8 AMPLIACIÓN DE LA APLICACIÓN: SERVICIOS VÍA SMS

Hemos implementado una aplicación J2EE, utilizando el modelo MVC, usando el framework de Struts y basándonos en unos patrones de diseño. Con esto se ha pretendido mostrar los beneficios de la modularidad y la separación de capas para mantenimiento y reusabilidad de la aplicación.

Se han hecho continuas referencias a posibles cambios en la aplicación, como las fuentes de datos o transformarla en una aplicación distribuida, y la facilidad con la que se pueden afrontar estos cambios gracias a las tecnologías y los patrones de diseño utilizados.

También se ha hecho hincapié en la importancia de la separación de roles de diseñadores de la vista de la aplicación y los programadores responsables del modelo. Se ha mostrado la facilidad para cambiar la vista de sin tener que modificar el código del modelo de la aplicación.

Ahora pretendemos ampliar la aplicación original para que incluya un servicio de SMS al través del cual se podrán acceder a los mismos datos y realizar las mismas operaciones que se podían realizar desde el navegador.

En este capítulo se realizará el diseño y desarrollo la ampliación para la aplicación ejemplo desarrollada en 6 APLICACIÓN EJEMPLO CON J2EE, para ofrecer servicios vía SMS.

El objetivo de esta ampliación es mejorar las prestaciones de la aplicación ejemplo en movilidad, permitiendo el acceso a datos y realización de operaciones, no sólo desde cualquier

dispositivo con acceso a Internet, si no también desde cualquier dispositivo móvil con posibilidad de recibir y enviar SMS. Sería por tanto accesible desde cualquier teléfono móvil de hoy en día.

Además, el desarrollo de esta ampliación, nos permitirá valorar las características de escalabilidad que exigíamos a la aplicación ejemplo.

8.1 ANÁLISIS DE OBJETIVOS Y NECESIDADES DE LA AMPLIACIÓN

La aplicación ejemplo desarrolla en capítulos anteriores, proporciona un nivel básico de movilidad. Las aplicaciones empresariales que ofrecen servicios vía Web, proporcionan el acceso inmediato a datos siempre y cuando se disponga de un ordenador con acceso a Internet.

El objeto del desarrollo de esta ampliación es conseguir unos servicios realmente competitivos en el mercado actual, aportando un mayor grado de movilidad a estos servicios.

La movilidad, además de proporcionar mayor facilidad y comodidad al usuario, en entornos empresariales, pueden reducir tiempos y costos en desplazamientos, además de un acceso a datos y operaciones, prácticamente en tiempo real.

El nivel de escalabilidad conseguido en el desarrollo de la aplicación Web, nos beneficiará a la hora de realizar esta ampliación. Se obtendrá mucho mayor grado de movilidad, con un impacto mínimo sobre la aplicación existente.

En el desarrollo de actualizaciones, ampliaciones y migraciones, el tiempo es un factor crítico, por lo que es imprescindible el desarrollo desde el inicio de una aplicación con un factor de escalabilidad elevado. El objetivo de escalabilidad que imponíamos a la aplicación ejemplo, se podrá valorar con el desarrollo de esta ampliación.

8.2 ÁMBITO Y ALCANCE

La ampliación del servicio se orientará sólo al servicio SMS. Tanto para la notificación de operaciones realizadas desde la Web, como para la realización de operaciones desde el móvil.

No aportará nuevas operaciones, la ampliación tiene como objetivo ofrecer los servicios existente a través de otra medio para conseguir una mayor accesibilidad y movilidad. Aunque se podría ofrecer un servicio sustitutivo del servicio Web, para nuestra aplicación, y para la mayoría de los casos, se buscará un complemento del servicio existente que proporcione mayor facilidad y comodidad a los usuarios.

Ofreceremos un servicio de solicitud de vacaciones vía SMS. Este servicio consistirá en operaciones de solicitud de vacaciones, consulta del estado de una solicitud existente y cancelación de una solicitud si no está tramitada. Además, este servicio se complementará con un servicio de notificación de tramitación de vacaciones y permitirá a los usuarios cambiar a través de SMS, el teléfono en el que desean recibir las notificaciones. Estos servicios ya son proporcionados a través de la Web por nuestra aplicación ejemplo.

Nos centramos en los servicios anteriormente mencionados por considerarlos de mayor prioridad en el conjunto de servicios prestados vía Web. En nuestro ejemplo, podrían ofrecerse todos los servicios de la Web a través de SMS, sin embargo, para cada caso concreto que pueda surgir en un entorno real, se recomienda realizar un estudio de la necesidad de movilidad para cada operación. No se trata de conseguir movilidad por movilidad, sino de conseguir una movilidad que nos proporcione mayores beneficios, facilidad de acceso y competitividad.

8.2.1 Utilidad del servicio

Aunque cada vez más usuarios disponen de un ordenador portátil y conexión a Internet, no siempre, cuando surge la necesidad de realizar una operación, ambos están disponibles. Además, a veces, simplemente por comodidad, sería interesante no depender de ellos.

Hoy en día existen otros dispositivos móviles que pueden acceder a Internet, ya sea a través de WIFI o por UMTS. No nos centramos en este tipo de conexiones ya que adaptar nuestra aplicación para proporcionar servicios a estos dispositivos sólo supondría una ampliación o modificación en la vista para cumplir los requisitos que las pequeñas pantallas de estos dispositivos imponen. Además, actualmente el acceso a través de UMTS no es demasiado rentable.

El servicio SMS es un servicio con un coste aceptable y con la capacidad de llegar a todo tipos de usuarios, tanto dentro de una empresa como para servicios orientados a clientes finales. La extensión de este servicio es una realidad. Con la ampliación para proporcionar servicios SMS, se consigue realmente los objetivos de en cualquier momento, en cualquier lugar. Con este servicio, los datos y operaciones realmente estarán al alcance de la mano.

8.3 IDENTIFICACIÓN DE REQUISITOS

Los servicios SMS que ofrecerá la ampliación, en general, serán un subconjunto de los servicios ofrecidos vía Web. Se complementará, además, con un servicio de notificación de tramitación de vacaciones.

8.3.1 Agentes

Aunque podríamos implementar todos los servicios proporcionados en la Web, nos centraremos en algunos de los servicios que se proporcionan a un usuario capaz de identificarse. No implementaremos los servicios exclusivos del administrador.

Por ejemplo, aunque se podría realizar, no tendría mucho sentido la implementación de un servicio de tramitación de vacaciones vía SMS. Normalmente, el usuario encargado de estas tramitaciones tendrá que realizar un estudio previo de la situación antes de conceder o no una petición, por lo que sería algo irreal, o al menos poco útil, ofrecer este servicio con las limitaciones que un SMS supone.

Por otra parte, el servicio de notificación de operaciones, aunque está orientado a los usuarios, sólo podrá realizarse si el agente de la operación es un administrador. Es decir, aquellas operaciones que se notificarán a los usuarios, serán operaciones realizadas por el administrador. No tendría mucho sentido notificar al usuario una operación que él mismo ha realizado.

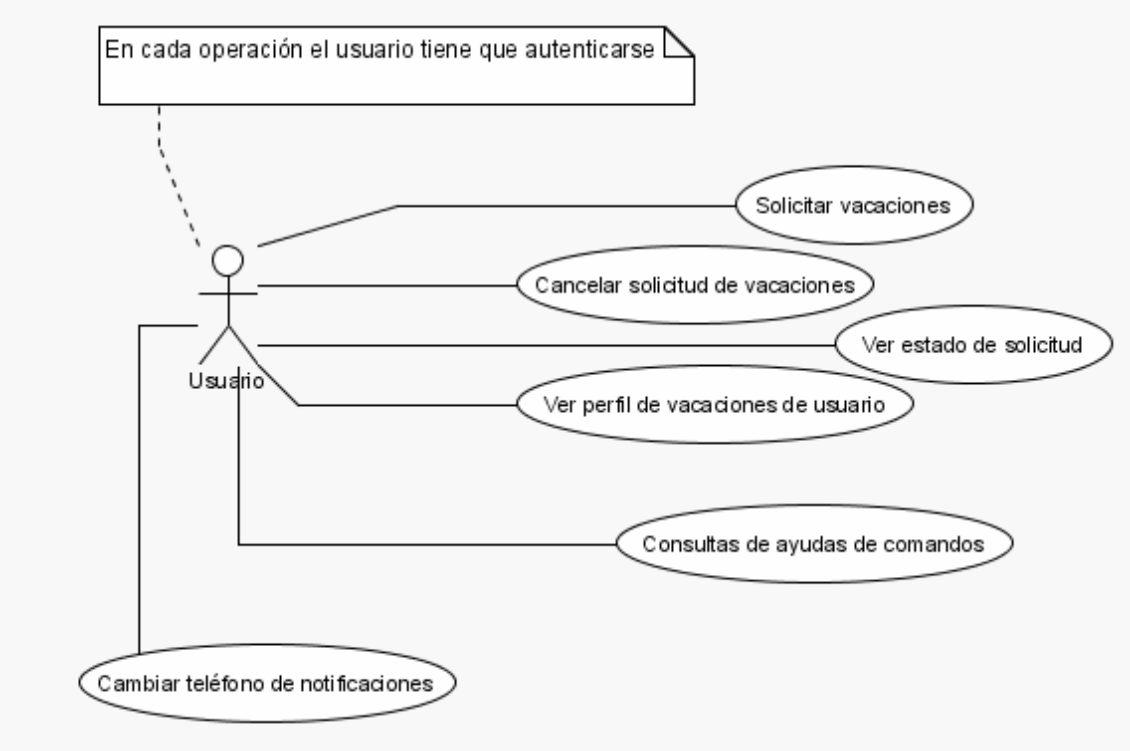
Cuando hablamos de notificación, nos centramos en la notificación de operaciones realizadas desde la Web. El mensaje SMS tendría su origen en la Web. Cuando un usuario realiza una operación vía SMS, realizará un mensaje de éxito o error de la operación, sin embargo esto no lo consideraremos notificación, sino respuesta.

Como ejemplo, implementaremos el servicio de notificación de tramitación de vacaciones. Cuando un administrador tramita una solicitud de vacaciones, se podrá enviar un mensaje al usuario informándole si su petición a sido aceptada o no.

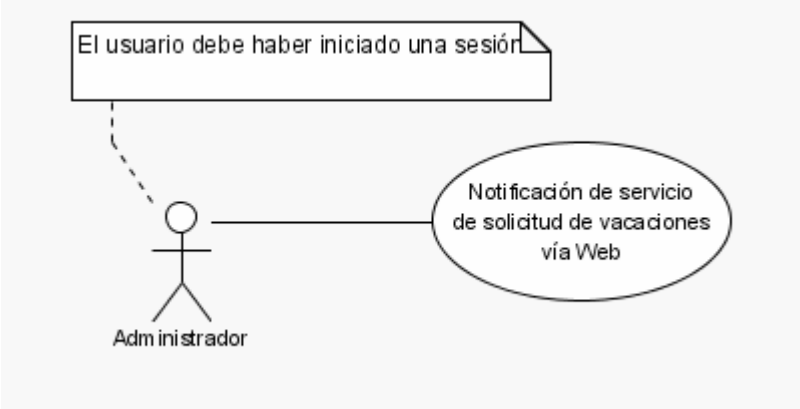
8.3.2 Actividades

Un usuario podrá realizar las operaciones de solicitud de vacaciones, cancelación de solicitud de vacaciones, y consulta sobre una petición de vacaciones a través de SMS. También podrá cambiar el número de teléfono activo para notificaciones.

El administrador podrá enviar una notificación de tramitación de solicitud de vacaciones a los usuarios.



F. 8-1: Ampliación de la aplicación ejemplo. Actividades y agentes operaciones vía SMS.



F. 8-2: Ampliación de la aplicación ejemplo. Actividades y agentes notificación SMS desde Web.

Cada usuario que pretenda realizar una operación SMS, deberá estar registrado en el sistema y aportar su identificador de usuario y contraseña para poder realizar la acción correspondiente.

Al igual que para el servicio vía Web, para realizar una petición de vacaciones, el usuario debe estar dado de alta en el servicio de vacaciones para el período solicitado.

Un usuario registrado, podrá cambiar a través de SMS el número de teléfono en el que desea recibir las notificaciones. Podrá asignar un número si no existía ninguno asignado, modificar el existente, o borrar el número existente.

Al tramitar una solicitud, el administrador tendrá la opción de enviar o no notificación. Si se decide enviar, sólo se enviará si el usuario a aportado un teléfono para el servicio de notificación. Este teléfono es el que forma parte del perfil de usuario como datos de contacto.

8.3.3 Objetos de datos

En el desarrollo de esta ampliación no será necesaria la modificación de los objetos de datos existentes, ni la creación de nuevos objetos de datos. Ofrecemos un subconjunto de servicios ya ofrecidos anteriormente, por lo que el impacto en los objectos de datos es nulo.

8.4 ARQUITECTURA TECNOLÓGICA

En el capítulo 6 APLICACIÓN EJEMPLO CON J2EE, en el apartado 6.4 ARQUITECTURA TECNOLÓGICA, se definió la arquitectura tecnológica de la aplicación completa, incluyendo el servicio de SMS (F. 6-10: Aplicación ejemplo. Arquitectura tecnológica completa.).

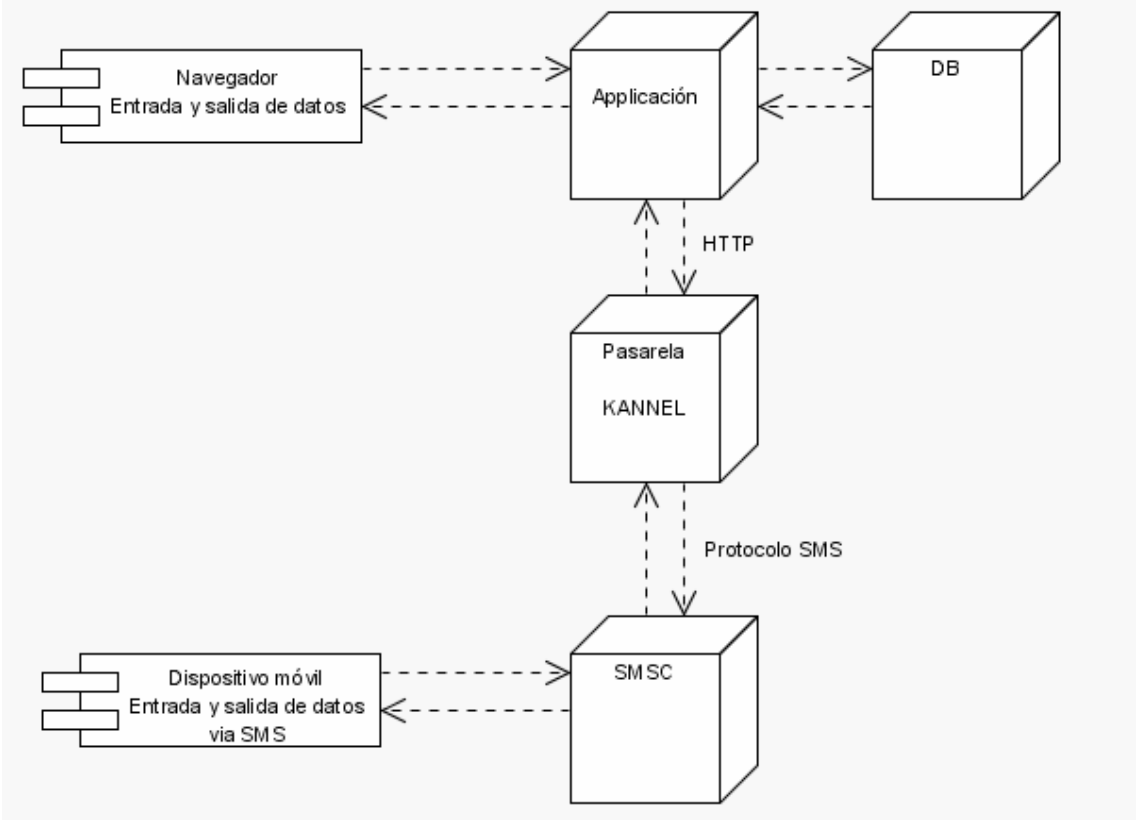
Los datos se almacenan en la base de datos y se acceden a ellos a través de una aplicación Web hospedada en el servidor Web (para el ejemplo utilizamos un contenedor Web que puede trabajar independientemente, sin necesidad de servidor). La entrada y salida de datos se realizará por peticiones HTTP.

Para el servicio SMS utilizaremos una pasarela de SMS. Esta pasarela, a través de un modem GSM o UMTS recibe los mensajes y los interpreta realizando la operación configurada en el fichero de configuración. Para la salidad de datos, el servidor recibe peticiones HTTP con los parámetros adecuados y los envía formando un SMS que será transmitido a través del moden GSM o UMTS a la red GSM o UMTS.

8.4.1 Implementación

Al igual que para el resto de la aplicación intentarán utilizarse, en la medida de lo posible, herramientas e implementaciones de Software Libre, siempre y cuando esto no suponga consecuencias negativas en cuanto potencia de la aplicación.

La implementación escogida ya se definió en el apartado 6.4.4 del capítulo 6 APLICACIÓN EJEMPLO CON J2EE y sobre esta arquitectura se ha basado todo el desarrollo de la aplicación ejemplo. Para la realización de la ampliación, recordar que como pasarela de SMS se escogió Kannel 1.4.0.



F. 8-3: Ampliación de aplicación ejemplo. Arquitectura tecnológica completa.

La pasarela Kannel ha sido desarollada en sistemas Linux escrita en lenguaje C. La distribución proporciona las fuentes y para su utilización requiere un compilador de C y otras librerías y herramientas de desarrollo. Para su utilización en Windows se recomienda la utilización de Cygwin. Para nuestro desarrollo utilizamos la versión Cygwin_nt-5.1 de Cygwin.

8.4.2 Estructura general y capa del modelo

El servicio que pretendemos proporcionar con esta ampliación, es el mismo servicio que ofrecíamos a través de la Web, pero a través de SMS.

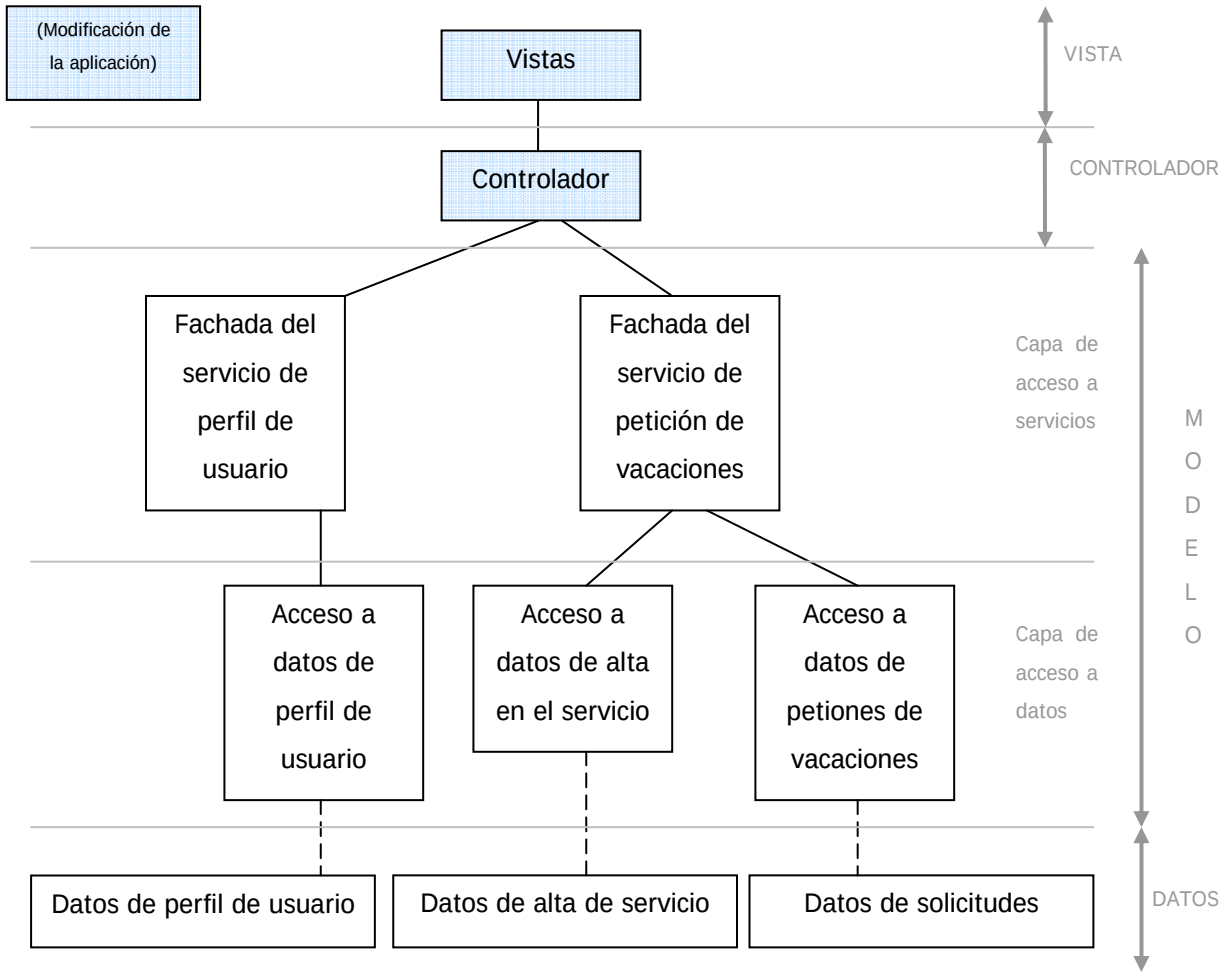
Puesto que la arquitectura J2EE se basa en un diseño de capas, utilizando el modelo MVC, y los patrones de diseño Core J2EE Patterns, se obtienen diseños que admiten diversidad de clientes con un impacto nulo en las capas del modelo.

Aunque la comunicación de Kannel con la aplicación se realiza con peticiones HTTP al igual que desde un navegador, surgen sin embargo algunas diferencias que afectarán a las capas de vista y controlador.

Las necesidades de nuevos elementos en la capa de la vista, surge principalmente por el medio de comunicación que utilizaremos. Aunque se Kannel con la aplicación se comunica a través de una intranet o Internet con HTTP, la comunicación con Kannel y los dispositivos se realizará a través de SMS, lo que supone un límite de unos 160 caracteres por mensaje. Por este motivo crearemos nuevos elementos de vista para la interacción con Kannel, basadas principalmente en texto.

Cuando un usuario utiliza la Web a través del navegador para interactuar con la aplicación, el usuario inicia una sesión que se mantedrá mientras el usuario realiza las distintas operaciones del servicio de perfil de usuario y del servicio de solicitud de vacaciones. Sin embargo, la comunicación a través de SMS es una comunicación orientada a petición/respuesta, no tiene por tanto sentido iniciar una sesión por cada usuario. Esta diferencias de enfoques son las que

provocan la necesidad de añadir nuevos elementos al controlador que maneje el control de recepción y envío de SMS.



F. 8-4: Modificaciones necesarias para la ampliación de la aplicación para proporcionar servcios a través de SMS y notificaciones.

En la figura F. 8-4: Modificaciones necesarias para la ampliación de la aplicación para proporcionar servcios a través de SMS y notificaciones., se muestra el impacto de la ampliación en la aplicación ejemplo inicial.

No utilizaremos ningún filtro para las operaciones realizadas desde SMS, la acción de filtrado se realiza al autentificar al usuario con los métodos de `SmsManager`. Sin embargo, utilizaremos para estas operaciones el patrón `Sms/*`.

8.4.3 Controlador

Como se ha comentado anteriormente, la nueva ampliación está formado por dos partes: servicio de notificación de operaciones a través de SMS; y servicio de solicitud de vacaciones vía SMS.

El servicio de notificación de operaciones debe integrarse con el servicio de solicitud de vacaciones existente. Para ello será necesaria la modificación de las classes Action involucradas para la integración del envio de la notificación. Este servicio además necesitará la integración de nuevos elementos en la capa de la vista en la interfaz de usuario.

Para el servicio de operaciones SMS se crearán nuevas clases Action con las nuevas operaciones necesarias y nuevas páginas JSP que proporcionen una visto sin elementos gráficos.

Para ambos servicios se utilizará una nueva fachada para encapsular la complejidad de las interacciones entre los objetos del modelo proporcionando una capa de acceso a los servicios uniforme.

8.4.3.1 Session Façade

En la aplicación ejemplo sólo utilizabamos como clientes los navegadores. Con ellos un usuario establecía una sesion de usuario y se creó una implementación del patrón Session Façade para este tipo de clientes. Aunque Session Façade es el nombre del patrón utilizado, no tiene que

estar orientado al manejo de una sesión, sino para encapsular la complejidad de las interacciones entre los objetos del modelo que participan en el flujo de operaciones proporcionando un servicio uniforme de acceso a los clientes.

Con la ampliación utilizamos un nuevo cliente, el pasarela de SMS, Kannel. Desde este cliente acceden los distintos usuarios al sistema a través de SMS. Puesto que los SMS no son orientados a conexión, no tiene sentido que cada usuario establezca una sesión al acceder a la aplicación. Tampoco tiene sentido el establecimiento de una sesión del pasarela de SMS. Es por esto que en la misma petición, se realizará una autenticación de usuario y la operación a realizar. Cada petición será independiente de la anterior, ya que cada petición puede ser originada por usuarios distintos.

Se crea una nueva implementación del patrón Session Façade para el manejo de la recepción y envío de mensajes a través de Kannel, `SmsManager`. Cada método de la fachada relacionado con notificación de operaciones (en nuestra ampliación sólo `reportUpdateHolidaysRequest(HttpServletRequest request, String phone, Integer autoId, Short state)`) crea su mensaje de notificación, y con el método `storeSmsToSend(HttpServletRequest request, Collection<SmsVO> smsVOs)` se guarda una collección de mensajes a enviar en la sesión. Puesto que el servicio de notificaión sólo se ha creado para la notificación de operaciones realizadas por el administrador desde la Web, tiene sentido utilizar una variable de sesión, ya que estos usuario sí tendrán establecida una sesión cuando utilicen este servicio.

Para el envío de mensajes creamos un Object Value, `SmsVO`, que guardará el número de destinatario del mensaje y el cuerpo del mensaje a enviar.

Cada uno de los métodos relacionados con la recepción de mensajes actúan como fachada para acceder a los distintos servicios proporcionados por el modelo, para ello cada método se inicia con una identificación de usuario y, a continuación, si no hay errores en la identificación, se realiza la operación deseada.

`SmsManager` y `SmsVO` se encuentran en los paquetas `application.http.controller.sms`.

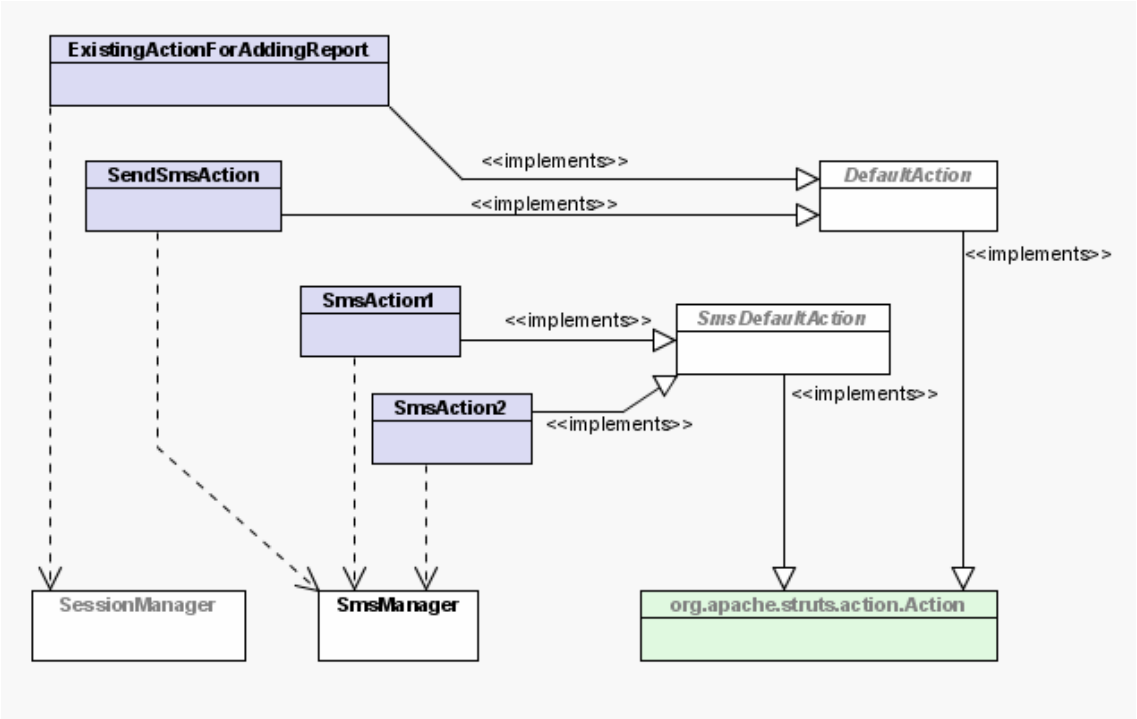
8.4.3.2 Struts – FrontController

Debido a los nuevos servicios ofrecidos son necesarias nuevas acciones definidas en el fichero de configuración de Struts junto a sus respectivas clases Action.

Además, serán necesarios nuevos formularios para la operaciones realizadas desde SMS que incluyan campos para la identificación de usuario. También tendrán que definirse las validaciones para estos formularios.

Se crean acciones para la solicitud de vacaciones (`SmsRegisterHolidaysRequestAction`), cancelación de solicitud de vacaciones (`SmsCancelHolidaysRequestAction`), consulta de solicitud de vacaciones (`SmsEditUserHolidaysRequestAction`) y consulta del perfil de vacaciones del usuario para un período (`SmsEditUserPeriodHolidaysRequestAction`). También se ofrece la posibilidad al usuario de cambiar su número de teléfono para la recepción de notificaciones (`SmsChangePhoneAction`).

Esta acciones implementará la nueva clase base `SmsDefaultAction`, que en caso de error interno, redireccionarán a `"smsInternalError"` que definirermos en Struts entre las redirecciones globales (`<global-forwards>`).



F. 8-5: Implementación de nuevos Actions para la ampliación de la aplicación ejemplo.

Por último, para el envío de notificaciones desde la Web, se crea `SendSmsAction`. Puesto que sólo los administradores están autorizados para realizar esta acción, `SendSmsAction` implementará `AdministratorDefaultAction`.

El servicio de notificación requiere también la modificación de las clases Actions implicadas. En nuestro ejemplo sólo afecta a `MapUpdatesHolidaysRequestAction`.

```
//MapUpdatesHolidaysRequestAction

//Resto de código

            try {
            SessionManager.updateHolidaysRequest(request,
                  Integer.decode(autoId), Short.decode(state));

            /*SMS -- */
            HolidaysRequestVO holidaysRequestVO =
                  SessionManager.findHolidaysRequest(request,
                        Integer.decode(autoId));
            String phone = SessionManager.findUser(request,
                  holidaysRequestVO.getLoginName()).getPhone();

            if(phone != null){
            SmsVO smsVO =
                  SmsManager.reportUpdateHolidaysRequest(request,
                  phone, Integer.decode(autoId), Short.decode(state));

            smsVOs.add(smsVO);
            }
            /* -- SMS*/

            } catch (InstanceNotFoundException e) {
                throw new InternalErrorException(e);
            }

//Resto de código
```

Bloque de código 8-1: Modificación de Actions para el servicio de notificación vía SMS.

Para integrar el nuevo servicio de notificación modificamos el flujo de acciones en `struts-config.xml`. Una vez realizada la acción se redirecciona a una pantalla de opción de envío de notificación. Si el usuario decide enviar la notificación lo hace a través de la página `kannel.jsp`, si no redirecciona a la página de éxito de la operación.

8.4.3.3 Validator – View Helper

Se definen las validaciones para los nuevos formularios. Los formularios creados para la realización de operaciones desde el móvil deben definir los campos de identificador de usuario y contraseña como obligatorios y con las mismas características que en el resto de los formularios, entre 3 y 9 caracteres alfanuméricos.

8.4.3.4 Tiles – Composite View

Se crea una nueva plantilla para las operaciones realizadas por SMS, `.smsDefaultTemplate`, y se definen las páginas asociadas de éxito de la operación `.smsSuccess`, errores en los comandos `.smsInput`, y error interno `.smsInternalError` utilizando esta plantilla.

Para el servicio de notificación se crea la página de opción de envio de notificación `.sendReport` y la página de envío de notificación `.kannel`. con la plantilla `.defaultTemplate` que utilizamos para el resto de páginas de la vista Web.

También se crean las vista Web y de impresión del manual del servcio de SMS, y la página que se enviará como respuesta a SMS de solicitud de ayuda de operaciones.

8.4.4 Vista

En cuanto a la vista, esta se ve afectada principalmente por la necesidad de incorporar nuevos elementos para la integración de los nuevos servicios.

Destacar del uso de páginas de errores en JSP, la librería de Struts `struts-bean.tld` y la creación de etiquetas personalizada.

Puesto que queremos ofrecer vistas de sólo texto no necesitaremos utilizar hojas de estilo para esta vista.

8.4.4.1 Páginas JSP

La especificación de Servlets nos permite definir páginas de error, a las que se redireccionará en caso de lanzarse alguna excepción de error, en el fichero `web.xml`.

```
<!-- web.xml -->

<web-apps>

<!-- ... -->

<error-page>
<error-code>404</error-code>
<location>/404.html</location>
</error-page>

<!-- ... -->

</web-apps>
```

Bloque de código 8-2: Definición de páginas de error en web.xml.

También la especificación de JSP permite definir páginas de error con la directiva `<%@page %>`. Si se define una página de error para una página en concreto, no redireccionará a la pagina definida en el fichero web.xml en caso de haberse definido.

Cuando una página es una página de error, debe utilizarse la directiva `page` con el atributo `isErrorPage`. Cuando en una página se definir la página a la que redireccionar en caso de error se utiliza la directiva `page` con el atributo `errorPage`.

Aunque en nuestra aplicación no utilizamos las páginas de error definidas en `web.xml`, para la página de envío de mensajes si utilizamos la definición de páginas de error que proporciona JSP. La página de envio de mensajes es la página `kannel.jsp`, esta página, en caso de poder conectar con el pasarela de SMS, lanzará una excepción de error. Para evitar mostrar este error en la conexión al usuario, se ha definido la página JSP `kannel_error.jsp`, que que mostrará el mensaje de error de conexión con una vista acorde con el resto la vista Web de la aplicación.

```
<!-- kannel.jsp -->
<%@page errorPage="/HTML/kannel_error.jsp"%>

<!-- kannel_error.jsp -->
<%@page isErrorPage="true"%>
```

Bloque de código 8-3: Definición de páginas de error en JSP.

Estas páginas de error no se mostrarán en caso de un error interno de nuestra aplicación ya que nosotros capturamos la excepciones de error interno, `ErrorInternalException`, en las distintas Actions de Struts y redireccionamos a `"internalErrorPage"` o `"smsInternalErrorPage"` direcciones definidas en el fichero de configuración `struts-config.xml`.

8.4.4.2 Librerías de etiquetas de Struts

Además de las librerías de etiquetas utilizadas en la aplicación inicial, para la aplicación ejemplo utilizaremos también las etiquetas `struts-bean.tld`.

Esta librería de etiquetas contiene utilidades para acceder a los distintos componentes y sus propiedades, y crear nuevos componentes a partir de éstos. Muchas de las etiquetas de esta

librería lanzarán una excepción JSP, `JspException`, en tiempo de ejecución cuando se produce algún error.

En concreto utilizaremos la etiqueta `<bean:include>` y `<bean:write>`. La primera es similar a la etiqueta estándar `<jsp:include>`, pero con la diferencia que la respuesta, el dato generado se almacena en un atributo. La URL a la que se pretende acceder, puede especificarse con el atributo `forward` para direcciones especificadas en `struts-config.xml` como `<global-fordwards>`, y con el atributo `href` se pueden especificar direcciones tanto internas como externas a la aplicación. También el atributo `page` permite definir URI relativas.

Para nuestra ampliación utilzaremos la etiqueta `<bean:include>` con el atributo `href`. Esta etiqueta nos permitirá definir la URL para acceder al servidor Kannel y almacenar la respuesta del servidor en el atributo especificado en el atributo `id` de la etiqueta.

Cuando se intente acceder al servidor Kannel utilizando esta etiqueta, si no se consigue establecer la conexión (porque, por ejemplo, esté fuera de servicio), se lanzará una excepción JSP, que se capturará utilizando la utilidad de JSP de definir páginas de errores con la directiva `<%@page%>`.

Para mostrar la respuesta capturada en el atributo especificado con `id`, utilizaremos la etiqueta `<bean:write>`, especificando con `name` el atributo a mostrar.

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jstl/fmt" %>
<%@ taglib prefix="bean" uri="http://struts.apache.org/tags-bean"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>

<%-- Definición de etiquetas personalizadas --%>
<%@taglib  prefix="my" uri="/mytags.tld" %>

<%@page errorPage="/HTML/kannel_error.jsp"%>

<blockquote>
```

```
<fmt:message key="success.content"/>

<c:choose> <c:when test="${empty requestScope.smss}">
<div class="row">No hay mensajes para enviar</div></c:when>

<c:otherwise>

<p><strong><fmt:message key="kannel.report.sms"/> : </strong></p>

<c:forEach var="sms" items="${requestScope.smss}">
<p><c:out value="${sms.smsTo}"/> :"<my:prepareSms
action="update.request"
compare="update.request" text="${sms.smsBody}" outvar="out"/>"
<bean:include id="kannel"
href="http://localhost:13013/sendsms?username=kannel&password=kannel&
to=${sms.smsTo}&text=${out}"/>
<bean:write name="kannel"/></p>

</c:forEach>

</c:otherwise></c:choose>

<p><b><fmt:message key="kannel.report.serverstatus"/></b> : </p>
<p><bean:include id="status"
href="http://localhost:13000/status.txt"/>
<bean:write name="status"/></p>
<br><br>
</blockquote>
```

Bloque de código 8-4: Uso de la librería de etiquetas struts-bean.tld para la comunicación con Kannel.

8.4.4.3 Librerías de etiquetas personalizadas

Para la ampliación hemos querido crear un ejemplo de etiqueta personalizada.

El cuerpo del mensaje a enviar a través de Kannel, debe utilizar el signo + como separador de palabras, no el espacio. Para ello hemos creado una etiqueta que tomará como atributo un mensaje y creará en un nuevo atributo el nuevo mensaje formateado con + como sepador de palabras. Además se podrá especificar un valor para el atributo `action` que si coincide con el valor del atributo `compare` se realizará la acción, en caso contrario no se realizará.

La utilización de los atributos `action` y `compare` es para evitar el uso de etiquetas <c:if> cuando la creación de un mensaje depende de una condición previa. Si siempre quiere crearse el mensaje basta con dar el mismo valor a estos atributos.

Para la creación de etiquetas personalizadas debemos crear varios componentes y definirlos adecuadamente:

— Clase que implementa cada operación deseada. Para nuestra ampliación creamos la clase `PrepareSmsCustomTag` en el paquete `application.http.view.customtags`. Esta etiqueta extiende a la clase `javax.servlet.jsp.tagext.TagSupport`, etiquetas sin cuerpo, es decir, sólo utiliza atributos para la definición de la acción. Son etiquetas del tipo `<etiqueta atributos />`, y no `<etiqueta [atributos]>cuerpo</etiqueta>`.

— Descriptor de la librería de etiquetas, `tld`. Nuestra librería sólo definirá la etiqueta `prepareSms`. En el descriptor de la librería, `mytags.tld`, se define la etiqueta y cada uno de sus atributos. Copiaremos nuestro descriptor de librería en `WEB-INF/CustomTags`.

— Definir la librería de etiquetas creada en el descriptor de despliegue de la aplicación web.xml.

— Definir la librería de etiquetas en cada página JSP que la utilice.

```
// Ejemplo de etiqueta personalizada
package application.http.view.customtags;
import java.io.IOException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.jsp.tagext.TagSupport;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;

public class PrepareSmsCustomTag extends TagSupport{

    // Definimos atributos utilizados.
    private String action;
    private String compare;
    private String text;
    private String outvar;

    // Se crean métodos set para dar valor a los atributos privados a
    // partir del valor de los atributos en la etiqueta.
    public void setAction(String action){
        this.action = action;
    }

    // Resto de métodos set.

    // Puesto que utilizamos una etiqueta sin cuerpo basta con
    // extender el método doStartTag.
    public int doStartTag() throws JspException {

        try{
            if(compare.equals(action)){
                JspWriter out = pageContext.getOut();
                out.print(text);
                String formattedText = text.replace(" ", "+");
                pageContext.setAttribute(outvar, formattedText);
        }

        // No necesitamos evaluar ningún cuerpo en la etiqueta.
        return SKIP_BODY;

        }catch (Exception e) {
           throw new JspException("Error");}
    }
}
```

Bloque de código 8-5: Ejemplo de clase de implementación de etiqueta personalizada sin cuerpo.

```
<!-- Ejemplo de descriptor de librería de etiquetas personalizadas.-->
<taglib>

    <tlibversion>1.0</tlibversion>
    <jspversion>1.2</jspversion>
    <shortname>my</shortname>
    <uri></uri>
    <info>My tags library</info>

    <tag>
    <name>prepareSms</name>
    <tagclass>application.http.view.customtags.PrepareSmsCustomTag
        </tagclass>
    <bodycontent>empty</bodycontent>
    <info>Prepares text value as sms for Kannel whe action and compare
        value is the same. Sms formatted is stored in outvar value
        parameter.</info>
    <attribute>
        <name>action</name>
        <required>true</required>
    </attribute>
    <attribute>
        <name>compare</name>
        <required>true</required>
        <rtexprvalue>true</rtexprvalue>
    </attribute>
    <attribute>
        <name>text</name>
        <required>true</required>
        <rtexprvalue>true</rtexprvalue>
    </attribute>
    <attribute>
        <name>outvar</name>
        <required>true</required>
    </attribute>
    </tag>

</taglib>
```

Bloque de código 8-6: Ejemplo de descriptor de desplieque de etiqueta personalizada.

```
<!-- web.xml -->
<web-apps>
<!-- ... -->
<!--  Descriptores de librerías de etiquetas personalizadas.  -->

         <taglib>
         <taglib-uri>/mytags.tld</taglib-uri>
         <taglib-location>/WEB-INF/CustomTags/mytags.tld
             </taglib-location>
         </taglib>

<!-- ... -->
</web-apps>
```

Bloque de código 8-7: Definición de etiquetas personalizadas en web.xml.

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jstl/fmt" %>
<%@ taglib prefix="bean" uri="http://struts.apache.org/tags-bean"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>

<%-- Definición de etiquetas personalizadas --%>
<%@taglib  prefix="my" uri="/mytags.tld" %>

<%@page errorPage="/HTML/kannel_error.jsp"%>

<blockquote>
<fmt:message key="success.content"/>

<c:choose> <c:when test="${empty requestScope.smss}">
<div class="row">No hay mensajes para enviar</div></c:when>

<c:otherwise>

<p><strong><fmt:message key="kannel.report.sms"/> : </strong></p>

<c:forEach var="sms" items="${requestScope.smss}">
<p><c:out value="${sms.smsTo}"/> :"<my:prepareSms
action="update.request"
compare="update.request" text="${sms.smsBody}" outvar="out"/>"
<bean:include id="kannel"
href="http://localhost:13013/sendsms?username=kannel&password=kannel&
to=${sms.smsTo}&text=${out}"/>
<bean:write name="kannel"/></p>
```

```
</c:forEach>

</c:otherwise></c:choose>

<p><b><fmt:message key="kannel.report.serverstatus"/></b> : </p>
<p><bean:include id="status"
href="http://localhost:13000/status.txt"/>
<bean:write name="status"/></p>
<br><br>
</blockquote>
```

Bloque de código 8-8: Ejemplo de etiquetas personalizadas en la amplicación de la aplicación ejemplo.

8.4.5 Configuración e integración de Kannel

Por último, para ofrecer los servicios SMS, es necesario integrarla la pasarela de SMS con nuestra aplicación.

Creamos el fichero de configuración de Kannel que llamaremos `appssms.config`. En este fichero tenemos que configurar el grupo núcleo (`grupo = core`). Este grupo nos permitirá especificar el puerto para la administración de la pasarela. En este puerto podremos consultar el estado de la pasarela y modificar este estado.

En la aplicación, en la página `kannel.jsp`, utilizamos este puerto para consultar el estado del servidor lanzando la petición `http://localhost:13000/status.txt`, donde `13000` es el puerto definido para las tareas de administración.

Para utilizar la pasarela como pasarela de SMS, configuramos también la smsbox, definiendo el puerto desde el que podremos enviar mensajes desde nuestra aplicación. La hemos configurado para utilizar el servidor local, en el puerto 13001, en el contexto `/sendsms`, de este modo, para enviar mensajes desde nuestra aplicación, tendremos que lanzar la petición `http://localhost:13013/sendsms?username=kannel&password=kannel&to=${smsTo}&text=${smsBody}`. Donde `smsTo` y `smsBody` son las variables que contienen el

destinatario del mensaje y el cuerpo del mensaje. También para el envío de mensajes es necesario configurar al menos un grupo de usuarios (`group = sendsms-user`).

Los grupos de SMSC y modems se configurarán para utilizar el módem interno del Alcatel OT535, que es la SMSC Virtual que utilizaremos.

Finalmente, configuraremos los servicios SMS que permitiremos. Cada servicio de define por la primera palabra del SMS que solicita un servicio, y la petición HTTP y argumentos que especificamos para cada una de ellas.

A continuación se incluye el fichero de configuración que cargaremos en la pasarela para poder utilizarla con nuestra aplicación.

```
group = core
admin-port = 13000
smsbox-port = 13001
admin-password = bar

# SMSBOX SETUP
group = smsbox
bearerbox-host = localhost
sendsms-port = 13013
sendsms-url = /sendsms
global-sender = MyApplication

# SMSC GSM
group = smsc
smsc = at
modemtype = alcatel
device = /dev/ttyS4
pin = 4212

group = modems
id = alcatel
name = "Alcatel"
detect-string = "Alcatel"
init-string = "AT+CNMI=2,3,0,0"

#SEND-SMS
group = sendsms-user
```

```
username = kannel
password = kannel
omit-empty = "true"
footer = "::MyApplication::"

#SERVICES
group = sms-service
keyword = default
text = ""
omit-empty = "true"

group = sms-service
keyword = setphone
get-url = "http://localhost:8084/AppsSms/Sms/ChangePhone.do?
loginName=%s&password=%s&phone=%s"
omit-empty = "true"

group = sms-service
keyword = hrequest
get-url =
"http://localhost:8084/AppsSms/Sms/RegisterHolidaysRequest.do?
loginName=%s&password=%s&startDate=%s&finalDate=%s&details=%r"
omit-empty = "true"

group = sms-service
keyword = hrcancel
get-url = "http://localhost:8084/AppsSms/Sms/CancelHolidaysRequest.do?
loginName=%s&password=%s&autoId=%s"
omit-empty = "true"
```

Bloque de código 8-9: Configuración e integración de Kannel con nuestra aplicación.

8.5 AMPLIACIÓN: J2ME

En este apartado se presenta una posible ampliación (que no implementaremos) con el uso de la plataforma J2ME. Esta ampliación permitiría descargar una pequeña aplicación para los dispositivos móviles, que permita acceder a los servicios SMS proporcionados por la aplicación inicial, desde un menú que controle de forma automática el envío de mensajes desde el dispositivo móvil.

Esta aplicación proporcionaría un acceso más fácil y cómodo al usuario, ya que bastaría con la selección de las opciones correspondientes en un menú evitando la necesidad de escribir el mensaje SMS directamente.

El uso de esta aplicación quedaría restringida a dispositivos móviles capaces de ejecutar Java.

La interfaz WMA (Wireless Messaging API) de J2ME, proporciona capacidades de SMS para clientes Java móviles. Esta interfaz permite a los terminales comunicarse con la red vía SMS o CBS (mensajes de difusión, Cell Broadcast Short Message Service), permitiendoles ejecutar aplicaciones de servidor basadas en SMS.

El paquete `javax.wireless.messaging` incluye la interfaz WMA, y para crear la clase que se descargará al dispositivo móvil tentremos que extender la clase `javax.microedition.midlet.MIDlet`.

Actualmente la mayoría de vendedores de telefonía móvil como Motorola, Siemens o Nokia soportan estas interfaces.