  <body>
  </body>
</html>
```

Listado A.70: index.html

A.2.31. index.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
        <div class="box">
          <h2>Las funcionalidades demostradas en la aplicacin incluyen:</h2>
          <div class="block">
            <ul>
              <li>Lgica de presentacin con Struts 2 demostrando el uso de conversaciones.</li>
              <li>Capa de servicios y gestin de transacciones usando Spring</li>
              <li>Capa de acceso a datos con JPA (usando la implementacin de Hibernate)</li>
              <li>Generacin de vistas demostrando el uso de Apache Tiles y JSP</li>
              <li>Gestin autenticacin y autorizacin con Spring Security</li>
              <li>Motor de base de datos relacional HSQLDB</li>
            </ul>
          </div>
        </div>

      <div class="clear"></div>

      <div class="block">
          <h2 style="text-align: center"><a href="search_show.action">Comience aqu su experiencia de viajar con UNLP Viajes
              !!!</a></h2>
      </div>
```

Listado A.71: index.jsp

A.2.32. infoAutor.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
        <div class="box">
          <h2>SOBRE LA TESINA</h2>

          <div class="block" id="tables">
            <table>
              <tr>
                <th>Ttulo</th>
                <td>Una Solucin para el Soporte de Conversaciones en Aplicaciones Web Java</td>
              </tr>
              <tr>
                <th>Director</th>
                <td>Lic. Javier Daz</td>
              </tr>
              <tr>
                <th>Codirector</th>
                <td>Lic. Claudia Queiruga</td>
              </tr>
              <tr>
                <th>Carrera</th>
```

```
                <td>Licenciatura en Informtica Plan 90</td>
              </tr>
              <tr>
                <th></th>
                <td></td>
              </tr>
            </table>
          </div>
        </div>

        <div class="box">
          <h2>SOBRE EL AUTOR</h2>

          <div class="block" id="tables">
            <table>
              <tr>
                <th>Nombre</th>
                <td>Gabriel Alfredo Belingueres</td>
              </tr>
              <tr>
                <th>Correo electrnico</th>
                <td><a href="mailto:gbelingueres@info.unlp.edu.ar">gbelingueres@info.unlp.edu.ar</a></td>
              </tr>
              <tr>
                <th>Redes sociales</th>
                <td>
                  <a href="http://www.linkedin.com/pub/gabriel-belingueres/2/57/903" target="_blank">Linkedin</a>
                  <a href="https://www.facebook.com/gabriel.belingueres" target="_blank">Facebook</a>
                </td>
              </tr>
            </table>
          </div>
        </div>
      <div class="clear"></div>

      <div class="block">
          <h2 style="text-align: center"><a href="search_show.action">Comience aqu su experiencia de viajar con UNLP Viajes
                !!!</a></h2>
      </div>
```

Listado A.72: infoAutor.jsp

A.2.33. login.jsp

```
<%@ taglib uri="/struts-tags" prefix="s" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<%@ page import="org.springframework.security.web.WebAttributes" %>
<%@ page import="org.springframework.security.core.AuthenticationException" %>

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>

        <div class="box">
          <h2>
            Formularios de Ingreso
          </h2>
          <div class="block" id="login-forms">
<c:if test="${not empty param.login_error}">
  <div class="errorMessage">
    Su intento de ingreso no fue exitoso, intente nuevamente.<br />
    Razn:
    <s:set var="AUTHENTICATION_EXCEPTION_KEY" scope="page" value="@org.springframework.security.web.
          WebAttributes@AUTHENTICATION_EXCEPTION"/>
    <s:text name="%{#attr.AUTHENTICATION_EXCEPTION_KEY}" />
  </div>
</c:if>
            <form name="login" action="${pageContext.request.contextPath}/j_spring_security_check" method="POST">
              <fieldset class="login">
                <legend>Ingreso</legend>
                <p class="notice">Ingrese para completar su reserva.</p>
                <p>
                  <label for="j_username">Usuario: </label>

                  <input type="text" name="j_username"
                  <c:if test="${not empty param.login_error}">value="<%= session.getAttribute(WebAttributes.LAST_USERNAME) %>
                        "</c:if>
                   />
                </p>
                <p>
                  <label for="j_password">Contrasea: </label>
                  <input type="password" name="j_password" />
                </p>
                <p>
                  <label for="_spring_security_remember_me">Recordarme por dos semanas: </label>
                  <input type="checkbox" name="_spring_security_remember_me" id="remember_me" style="width: 1em;" />
                </p>
                <input class="login button" type="submit" value="Ingreso" />
              </fieldset>
            </form>
            <form action="">
              <fieldset>
                <legend>Usuarios de prueba</legend>
                <p>Ingrese al sistema con alguno de los siguientes usuario/contrasea:</p>
                <p class="notice">
                javier/javier<br/>
                claudia/claudia<br/>
                gabriel/gabriel<br/>
                </p>
              </fieldset>
            </form>

          </div>
        </div>
<script type="text/javascript">
document.forms['login'].j_username.focus();
</script>
```

Listado A.73: login.jsp

A.2.34. Menu.jsp

```
<%@ taglib uri="/struts-tags" prefix="s" %>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
      <div class="grid_3">
        <div class="box">
          <h2>Sobre UNLP Viajes</h2>
          <div class="block">
            <p>Esta aplicacin demuestra el uso de conversaciones en Struts 2.</p>
          </div>
        </div>
        <div class="box">
          <img alt="Hotel" width="100%" src="<s:url includeContext="true" includeParams="none" namespace="/" value="/img/
              Dubaihotel.jpg" />">
        </div>
      </div>
      <div class="grid_9">
```

Listado A.74: Menu.jsp

A.2.35. reservaHotel_showConfirm.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
<%@ taglib uri="/conversation-struts2-tags" prefix="sconv" %>
<%@ taglib uri="/struts-tags" prefix="s" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>

<sconv:useConversation/>

<div class="grid_3">
    <h2>${currentHotel.name}</h2>
    <div id="blockquote" class="block" style="margin: 0px;">
        <blockquote>
            <p>
                ${currentHotel.address}<br /> ${currentHotel.city}, ${currentHotel.state}, ${currentHotel.zip}
            </p>
            <p class="cite">
                <cite>${currentHotel.country}</cite>
            </p>
            <p>Precio por da: $${currentHotel.price}</p>
        </blockquote>
    </div>
</div>

<div class="grid_5">
    <s:form action="reservaHotel_confirm" namespace="/protected" method="POST">
        <sconv:conversationPropagation mode="join"/>
        <fieldset>
            <legend>Confirmar Hotel</legend>
            <p>
                <label>Ingreso: </label>
                <fmt:formatDate value="${reserva.checkinDate}" pattern="dd/MM/yyyy"/>
            </p>
            <p>
                <label>Egreso: </label>
                <fmt:formatDate value="${reserva.checkoutDate}" pattern="dd/MM/yyyy"/>
            </p>
            <p>
                <label>Camas: </label>
                <s:set var="mapaCamas" value="#{1:'Una cama king size', 2:'Dos camas matrimoniales', 3:'Tres camas simples'}"
                    />
                <s:property value="#mapaCamas[#attr.reserva.beds]"/>
            </p>
            <p>
                <label>Habitacin Fumador: </label>
                <s:set var="mapaSmoking" value="#{true:'Fumador', false:'No fumador'}" />
                <s:property value="#mapaSmoking[#attr.reserva.smoking]"/>
            </p>
            <p>
                Comodidades:
                <s:if test="#attr.reserva.amenities.isEmpty()">
                  No se han seleccionado comodidades
                </s:if>
                <s:else>
                  <s:property value="#attr.reserva.amenities.{description}"/>
                </s:else>
            </p>
            <p>
                <label>Medio de pago: </label>
                <s:property value="#{'VISA':'Visa', 'AMEX':'American Express', 'MASTERCARD':'MasterCard', 'DINERS':'Diners
                    '}[#attr.reserva.creditCardName]" />
            </p>
            <p>
                <label>Nro. Tarjeta de Crdito: </label>
                <s:property value="#attr.reserva.creditCardForShow" />
            </p>
            <p>
                <label>Fecha de expiracin: </label>
                <s:property value="#{1:'Enero',2:'Febrero',3:'Marzo',4:'Abril',5:'Mayo',6:'Junio',7:'Julio',8:'Agosto',9:'
                    Setiembre',10:'Octubre',11:'Noviembre',12:'Diciembre'}[#attr.reserva.creditCardExpiryMonth]" />
                /
                ${reserva.creditCardExpiryYear}
            </p>
        </fieldset>
        <s:submit name="btnSubmit" value="Reservar" cssClass="confirm button" />
        <s:submit name="btnRevisar" value="Reveer" cssClass="confirm button" action="reservaHotel_showForm" />
        <s:submit name="btnCancel" value="Cancelar" cssClass="confirm button" action="reservaHotel_cancel" />
    </s:form>
</div>
```

Listado A.75: reservaHotel_showConfirm.jsp

A.2.36. reservaHotel_showForm.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
<%@ taglib uri="/conversation-struts2-tags" prefix="sconv" %>
<%@ taglib uri="/struts-tags" prefix="s" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>

```

```
<sconv:useConversation/>

<div class="grid_3">
  <h2>${currentHotel.name}</h2>
  <div id="blockquote" class="block" style="margin: 0px;">
    <blockquote>
      <p>
        ${currentHotel.address}<br /> ${currentHotel.city}, ${currentHotel.state}, ${currentHotel.zip}
      </p>
      <p class="cite">
        <cite>${currentHotel.country}</cite>
      </p>
      <p>Precio por da: $${currentHotel.price}</p>
    </blockquote>
  </div>
</div>

<div class="grid_5">
  <sconv:form action="reservaHotel_showConfirm" namespace="/protected" includeContext="true" method="POST">
    <s:actionerror/>
    <fieldset>
        <s:fielderror fieldName="reserva.checkinDate" />
      <legend>Reservar Hotel</legend>
      <p>
        <label>Ingreso: </label>
        <fmt:formatDate var="fecha" value="${reserva.checkinDate}" pattern="dd/MM/yyyy"/>
        <s:textfield name="reserva.checkinDate" value="%{#attr.fecha}" maxlength="10" />
      </p>
      <p>
                <s:fielderror fieldName="reserva.checkoutDate" />
        <label>Egreso: </label>
                <fmt:formatDate var="fecha" value="${reserva.checkoutDate}" pattern="dd/MM/yyyy"/>
        <s:textfield name="reserva.checkoutDate" value="%{#attr.fecha}" maxlength="10" />
      </p>
      <p>
                <s:fielderror fieldName="reserva.beds" />
        <label>Camas: </label>
        <s:select name="reserva.beds" list="#{'1':'Una cama king size', '2':'Dos camas matrimoniales', '3':'Tres camas
             simples'}" />
      </p>
      <p style="text-align: center;">
                <s:fielderror fieldName="reserva.smoking" />
        <s:radio name="reserva.smoking" list="#{true:'Fumador', false:'No fumador'}" cssStyle="width: 1em" />
      </p>
      <p>
        Comodidades:
        <s:checkboxlist name="reserva.amenities"
            list="#attr.reserva.availableAmenities"
            listValue="description"
            cssStyle="width: 1em" />
      </p>
      <p>
                <s:fielderror fieldName="reserva.creditCardName" />
        <label>Medio de pago: </label>
        <s:select name="reserva.creditCardName" list="#{'VISA':'Visa', 'AMEX':'American Express', 'MASTERCARD':'MasterCard',
             'DINERS':'Diners'}" />
      </p>
      <p>
            <s:fielderror fieldName="reserva.creditCard" />
        <label>Nro. Tarjeta de Crdito: </label>
        <s:textfield name="reserva.creditCard" />
      </p>
      <p>
        <label>Fecha de expiracin: </label>
        <s:select name="reserva.creditCardExpiryMonth"
            list="#{1:'Enero',2:'Febrero',3:'Marzo',4:'Abril',5:'Mayo',6:'Junio',7:'Julio',8:'Agosto',9:'Setiembre',10:'
                 Octubre',11:'Noviembre',12:'Diciembre'}" />
        <s:select list="availableYearsCreditCard" name="reserva.creditCardExpiryYear" size="1" />
                <s:fielderror fieldName="reserva.creditCardExpiryMonth" />
                <s:fielderror fieldName="reserva.creditCardExpiryYear" />
      </p>
    </fieldset>
    <s:submit name="btnSubmit" value="Reservar" cssClass="confirm button" />
    <s:submit name="btnCancel" value="Cancelar" cssClass="confirm button" action="reservaHotel_cancel" />
  </sconv:form>
</div>
```

Listado A.76: reservaHotel_showForm.jsp

A.2.37. search.jsp

```
<%@ taglib uri="/struts-tags" prefix="s" %>
<%@ taglib prefix="security" uri="http://www.springframework.org/security/tags" %>
<%@ taglib uri="http://displaytag.sf.net" prefix="display" %>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>

<s:if test="hasActionErrors()">
  <s:actionerror/>
</s:if>

<s:actionmessage escape="false" />

        <div class="box">
          <h2>Busqueda de Hoteles:</h2>
          <div class="block" id="forms">

<s:form action="search_execute" namespace="/" includeContext="true" method="POST">
              <fieldset class="login">
                <legend>Bsqueda</legend>
                <p>
                  <label>Criterio: </label>
                  <s:textfield name="criteria" size="40" />
                </p>
                <s:submit name="btnSubmit" value="Buscar" cssClass="confirm button" />
              </fieldset>
</s:form>

          </div>
        </div>

<security:authorize ifAllGranted="ROLE_USER">
        <div class="box">
          <h2>Reservas de Hoteles realizadas:</h2>

```

```
34          <s:if test="bookingList.isEmpty()">
35            No se encontraron Reservas
36          </s:if>
37          <s:else>
38            <div class="clear"></div>
39
40            <display:table name="${bookingList}" id="booking" requestURI="">
41              <display:column property="hotel.name" title="Hotel" />
42              <display:column property="hotel.address" title="Direccin" />
43              <display:column title="Ciudad, Prov." >${booking.hotel.city}, ${booking.hotel.state}</display:column>
44              <display:column property="checkinDate" title="Ingreso" format="{0,date,dd/MM/yyyy}" />
45              <display:column property="checkoutDate" title="Egreso" format="{0,date,dd/MM/yyyy}" />
46              <display:column property="id" title="Id" />
47
48              <display:column title="Accin" >
49                <s:url var="urlCancel" action="cancelBooking" includeContext="true" includeParams="none" namespace="/
                      protected">
50                  <s:param name="bookingId">${booking.id}</s:param>
51                </s:url>
52                <a href="${urlCancel}" onclick='return confirm("Realmente desea cancelar la reserva en ${booking.hotel.name}?
                      ");'>Cancelar</a>
53              </display:column>
54            </display:table>
55          </s:else>
56        </div>
57 </security:authorize>
```

Listado A.77: search.jsp

A.2.38. search_result.jsp

```
1 <%@ taglib uri="/struts-tags" prefix="s" %>
2 <%@ taglib uri="http://displaytag.sf.net" prefix="display" %>
3 <%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
4
5         <div class="box">
6           <h2>Busqueda de Hoteles:</h2>
7           <div class="block" id="forms">
8
9 <s:form action="search_execute" namespace="/" includeContext="true" method="POST">
10              <fieldset class="login">
11                <legend>Bsqueda</legend>
12                <p>
13                  <label>Criterio: </label>
14                  <s:textfield name="criteria" size="40" />
15                </p>
16                <s:submit name="btnSubmit" value="Buscar" cssClass="confirm button" />
17              </fieldset>
18 </s:form>
19
20          </div>
21        </div>
22
23        <div class="clear"></div>
24
25        <display:table name="${hotelList}" id="hotel" requestURI="">
26          <display:column property="name" title="Nombre" sortable="true" />
27          <display:column property="address" title="Direccin" sortable="true" />
28          <display:column title="Ciudad, Prov." sortable="true">${hotel.city}, ${hotel.state}</display:column>
29          <display:column property="zip" title="CP" sortable="true" />
30          <display:column title="Accin" sortable="true" style="text-align: center">
31            <s:url var="urlVer" action="viewHotel" includeContext="true" includeParams="none" namespace="/">
32              <s:param name="hotelId">${hotel.id}</s:param>
33            </s:url>
34            <a href="${urlVer}">Ver</a>
35          </display:column>
36        </display:table>
```

Listado A.78: search_result.jsp

A.2.39. viewHotel.jsp

```
1 <%@ taglib uri="/struts-tags" prefix="s"%>
2 <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
3   pageEncoding="ISO-8859-1"%>
4
5 <div class="box">
6   <h2>${hotel.name}</h2>
7   <div id="blockquote" class="block" style="margin: 0px;">
8     <blockquote>
9       <p>
10        ${hotel.address}<br /> ${hotel.city}, ${hotel.state}, ${hotel.zip}
11      </p>
12      <p class="cite">
13        <cite>${hotel.country}</cite>
14      </p>
15      <p>Precio por da: $${hotel.price}</p>
16    </blockquote>
17  </div>
18  <s:form action="reservaHotel_showForm" namespace="/protected" includeContext="true" method="POST">
19    <input type="hidden" name="hotelId" value="${hotel.id}" />
20    <s:submit name="btnSubmit" value="Reservar" cssClass="confirm button" />
21  </s:form>
22 </div>
```

Listado A.79: viewHotel.jsp

A.2.40. applicationContext.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans
3   xmlns="http://www.springframework.org/schema/beans"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xmlns:context="http://www.springframework.org/schema/context"
6   xmlns:jdbc="http://www.springframework.org/schema/jdbc"
7   xmlns:jee="http://www.springframework.org/schema/jee"
```

```
  xmlns:lang="http://www.springframework.org/schema/lang"
  xmlns:p="http://www.springframework.org/schema/p"
  xmlns:task="http://www.springframework.org/schema/task"
  xmlns:tx="http://www.springframework.org/schema/tx"
  xmlns:util="http://www.springframework.org/schema/util"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans
      -3.0.xsd
    http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.0.xsd
    http://www.springframework.org/schema/jdbc http://www.springframework.org/schema/jdbc/spring-jdbc-3.0.xsd
    http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-jee-3.0.xsd
    http://www.springframework.org/schema/lang http://www.springframework.org/schema/lang/spring-lang-3.0.xsd
    http://www.springframework.org/schema/task http://www.springframework.org/schema/task/spring-task-3.0.xsd
    http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.0.xsd
    http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-3.0.xsd
    http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">

  <!-- post-processors for all standard config annotations -->
  <context:annotation-config/>

  <!--  INFRAESTRUCTURA  -->

  <!-- Deploys a in-memory "hotel" datasource populated -->
  <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="org.hsqldb.jdbc.JDBCDriver" />
    <property name="url" value="jdbc:hsqldb:mem:hotel" />
    <property name="username" value="sa" />
    <property name="password" value="" />
  </bean>

   <!-- Creates a EntityManagerFactory for use with the Hibernate JPA provider and a simple in-memory data source populated
       with test data -->
  <bean id="entityManagerFactory"
    class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="jpaVendorAdapter">
      <bean
        class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
        <property name="database" value="HSQL" />
        <property name="showSql" value="true" />
      </bean>
    </property>
    <!-- not possible with Tomcat (JNDI is read-only)
    <property name="jpaProperties">
      <value>
        hibernate.session_factory_name=mySessionFactory
      </value>
    </property>
    -->
  </bean>

  <bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
    <property name="entityManagerFactory" ref="entityManagerFactory"/>
  </bean>

<!--
  <tx:advice id="txAdvice">
    <tx:attributes>
      <tx:method name="get*" read-only="true" propagation="REQUIRED"/>
      <tx:method name="*" propagation="REQUIRED"/>
    </tx:attributes>
  </tx:advice>

  <aop:config>
    <aop:pointcut id="serviceMethods" expression="execution(* com.hexaid.examples.hotel.service.*.*(..))"/>
    <aop:advisor advice-ref="txAdvice" pointcut-ref="serviceMethods"/>
  </aop:config>
-->

  <!-- Spring security -->
  <import resource="security-config.xml"/>

  <!-- JSR 303 Validation -->
  <bean id="validator" class="org.springframework.validation.beanvalidation.LocalValidatorFactoryBean" />

  <!--  CONSTANTES  -->

  <!--  DAOs  -->

  <bean id="hotelDAO" class="com.hexaid.examples.hotel.dao.HotelDAO" scope="singleton" />
  <bean id="userDAO" class="com.hexaid.examples.hotel.dao.UserDAO" scope="singleton" />
  <bean id="bookingDAO" class="com.hexaid.examples.hotel.dao.BookingDAO" scope="singleton" />

  <!--  SERVICIOS  -->

  <bean id="hotelService" class="com.hexaid.examples.hotel.service.HotelService" scope="singleton" />
  <bean id="userService" class="com.hexaid.examples.hotel.service.UserService" scope="singleton" />
  <bean id="bookingService" class="com.hexaid.examples.hotel.service.BookingService" scope="singleton" />

</beans>
```

Listado A.80: applicationContext.xml

A.2.41. security-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:security="http://www.springframework.org/schema/security"
  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.
      xsd
    http://www.springframework.org/schema/security http://www.springframework.org/schema/security/spring-security.xsd">

  <security:http auto-config="true">
    <security:intercept-url pattern="/*action*" access="IS_AUTHENTICATED_ANONYMOUSLY"/>
    <security:intercept-url pattern="/css/**" filters="none"/>
    <security:intercept-url pattern="/img/**" filters="none"/>
    <security:intercept-url pattern="/js/**" filters="none"/>
    <security:intercept-url pattern="/protected/**" access="ROLE_USER" />
    <security:form-login
      login-page="/login.action"
      default-target-url="/index.action"
      authentication-failure-url="/login.action?login_error=1" />
    <security:logout/>
```

```
    <!-- cookie dura 15 dias -->
    <security:remember-me token-validity-seconds="1296000"/>
  </security:http>

  <security:authentication-manager>
    <security:authentication-provider>
      <security:user-service>
        <security:user name="gabriel" password="gabriel" authorities="ROLE_USER"/>
        <security:user name="javier" password="javier" authorities="ROLE_USER"/>
        <security:user name="claudia" password="claudia" authorities="ROLE_USER"/>
      </security:user-service>
    </security:authentication-provider>
  </security:authentication-manager>

</beans>
```

Listado A.81: security-config.xml

A.2.42. tiles.xml

```
<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE tiles-definitions PUBLIC
       "-//Apache Software Foundation//DTD Tiles Configuration 2.1//EN"
       "http://tiles.apache.org/dtds/tiles-config_2_1.dtd">
<tiles-definitions>
  <definition name="baseLayout" template="/BaseLayout.jsp">
    <put-attribute name="title" value="" />

    <put-attribute name="header" value="/Header.jsp" />
    <put-attribute name="menu" value="/Menu.jsp" />

    <put-attribute name="body" value="" />
    <put-attribute name="footer" value="/Footer.jsp" />
  </definition>

  <definition name="/exception.tiles" extends="baseLayout">
    <put-attribute name="title" value=" - Ha ocurrido un error" />
    <put-attribute name="body" value="/exception.jsp" />
  </definition>

  <definition name="/index.tiles" extends="baseLayout">
    <put-attribute name="title" value="" />
    <put-attribute name="body" value="/index.jsp" />
  </definition>

  <definition name="/search.tiles" extends="baseLayout">
    <put-attribute name="title" value=" - Búsqueda" />
    <put-attribute name="body" value="/search.jsp" />
  </definition>

  <definition name="/search_result.tiles" extends="baseLayout">
    <put-attribute name="title" value=" - Resultados Búsqueda" />
    <put-attribute name="body" value="/search_result.jsp" />
  </definition>

  <definition name="/viewHotel.tiles" extends="baseLayout">
    <put-attribute name="title" value=" - Detalles del Hotel" />
    <put-attribute name="body" value="/viewHotel.jsp" />
  </definition>

  <definition name="/login.tiles" extends="baseLayout">
    <put-attribute name="title" value=" - Ingreso" />
    <put-attribute name="body" value="/login.jsp" />
  </definition>

  <definition name="/infoAutor.tiles" extends="baseLayout">
    <put-attribute name="title" value=" - Información sobre la Tesina y su Autor" />
    <put-attribute name="menu" value="/fotoAutor.jsp" />
    <put-attribute name="body" value="/infoAutor.jsp" />
  </definition>

  <definition name="/protected/reservaHotel_showForm.tiles" extends="baseLayout">
    <put-attribute name="title" value=" - Paso 1 - Ingrese los datos de su reserva" />
    <put-attribute name="body" value="/protected/reservaHotel_showForm.jsp" />
  </definition>

  <definition name="/protected/reservaHotel_showConfirm.tiles" extends="baseLayout">
    <put-attribute name="title" value=" - Paso 2 - Confirme los datos de su reserva" />
    <put-attribute name="body" value="/protected/reservaHotel_showConfirm.jsp" />
  </definition>

</tiles-definitions>
```

Listado A.82: tiles.xml

A.2.43. web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    version="2.5">
  <display-name>Hexaid's Hotel example featuring conversations (${project.artifactId}-${project.version})</display-name>

  <filter>
    <filter-name>springSecurityFilterChain</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
  </filter>

  <filter>
    <filter-name>struts2</filter-name>
    <filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>
  </filter>

  <filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

  <filter-mapping>
```

```
24      <filter-name>struts2</filter-name>
25      <url-pattern>/*</url-pattern>
26    </filter-mapping>
27
28    <listener>
29      <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
30    </listener>
31    <listener>
32      <listener-class>org.apache.struts2.tiles.StrutsTilesListener</listener-class>
33    </listener>
34
35    <context-param>
36      <param-name>tilesDefinitions</param-name>
37      <param-value>/WEB-INF/tiles.xml</param-value>
38    </context-param>
39
40    <welcome-file-list>
41      <welcome-file>index.action</welcome-file>
42      <welcome-file>index.html</welcome-file>
43    </welcome-file-list>
44  </web-app>
```

Listado A.83: web.xml

Bibliografía

[1] Deepak Alur, Dan Malks, and John Crupi. *Core J2EE Patterns: Best Practices and Design Strategies (2nd Edition)*. Prentice Hall, 2 edition, 2003.

[2] Apache. Apache maven. Sitio Web, http://maven.apache.org/.

[3] Apache. Apache tiles. Sitio Web, http://tiles.apache.org/.

[4] ASF. Struts 1. Sitio Web, http://struts.apache.org.

[5] ASF. Struts 2. Sitio Web, http://struts.apache.org.

[6] ASF. Struts 2 documentation. Sitio Web, http://struts.apache.org/2.2.3/docs/home.html.

[7] Christian Bauer and Gavin King. *Java Persistence with Hibernate*. Manning, revised. edition, 2006.

[8] Kent Beck. Junit. Sitio Web, http://www.junit.org/.

[9] Codehaus. Mvel. Sitio Web, http://mvel.codehaus.org/.

[10] World Wide Web Consortium. The original http as defined in 1991. Sitio Web, http://www.w3.org/Protocols/HTTP/AsImplemented.html.

[11] Danny Coward and Yutaka Yoshida. Java servlet specification 2.4. JCP Specification, November 2003.

[12] Pierre Delisle, Mark Roth Jan Luehe, and Kin man Chung. Javaserver pages, specification version 2.2, maintenace release 2. Technical report, Sun Microsystems, December 2009.

[13] Matt Doyle. Preventing multiple form submits. Sitio Web, http://www.elated.com/articles/preventing-multiple-form-submits/.

[14] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol – http/1.1. RFC 2616 (Draft Standard), June 1999. Updated by RFC 2817.

[15] Apache Software Foundation. Tomcat. Sitio Web, http://tomcat.apache.org.

[16] Codehaus Foundation. Jetty. Sitio Web, http://jetty.codehaus.org/jetty.

[17] FreeMarker. Freemarker. Sitio Web, http://freemarker.sourceforge.net/.

[18] Erich Gamma, Richard Helm, Ralph E. Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.

[19] HTML Goodies. Disabling the back button. Sitio Web, http://www.htmlgoodies.com/tutorials/buttons/article.php/3478911/Disabling-the-Back-Button.htm.

[20] Google. Google guice. Sitio Web, http://code.google.com/p/google-guice/.

[21] JBoss. Hibernate. Sitio Web, http://www.hibernate.org.

[22] JBoss. jbpm. Sitio Web, http://www.jboss.org/jbpm.

[23] JBoss. Patrón open session in view. Sitio Web, https://community.jboss.org/wiki/OpenSessionInView.

[24] JBoss. The seam framework. Sitio Web, http://www.seamframework.org.

[25] Michael Jouravlev. Redirect after post, August 2004. Sitio Web, http://www.theserverside.com/news/1365146/Redirect-After-Post.

[26] OGNL. Object-graph navigation system. Sitio Web, http://www.opensymphony.com/ognl/.

[27] OpenJS. Context menus in web application using javascript. Sitio Web, http://www.openjs.com/scripts/ui/context_menu/.

[28] OpenSymphony. Webwork. Sitio Web, http://www.opensymphony.com/webwork.

[29] Oracle. The java ee 5 tutorial. Sitio Web, http://download.oracle.com/docs/cd/E17477_01/javaee/5/tutorial/doc/.

[30] Oracle. Java server faces. Sitio Web, http://java.sun.com/javaee/javaserverfaces.

[31] Oracle. Java transaction api. Sitio Web, http://www.oracle.com/technetwork/java/javaee/jta/index.html.

[32] Oracle. Jsr 317: Java persistence 2.0. Sitio Web, http://www.jcp.org/en/jsr/detail?id=317.

[33] Sitemesh. Sitemesh framework. Sitio Web, http://www.sitemesh.org/.

[34] SpringSource. Spring framework. Sitio Web, http://www.springsource.org.

[35] SpringSource. Spring security framework. Sitio Web, http://www.springsource.org/spring-security.

[36] Springsource. Spring web flow. Sitio Web, http://www.springsource.org/go-webflow2.

[37] Welie.com. A pattern library for interaction design. Sitio Web, http://www.welie.com/.

[38] Wikipedia. Continuation. Sitio Web, http://en.wikipedia.org/wiki/Continuation.

[39] Wikipedia. Html. Sitio Web, http://en.wikipedia.org/wiki/HTML.

[40] Wikipedia. Process control block. Sitio Web, http://en.wikipedia.org/wiki/Process_control_block.

[41] Wikipedia. Representational state transfer. Sitio Web, http://en.wikipedia.org/wiki/Representational_state_transfer.

[42] Yahoo. Yui library. Sitio Web, http://http://developer.yahoo.com/yui/.