Universidad Autónoma Metropolitana

Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería

Generador de soluciones al problema de Reservación y Cotización de Viajes utilizando la Composición Automatizada de Servicios Web

Reporte final de proyecto terminal

Alumno
Daniel Armando Malagón Mercado
207303605

Asesora
Dra. Maricela Claudia Bravo Conteras
Departamento de sistemas

Abril 2013

Índice

1. Introducción

Los servicios web son un tipo de middleware mediante el cual se comunican aplicaciones remotas. En esencia, funciona como cualquier otro tipo de middleware, pero con la diferencia importante de que los mensajes que se envían y se reciben se adhieren a un protocolo estandarizado llamado SOAP (Simple Object Access Protocol). Tanto la llamada al servicio remoto como la respuesta se codifican en SOAP y se transportan, normalmente, mediante http.

Se ha propuesto SOAP, un protocolo de mensajería basado en XML: así, la llamada a una operación ofrecida por un servidor consiste realmente en la transmisión de un mensaje SOAP, el resultado devuelto es otro mensaje SOAP[1], etc. De este modo, el cliente puede estar construido en Java y el servidor en .NET, pero ambos conseguirán comunicarse gracias a la estructura de los mensajes que intercambian. Por otro lado, los proveedores ofrecen, a sus posibles clientes, una lista con los servicios web que ofrecen, describiéndolos también con un lenguaje estandarizado llamado WSDL (Web Services Description Language), que es una representación en XML[2] de los servicios ofrecidos. Así, un cliente puede conocer los métodos ofrecidos por el servidor, sus parámetros con sus tipos, etc., simplemente consultando el correspondiente documento WSDL[3].

Durante los últimos años los servicios Web han tenido un importante auge en la solución de diversos problemas, la composición de servicios es una técnica muy útil para la solución de problemas que requieren la integración e interacción de diferentes proveedores de software. Sin embargo, la composición de servicios Web es una tarea difícil de implementar ya que requiere de mucho tiempo por parte del desarrollador e integrador de sistemas.

Existen 2 técnicas principales para realizar la composición de servicios web cuyas diferencias se detalla a continuación:

Orquestación: Esta técnica usa un orquestador para sincronizar los pasos de mensajes, debido a esto cada servicio solamente conoce su proceso y el orquestador le va indicando cuando y como debe de realizar los procesos y a quien enviar los mensajes de resultados, un buen ejemplo de este tipo de composición es BPEL[4].

Coreografía: Esta técnica no utiliza el orquestador y en su lugar los servicios trabajan de manera conjunta para realizar la composición, cada servicio tiene una tarea asignada y un

[1] Simple Object Access Protocol
[2] Lenguaje de marcas extensible (W3C)
[3] Web Services Description Language
[4] Business Process Execution Language

tiempo de ejecución y de este modo sin necesidad del orquestador se realiza una composición sincronizada.

Este proyecto trata de dar una aproximación a este orquestador, puesto que indica que servicios deben de ir ejecutándose para resolver un problema de agencia de viajes, pero no dando solamente una solución sino planteando un conjunto de soluciones de un mismo problema. Lo que se busca con este proyecto es ofrecer una solución que le ayude al desarrollador con esta tarea mediante la generación automatizada de composiciones de servicios Web de manera más sencilla a las que actualmente se utilizan, si bien no lo suficientemente poderosa para utilizarse en cualquier caso o problema que se presente, si se aplicará en la generación de soluciones a problemas como el de la cotización y reservación de viajes. Este proyecto permitirá que una agencia de viajes cuente con una aplicación que le permita comunicarse mediante protocolos interoperables a través de Internet a las diferentes aplicaciones de compañías de hoteles y vuelos así como a los bancos para realizar pagos electrónicos. Como resultado, esta aplicación generará no solo una opción para el viaje, sino varias posibles soluciones para que los usuarios escojan la que más les agrade todo esto de manera automática usando servicios Web.

La tabla siguiente muestra la ventaja del uso de servicios web frente a técnicas de programación más clásicas para el desarrollo de sistemas empresariales.

	Programación Estructurada	**Objetos**	**Componentes**	**Servicios**
Granularidad	Muy fina	Fina	Intermedia	Gruesa
Contrato	Definido	Privado/Publico	Publico	Publicado
Reusabilidad	Baja	Baja	Intermedia	Alta
Acoplamiento	Fuerte	Fuerte	Débil	Muy débil
Dependencias	Tiempo de Compilación	Tiempo de Compilación	Tiempo de Compilación	Run-Time
Ámbito de Comunicación	Intra-Aplicación	Intra-Aplicación	Inter-Aplicaciones	Inter-Empresas

Figura 1 Características de las distintas técnicas de programación

Si bien se observa que los métodos más clásicos tienen ciertas ventajas sobre los servicios, hay algunas características que los vuelven esenciales para los procesos que requieren las empresas y que son muy importantes para sus aplicaciones, de la tabla anterior podemos destacar la granularidad y la reusabilidad, la figura siguiente muestra con más detalle los distintos tipos de granularidad existentes.

Tamaño de grano	Descripción	Intervalo de sincronización (instrucciones)
Fino	Paralelismo inherente en un único flujo de instrucciones.	< 20
Medio	Procesamiento paralelo o multitarea dentro de una aplicación individual	20-200
Grueso	Multiprocesamiento de procesos concurrentes en un entorno multiprogramado.	200-2000
Muy grueso	Proceso distribuido por los nodos de una red para formar un solo entorno de computación.	2000-1M
Independiente	Varios procesos no relacionados.	(N/A)

Figura 2 Tipos de granularidad

Los servicios web tienen una función gruesa de granularidad y es por esto que son tan usados para entornos empresariales, permiten acelerar en muchas situaciones los procesos pero también permiten realizar la interoperabilidad de aplicaciones. Esta interoperabilidad es el requerimiento principal de las empresas ya que necesitan comunicar sus sistemas con sistemas de otras empresas y en el caso de las agencias de viajes necesitan sincronizar procesos de compañías hoteles con compañías de transporte e incluso con instituciones bancarias.

Objetivo general

Diseñar e implementar un sistema que genere distintas opciones de *Composición de Servicios Web* que den solución a un problema de reservación y cotización de viajes a partir de un archivo de entrada con la especificación detallada en un formato de texto predefinido.

Objetivos específicos

Diseñar e implementar un módulo que obtenga del usuario la especificación del problema de planeación de un viaje a través de un archivo de texto.

Diseñar e implementar un módulo que a partir de la especificación recibida busque y seleccione los servicios Web que satisfacen los requerimientos especificados por el usuario.

Diseñar e implementar un módulo que integre los servicios web y muestra los resultados de las diferentes composiciones realizadas.

2. Desarrollo

2.1. Diseño

2.1.1. Poblado de la base de datos

Para su correcto funcionamiento la aplicación requirió de un modulo auxiliar independiente al principal este modulo lo que hace es, a partir de varios descriptores de servicios ubicados en una página aplicarles un parseo sencillo para obtener los datos necesarios para llenar la base de datos, este parseo solo incluye el nombre del servicio, la url de invocación, sus métodos y sus parámetros, este último punto solo llega al primer nivel o el paso de mensajes de entrada y de salida. El poblador como es llamado este modulo se adjunta en el CD siendo una aplicación independiente al proyecto original pero necesaria cuando no se tiene una base de datos de servicios previa. Se eligió esta solución puesto que es bien sabido que en algún momento los repositorios pueden dejar de estar disponibles, sin embargo el servicio puede seguir funcionando en la dirección en donde se público y con el uso de una base de datos auxiliar la empresa que utilice la aplicación tendrá un respaldo de esa información de los servicios sin depender de los repositorios de nuevo. Para más detalles consulte los anexos.

2.1.2. Arquitectura y funcionamiento de la base de datos

Se utilizo el manejador mysql para el montaje de la base de datos y se trato de elaborar un diseño que permitiera la expansión de la aplicación para que no funcionara solamente con servicios de vuelos sino que funcionara con servicios de otras áreas, el diseño fue el primer paso en la solución del problema y la parte que más tiempo llevo realizar puesto que debía ser expandible y además se busco que las consultas fueran sencillas, el diseño original tenía 3 tablas y no contemplaba la expansión a otras aéreas, además de que estaba diseñado sobre el manejador postgres[5] que a diferencia de mysql y oracle[6] es un poco distinto en la estructura de sus consultas y en la manera de insertar datos a las tablas. El diseño final quedo como se muestra en la siguiente figura:

[5] Postgre SQL es un manejador de base de datos relacionales
[6] Oracle: sistema de gestión de base de datos objeto-relacional

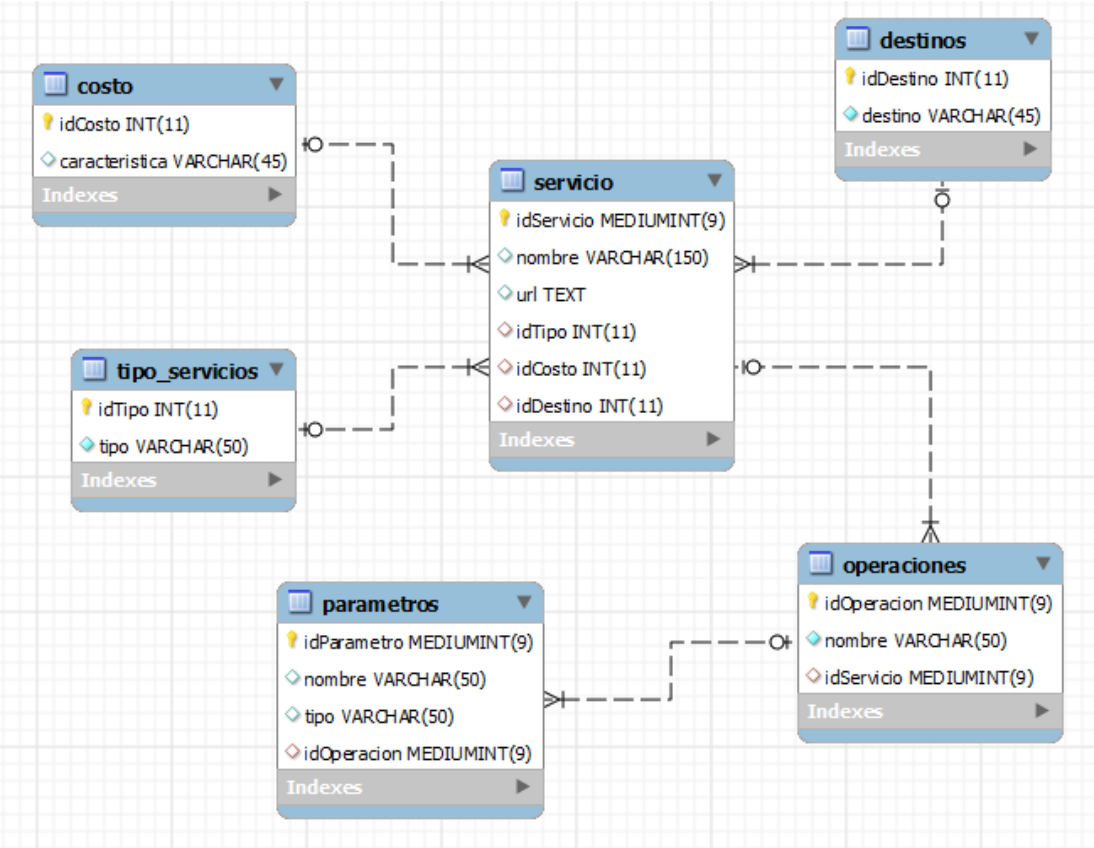


Figura 3 Modelo de entidad relación

Las tablas de costo, tipo_servicios y destinos son expandibles con solo agregar un registro nuevo a la tabla, la base de datos actual solamente tiene servicios de vuelos y de hoteles y los destinos están limitados pero también son fácilmente expandibles agregando nuevos registros, el modulo poblador fue el encargado de llenar la base de datos, sin embargo este método tiene el inconveniente en que los destinos y costos no aparecen en los descriptores por lo que estas columna debe modificarse a mano.

Las pruebas se realizaron con una cantidad moderada de servicios; 75 para ser exactos; y el llenado fue de manera secuencial y con números aleatorios para obtener la columna de destinos y de costos.

Una vez que la base de datos estuvo terminada se utilizo un marco de trabajo (*framework)* que ayudo a pasar los datos de la base de datos a condigo java. Este *framework* fue mybatis.

2.1.3. Software utilizado

Spring tool suite: es un entorno de desarrollo integrado que está enfocado al *framework* spring, cuenta con herramientas únicas para facilitar la implementación de spring, al estar basado en eclipse es compatible en su totalidad con agregados para este ultimo y además permite importar proyectos realizados con otras aplicaciones.

Mysql[1]: manejador gratuito de base de datos, se escogió este por su amplio uso a nivel de aplicaciones y proyectos universitarios así como por su rendimiento y facilidad de uso. Además la compatibilidad con cualquier plataforma lo hacía indicado para el montaje en el servidor Centos.

Apache Tomcat: es una implementación de los servlets de java y del framework de java server faces para mostrar páginas, este software es muy importante en todo lo relacionado a aplicaciones java edición empresarial y su capacidad para funcionar en distintas plataformas lo convierte en una herramienta básica del desarrollo web sobre Java.

Java: lenguaje de programación orientado a objetos que funciona en cualquier sistema operativo gracias a su máquina virtual que sirve para ejecutar el código. Es esencial para la aplicación realizada en este proyecto.

Mybatis[2]: es un framework que trabaja sobre la capa de los datos facilitando el manejo de la información y eliminado casi completamente el código jdbc requerido para hacer conexiones a base de datos.

Bootstrap[3]: software que permite realizar vistas de manera fácil y rápida mediante el uso de plantillas prediseñadas, se utilizo para la vista de la paginas jsp.

2.2. Implementación

2.2.1. Resumen y utilización del framework de MyBatis

Este framework tiene bastante similitud con otro framework para manejar base de datos con más años funcionando: Hibernate. El principio también se basa en archivos de configuración xml que contienen los datos del framework pero además contiene los datos de la conexión, estos datos son los mismos que cuando se realiza la conexión mediante el lenguaje de programación: driver, host, usuario, password y el esquema que se usara. Este archivo lleva por nombre database.properties y se ubica dentro de la carpeta oculta de la

aplicación web, de esta manera los visitantes no pueden ver los datos de la conexión y se agrega un plus de seguridad del que carecen las conexiones realizadas en el código fuente.

Además de esta ventaja mybatis tiene la característica de utilizar una fábrica de sesiones para las consultas sql[7], la función de esta fabrica es crear la instancia de conexión a partir del archivo de configuración, esta instancia o lo que es lo mismo un objeto que se instancia de la fábrica de sesiones mantiene la conexión abierta durante el uso de la aplicación hasta que ya no sea requerido. Si en algún momento se requiere de nuevo la conexión simplemente se inicia otra sesión de la fábrica.

Después de que se crea la sesión de conexión se utilizan las instancias de mapeos, estos archivos también en xml son los que le dicen a mybatis de donde obtener la información y en que objetos se va a guardar para su posterior uso, el query que retorna la información se debe ubicar esta parte, también es necesario colocar aquí en que objetos se va a guardar la información, esto depende de la lógica de negocio que se requiera.

La siguiente figura muestra la arquitectura del *framework* y en ella se ve claramente la función principal que realiza:

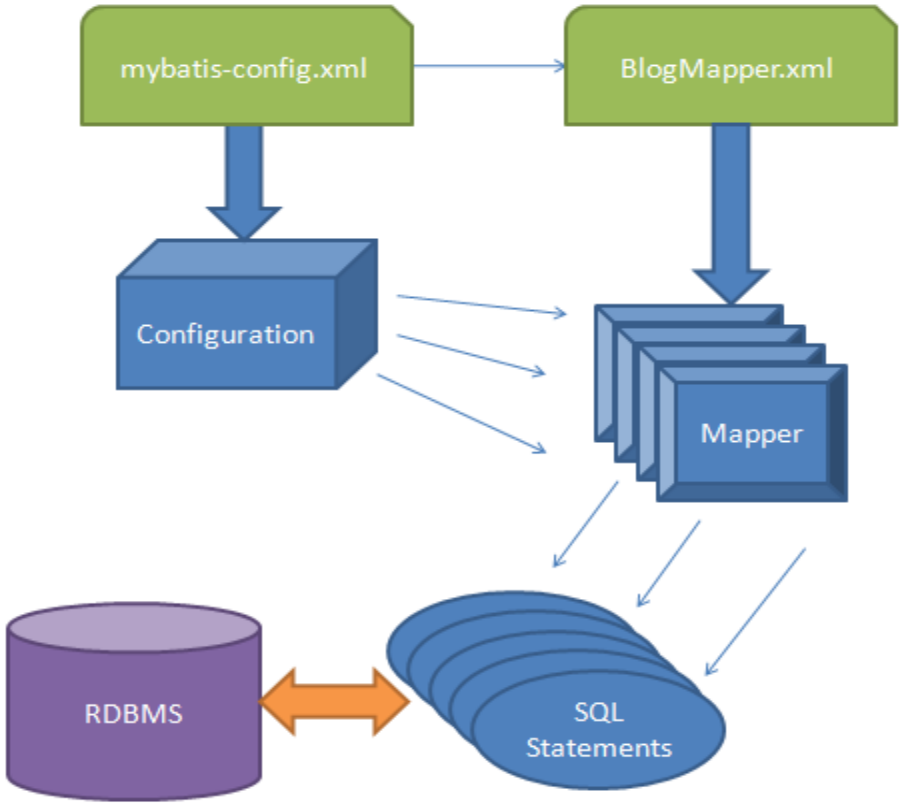


Figura 4. Arquitectura del framework mybatis

[7] Structured Query Language, lenguaje estructurado de consultas

2.2.2. Estructura de las consultas utilizadas

En la aplicación se hizo uso de querys que utilizan operadores de los manejadores de base de datos, el operador utilizado fue join. Los joins son una manera de agrupar tablas para obtener información de varias tablas a la vez con pocas condiciones en la sección WHERE de una consulta.

EL uso de joins tiene ventajas significativas sobre todo a la hora de hacer consultas con bases de datos muy grandes, permiten reducir los tiempos y si son correctamente usados hacer querys más sencillos en caso de que se tenga que filtrar mucha información. También permiten realizar operaciones en el mismo query por ejemplo productos cruzados de varias tablas. En el caso del proyecto se utilizaron siempre por la izquierda o sencillos con el único fin de agrupar tablas para dejar la clausula WHERE con solamente 1 o 2 condiciones según se necesitara.

Las consultas que utiliza la aplicación son los siguientes:

```
SELECT
        cost.caracteristica
FROM
        servicios.costo as cost
```

```
SELECT
        dest.destino
FROM
        servicios.destinos as dest
```

```
SELECT
        serv.idservicio as serv_id,
        serv.nombre as serv_nombre,
        serv.url as serv_url,
        op.idoperacion as op_id,
        op.nombre as op_nombre,
        param.idparametro as pa_id,
        param.nombre as pa_nombre,
        param.tipo as pa_tipo
FROM
        servicios.servicio as serv
```

```
        left JOIN servicios.operaciones as op on op.idservicio=serv.idservicio
        left JOIN servicios.parametros as param on param.idoperacion=op.idoperacion
        JOIN servicios.costo as costo on costo.idcosto=serv.idcosto
        JOIN servicios.destinos as dest on dest.iddestino=serv.iddestino
        JOIN servicios.tipo_servicios as tipo on tipo.idtipo=serv.idtipo
WHERE
        costo.caracteristica='precio'
        AND dest.destino='destino'
        AND     tipo.idtipo='tipo_servicio'
```

Las 2 primeros consultas son utilizadas para llenar los botones del filtrado; el primero recupera los costos y el segundo recupera los destinos, esta consulta se realiza 2 veces para seleccionar el costo del hotel y el costo del avión de manera separada, el segunda consulta se manda a llamar una sola vez y abarca ambos tipos de servicios.

El tercer query es el que recupera la información de los servicios de las distintas tablas que componen la base de datos. Es en este donde se aprecia la utilización de los joins, traducido a lenguaje común el query hace lo siguiente:

Recupera el id, nombre, url, nombre de la operación, parámetros de la operación, nombre del parámetro y tipo del parámetro de la tabla servicio que pertenece al esquema servicios.

Y a continuación vienen las sentencias join: une por la izquierda la tabla operaciones del esquema servicios con la tabla servicio donde el id de la operación de la tabla operación sea igual al id operación de la tabla servicio; une por la izquierda la tabla parámetros donde el id del parámetro de la tabla parámetros sea igual id parámetro de la tabla operaciones, y después se unen las tablas destinos, tipo_servicio y costo cuyos id sean iguales a los id de la tabla servicio.

Las tablas destinos, tipo_servicio y costo son auxiliares y sirven solamente para dar información para que la consulta pueda filtrar la información, este filtrado aparece en la parte de la clausula WHERE y es por esta misma razón que se hace un join sencillo para usarlos, puesto que no traen información. Las otras tablas de parámetros y operaciones si tienen información relevante que está relacionada a la tabla servicios, la razón por la que se tiene que hacer un join por la izquierda es justamente para que lea la tabla completa a la hora de juntarlos el left join recupera información solamente si hay coincidencias en la condición por lo que recupera las operaciones y los parámetros que tengan coincidencia con los servicios resultantes del filtrado.

Si se hubiera utilizado solo la clausula WHERE se tendrían que comparar las tablas operaciones y parámetros en busca de coincidencias y hubiera sido necesario llenar una columna extra en ambas tablas, el query utilizado por la aplicación solamente realiza

comparaciones en la tabla servicios y gracias a los joins recupera la información completa del servicio seleccionado con solo 3 comparaciones.

2.2.3. Programación del proyecto

La implementación se realizo usando spring, este framework permite realizar la aplicación diferenciando las 3 capas de la aplicación, la capa de datos, el modelo de negocio y la vista. La figura siguiente muestra la estructura del framework y en que parte de la estructura aparece mybatis es importante hacer mención que gracias a este *framework* la aplicación tiene bien marcadas las 3 capas de la aplicación y cada capa se puso en distintos paquetes dentro del código fuente, esto permite un manejo bastante sencillo de los distintos recursos y permite trabajar en el apartado grafico sin interferir con el trabajo de la parte de datos.

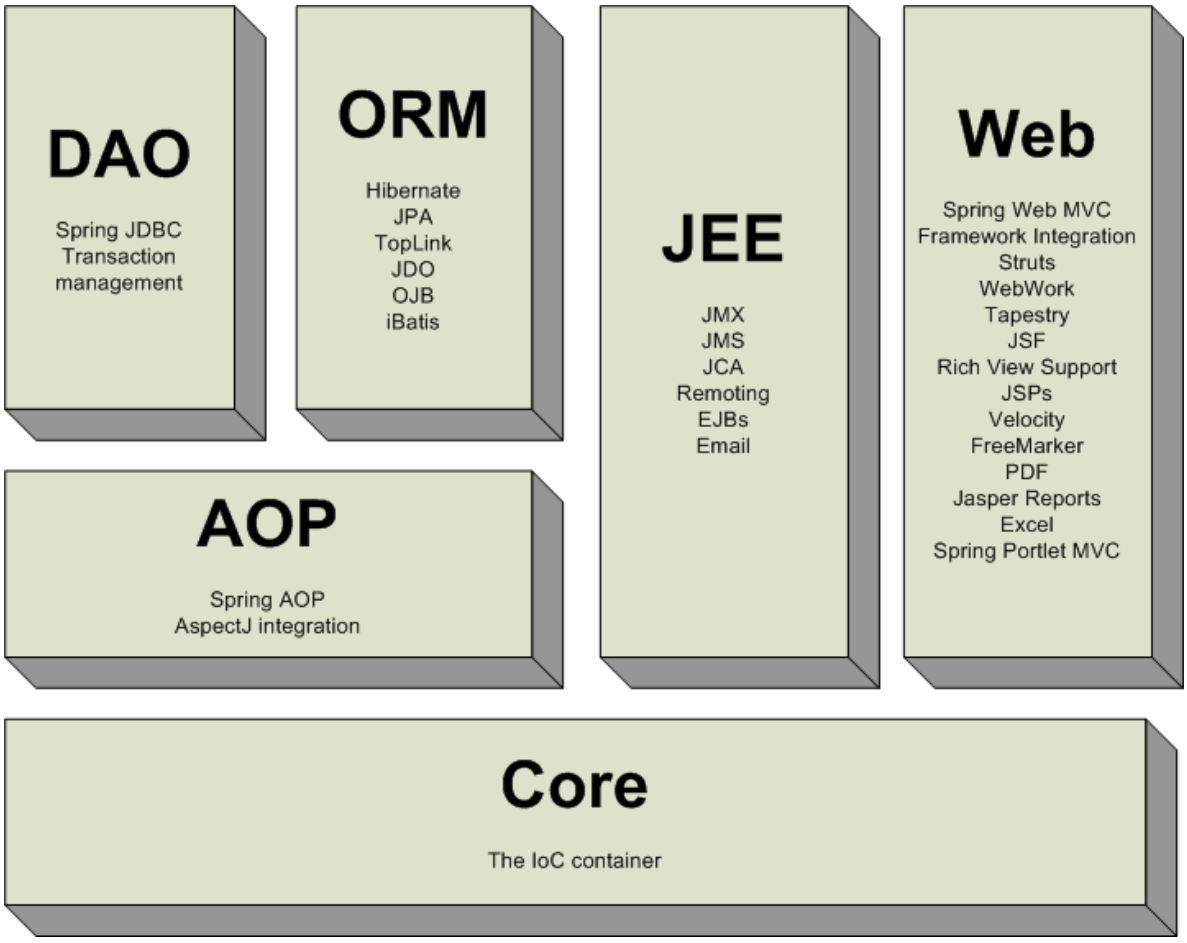


Figura 4 Arquitectura de Spring MVC

De esta imagen el proyecto utiliza únicamente el modulo DAO que son las interfaces dentro del código fuente, el ORM que son el equivalente a los xml del mybatis el Core y la parte

Web que corresponde a los jsp. Los *controller*[8] pasan las variables al jsp y con esto se pintan los formularios y las paginas que muestra la aplicación.

Una vez conocido el funcionamiento del *framework* y sabiendo que hacen los querys se puede ver como se realizo la implementación dentro el código. Primero se ubicaron los archivos de configuración del framework y el archivo de la conexión y de la fábrica de sesiones.

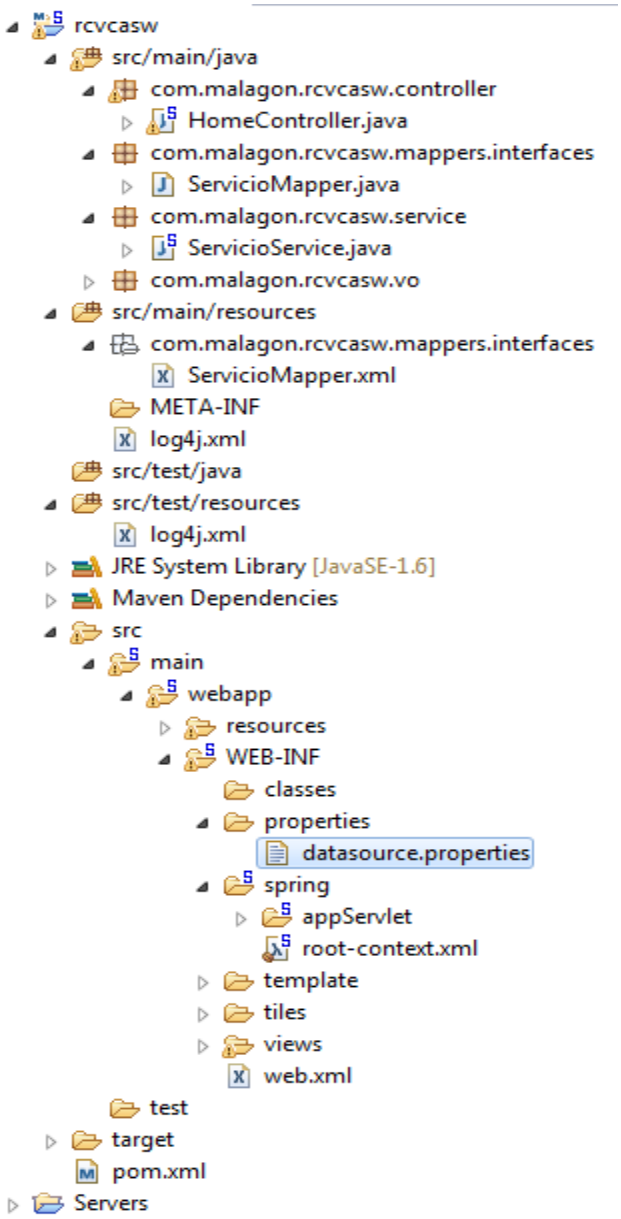


Figura 5 Estructura del código fuente

[8] Controlador(controller) son estructuras que responden a eventos que genera el usuario

Los archivos resaltados son parte del framework mybatis, dentro del directorio src/main/resources se ubica el mapeador, es ahí donde están los querys y las funciones a las que se mandan llamar para mapear los resultados que se obtengan, el otro archivo datasource.properties es el archivo en donde se encuentran los datos de la conexión. Ambos archivos así como su contenido son mostrados a continuación:

Archivo datasource.properties:

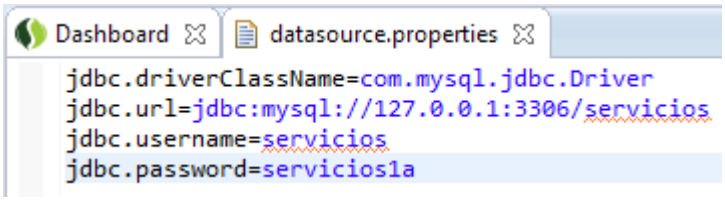


Figura 6 Datos de la conexión a base de datos

Como se puede apreciar simplemente incluye el driver, la cadena de conexión con el esquema (base de datos) a usar, el usuario y la contraseña, este archivo es modificable para cualquier tipo de conexión y cualquier manejador.

Archive ServicioMapper.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper
  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.malagon.rcvcasw.mappers.interfaces.ServicioMapper">
        <select id="permutacion" parameterType="com.malagon.rcvcasw.vo.Filtro"
resultType="com.malagon.rcvcasw.vo.IdsServicios">
                SELECT idhoteles.idServHotel, idvuelos.idServVuelo FROM
                (
                        (
                        SELECT serv.idservicio as idServHotel
                        FROM servicios.servicio as serv
                        JOIN servicios.costo as costo on costo.idcosto=serv.idcosto
                        JOIN servicios.destinos as dest on
dest.iddestino=serv.iddestino
                        JOIN servicios.tipo_servicios as tipo on
tipo.idtipo=serv.idtipo
```

```
                    WHERE
                        costo.caracteristica=#{hotelPrecio}
                        AND dest.destino=#{hotelDestino}
                        AND   tipo.tipo='hoteles'
                    ) as idhoteles,
                    (
                    SELECT      serv.idservicio as idServVuelo
                    FROM servicios.servicio as serv
                        JOIN      servicios.costo      as      costo      on
costo.idcosto=serv.idcosto
                        JOIN      servicios.destinos      as      dest      on
dest.iddestino=serv.iddestino
                        JOIN     servicios.tipo_servicios     as     tipo     on
tipo.idtipo=serv.idtipo
                    WHERE
                        costo.caracteristica=#{vueloPrecio}
                        AND dest.destino=#{hotelDestino}
                        AND   tipo.tipo='vuelos'
                    ) as idvuelos
            )
    </select>

    <select id="obtenCostos" resultType="String">
        SELECT
            cost.caracteristica
        FROM
            servicios.costo as cost
    </select>

    <select id="obtenDestinos" resultType="String">
        SELECT
            dest.destino
        FROM
            servicios.destinos as dest
    </select>

    <select id="filtrarServiciosWeb" parameterType="com.malagon.rcvcasw.vo.Filtro"
            resultMap="servicioResult">
        SELECT
            serv.idservicio as serv_id,
            serv.nombre as serv_nombre,
            serv.url as serv_url,
            op.idoperacion as op_id,
            op.nombre as op_nombre,
            param.idparametro as pa_id,
            param.nombre as pa_nombre,
            param.tipo as pa_tipo
```

```
            FROM
                servicios.servicio as serv
            left JOIN servicios.operaciones as op on op.idservicio=serv.idservicio
            left JOIN servicios.parametros as param on
param.idoperacion=op.idoperacion
            JOIN servicios.costo as costo on costo.idcosto=serv.idcosto
            JOIN servicios.destinos as dest on dest.iddestino=serv.iddestino
            JOIN servicios.tipo_servicios as tipo on tipo.idtipo=serv.idtipo
            WHERE
                <if test="tipoServicio == 1">
                    costo.caracteristica=#{vueloPrecio}
                    AND dest.destino=#{hotelDestino}
                    AND   tipo.idtipo=1
                </if>
                <if test="tipoServicio == 2">
                    costo.caracteristica=#{hotelPrecio}

                    AND dest.destino=#{hotelDestino}
                    AND   tipo.idtipo=2
                </if>


    </select>

    <resultMap type="com.malagon.rcvcasw.vo.Servicio" id="servicioResult">
        <id property="idServicio" column="serv_id"/>
        <result property="nombre" column="serv_nombre"/>
        <result property="url" column="serv_url"/>
        <collection property="operaciones"
ofType="com.malagon.rcvcasw.vo.Operacion"
            resultMap="operacionResult"/>
    </resultMap>

    <resultMap type="com.malagon.rcvcasw.vo.Operacion" id="operacionResult">
        <id property="idOperacion" column="op_id"/>
        <result property="nombre" column="op_nombre"/>
        <collection property="parametros"
ofType="com.malagon.rcvcasw.vo.Parametro"
            resultMap="parametroResult"/>
    </resultMap>

    <resultMap type="com.malagon.rcvcasw.vo.Parametro" id="parametroResult">
        <id property="idParametro" column="pa_id"/>
        <result property="nombre" column="pa_nombre"/>
        <result property="tipo" column="pa_tipo"/>
    </resultMap>
</mapper>
```

Como se puede apreciar, este archivo tiene los querys y en la parte final se establece que hacer con los resultados que se obtienen de las consultas, al inicio del archivo aparece un método permutación que tiene un query que pareciera estar repetido, en realidad lo que hace es un producto cruz de 2 resultados que trae cada uno de los querys, el primer query con el alias de idhoteles y el segundo query con el alias de idvuelos, estos 2 querys realizan el mismo filtrado que el que obtiene los datos del servicio pero además hace un producto de los 2 resultados obteniendo de ese modo todas las combinaciones posibles de ambos resultados, o las combinaciones de los distintos servicios.

Más abajo se repite un query de manera muy similar sin embargo en su select obtiene todos los demás datos: nombre, parámetros, url, operaciones y tipo. También se encuentran 2 querys más pequeños que son los que llenan los botones para seleccionar el filtrado, estos son dinámicos y en cuanto se agregan más tipos de servicios, más destinos o más características los devuelve y aparecen en la ventana principal de la aplicación.

La parte final del archivo le dice a los controladores superiores de la capa de vista que hacer con los datos obtenidos, el query de los servicios que recupera la información mapea a una interfaz de servicios:

```
<resultMap type="com.malagon.rcvcasw.vo.Servicio" id="servicioResult">
                <id property="idServicio" column="serv_id"/>
                <result property="nombre" column="serv_nombre"/>
                <result property="url" column="serv_url"/>
                <collection                                                property="operaciones"
ofType="com.malagon.rcvcasw.vo.Operacion"
                        resultMap="operacionResult"/>
        </resultMap>
```

Este resultmap tiene las propiedades del objeto servicio definido en una clase java, las siguientes figuras muestran los atributos de las clases servicio, parámetros y operaciones. Estas clases son las que son mapeadas por mybatis y en cada uno de los atributos de las clases se va colocando una columna del query

Figura 7 Paquete con las clases del servicio

Figura 8 Estructura de la clase servicio

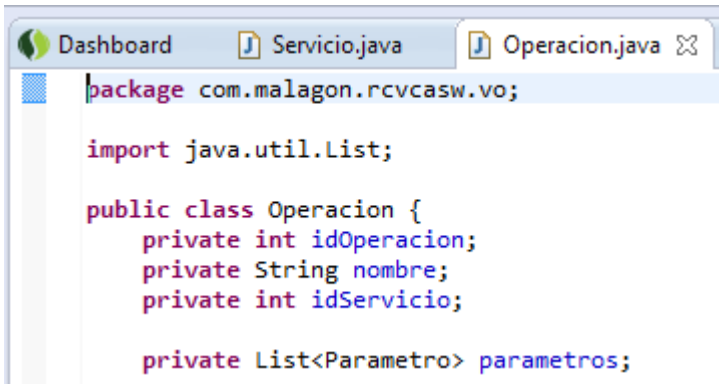


Figura 9 Estructura de la clase operación

Y la última clase mostrada es la que corresponde a los parámetros

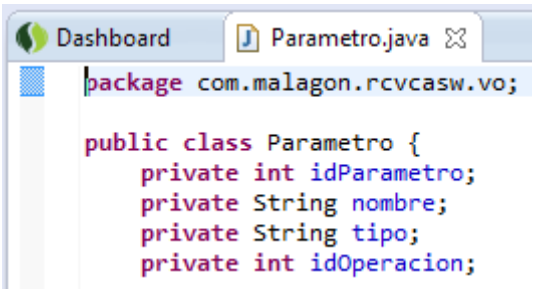


Figura 10 Estructura de la clase parámetro

Ya conociendo las clases y los querys, se puede ver como el resultmap lo que hace es recuperar una columna del query y colocarla en la propiedad de cada una de las distintas clases. Una vez que se tienen en objetos simplemente se manipulan en un archivo jsp utilizando el framework de spring y se muestran en pantalla.

El funcionamiento completo del programa es como se ve en la siguiente imagen.

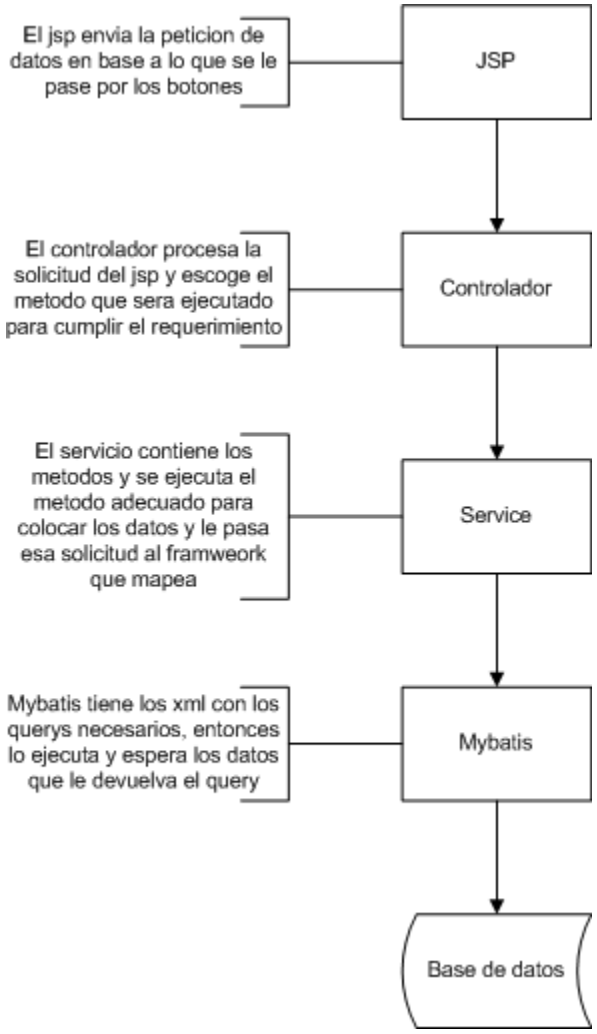


Figura 11 Secuencia de funcionamiento

Cuando mybatis tiene los datos simplemente los va regresando a los atributos del servicio y regresan de ese modo al jsp que únicamente los refleja en pantalla.

2.3. Ejecución del programa

Entramos a la dirección de la instalación de tomcat y abrimos el proyecto agregando al final de la url el nombre del archivo .war sin la extensión. Se abrirá la ventana principal del programa.

Esta ventana muestra las palabras claves que servirán para filtrar los servicios web, para este caso se manejaran servicios de vuelos y de hoteles y sus filtros serán los costos y los destinos, la pestaña de destinos no aparece en los vuelos ya que se asume que se viajara al mismo lugar, una vez seleccionadas las características basta con presionar el botón enviar.

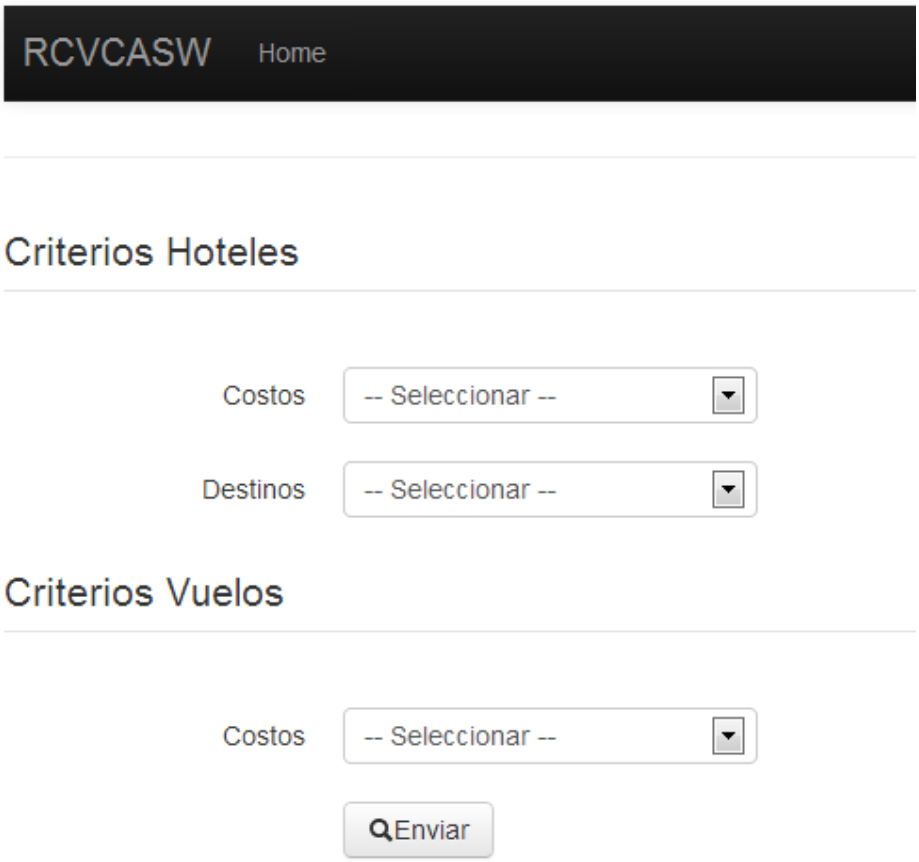


Figura 12 Ventana principal de la aplicación

Después de seleccionar los filtros el programa mostrara los resultados de hacer el filtrado y mostrara en pantalla los servicios que cumplen con las condiciones seleccionadas, el programa muestra 2 secciones claramente identificadas, en la primera, muestra toda la información referente a los servicios, el id (el cual sirve como identificación para la base de

datos) el nombre, del servicio, la url de invocación del servicio y las operaciones que tiene el servicio.

Figura 13 Ejemplo del filtrado del servicio web

Al darle clic en el nombre de las operaciones el sistema muestra los parámetros de entradas y de salida de ese método en específico.

Figura 14 Vista de las operaciones y de sus parámetros

La parte de abajo de la pantalla muestra la composición, muestra el id del servicio y con cual otro servicio puede juntarse para resolver el problema.

Combinacion de servicios

idVuelo	idHotel
27	18
31	18
39	18
27	22
31	22
39	22
27	44
31	44
39	44

Figura 15 Salida con las combinaciones posibles de los servicios web

3. Conclusiones

Este proyecto trata sobre un tema sumamente importante en lo relacionado a la comunicación empresarial actual, se trató de buscar una solución sencilla a la problemática de la composición de los servicios web, sin embargo el proyecto no toma el tema de la invocación de los servicios que sería el siguiente paso a realizar, la invocación de servicios es un tema que tiene una complejidad mayor a lo que es la planeación de la composición que es el resultado obtenido de este proyecto.

El problema tratado aquí puede ser resuelto utilizando lenguaje de programación puro como java o algún equivalente, sin embargo se optó por buscar una solución que viniera desde una capa más baja y que mantuviera la interoperabilidad de las aplicaciones, este proyecto al ser realizado enteramente a base de consultas y manejo de base de datos mantiene en esencia la característica de los servicios web al ser operables sin importar el lenguaje de programación, es bien sabido que hay distintos manejadores e base de datos sin embargo las funciones utilizadas son comunes en cada uno de los manejadores (especialmente joins) así como la estructura de la base de datos es igual en cada manejador por lo que no es necesario hacer cambios estructurales importantes para hacer funcionar el proyecto.

Durante el desarrollo el principal problema fue llevar archivos WSDL a base de datos, para eso se tuvo que hacer un diseño y un análisis exhaustivo de los requerimientos porque si no podría haber quedado muy cargada la base de datos, fue necesario primero diseñador un programa que poblara la base de datos ajeno a la aplicación principal, no se incluyó dentro del programa principal porque el objetivo del proyecto no era llenar una base de datos, sino dejar una salida que mencionara los pasos de la composición.

El segundo problema fue que se tuvo que hacer un cambio de diseño y de programación de la aplicación, originalmente estaba pensada para funcionar sobre java swing como aplicación de escritorio, sin embargo al final se opto por hacerla funcionar sobre apache tomcat para que fuera aplicación web con la ventaja de que es accesible a través del internet.

4. Anexos

4.1. Diagrama de clases del programa principal

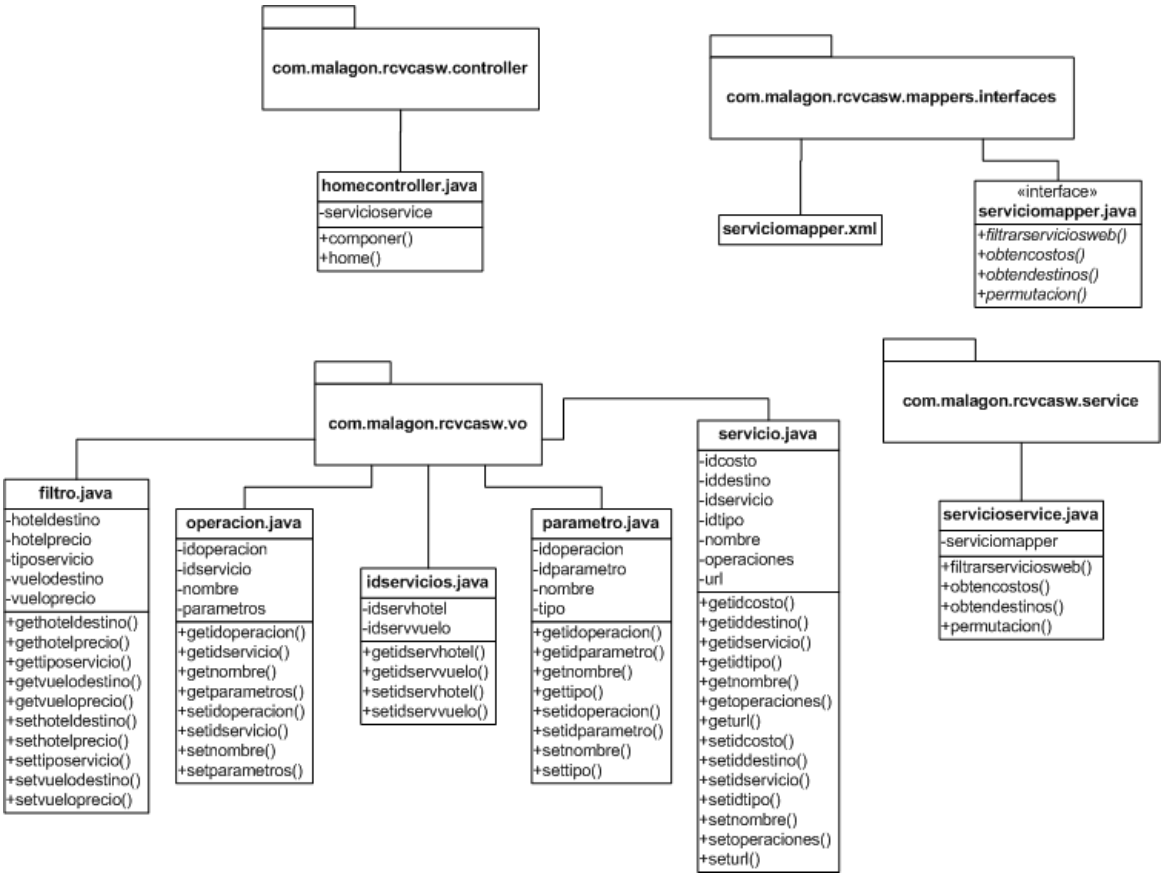


Figura 16 Diagrama de clases del programa

4.2. Modulo poblador

Este modulo es un programa auxiliar que realiza el poblado de la base de datos, se encarga de meter wsdls de algún repositorio en la base de datos del proyecto, es necesario si no hay una base de datos existente. Como trabaja directamente con los wsdl necesita un api extra.

JDOM[7]: Es un API implementado en java para trabajar con archivos xml permite crear, leer y manipular este tipo de documentos. Proporciona una capa de abstracción entre un parser de XML y la aplicación. JDOM está enfocado para trabajar en java y como los archivos WSDL están basados en xml la manipulación de estos se facilita al utilizar esta API, al leer algún documento JDOM nos proporciona clases y métodos para manipularlo, a partir de la estructura del archivo se crea un árbol basado en nodos que nos permite recorrer los nodos y recuperar la información que se necesite. JDOM ayudo en la recuperación de la información de los archivos de descripción permitiendo una navegación a través del los nodos de una forma más fácil a comparación de otras apis similares.

La siguiente figura muestra la estructura del código fuente del programa poblador:

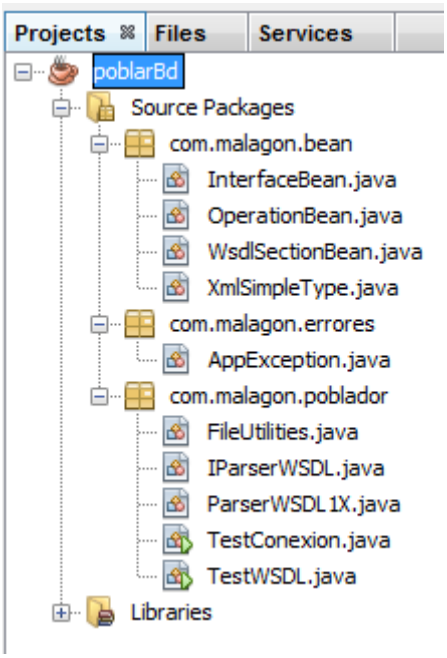


Figura 17 Estructura del código del poblador

Se dividió el código en 3 paquetes que tienen diferenciados claramente sus funciones, el paquete bean tiene las interfaces, el paquete error simplemente es una clase que recupera las excepciones y manda distintos mensajes dependiendo el error que se haya obtenido, el

poblador implementa las interfaces y es el que guarda los datos en la base de datos, se adjunta una imagen del diagrama de clases del poblador.

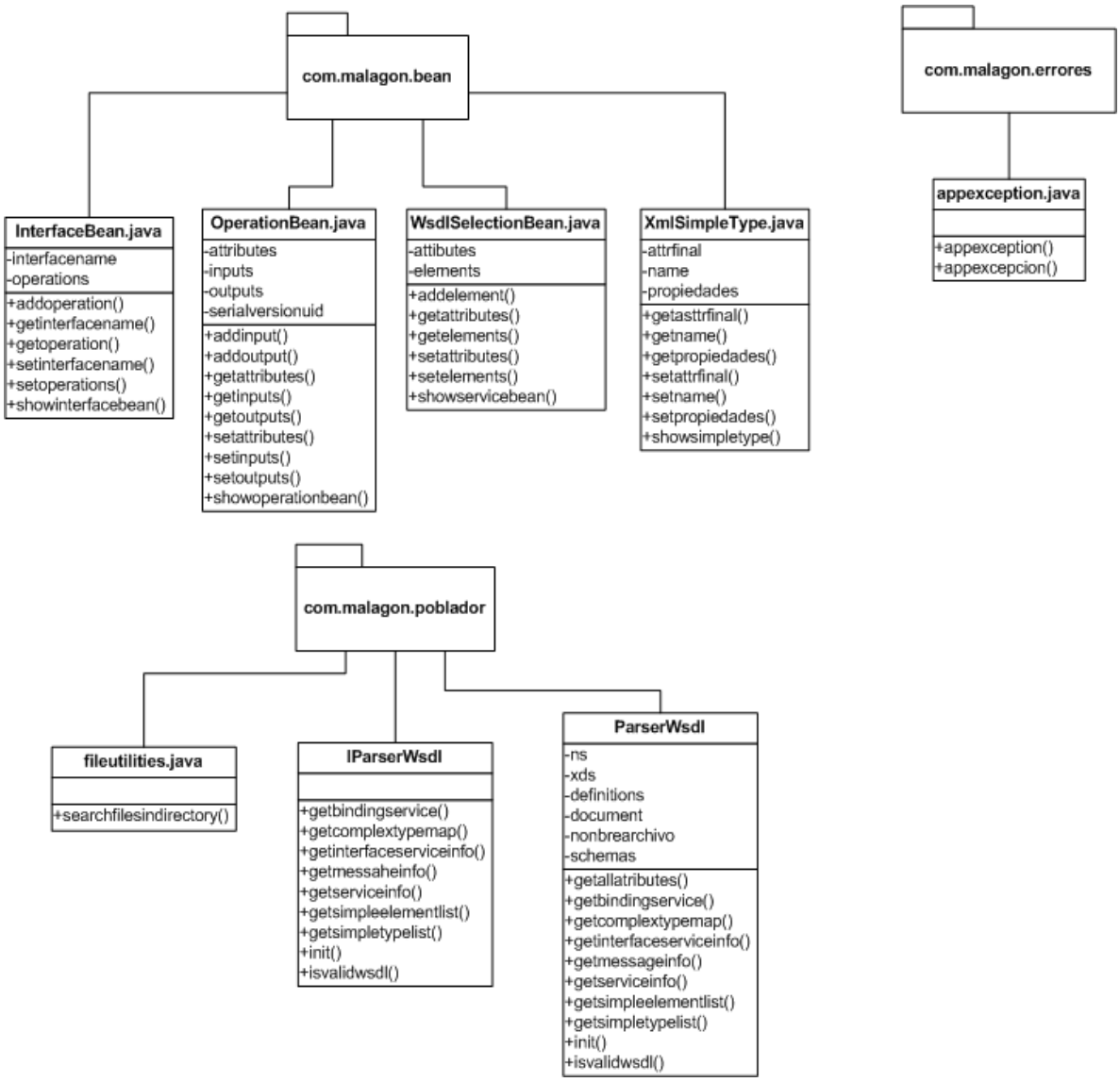


Figura 18 Diagrama de clases del poblador

5. Referencias

[1] MYSQL. mysql [En línea]. “Website”, Abril 2013
http://www.mysql.com

[2] MYBATIS. Mybatis [En línea]. “Website”, Abril 2013
http://mybatis.github.io/mybatis-3/es/index.html

[3] BOOTSTRAP. Bootstrap. [En Línea]. “Website”, Abril 2013
http://twitter.github.io/bootstrap/index.html

[4] Oracle sql tunning guide. [En línea]. “Website”, Abril 2013
http://www.orafaq.com/tuningguide/

[5] Mysql 5.6 Reference Manual. [En línea]. “Website”, Abril 2013
http://dev.mysql.com/doc/refman/5.6/en/index.html

[6] Spring Framework Reference Documentation. [En línea]. “Website”, Febrero 2013
http://static.springsource.org/spring/docs/3.2.x/spring-framework-reference/html/

[7] JDOM. Jdom [En línea]. "Website", Enero 2013.
http://www.jdom.org/

Universidad Autónoma Metropolitana

Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería

Generador de soluciones al problema de Reservación y Cotización de Viajes utilizando la Composición Automatizada de Servicios Web

Manual de usuario

Alumno
Daniel Armando Malagón Mercado
207303605

Asesora
Dra. Maricela Claudia Bravo Conteras
Departamento de sistemas

Abril 2013

Contenido

Requisitos mínimos

- Sistema operativo: Windows XP
- Java Development Kit 6
- Apache Tomcat 6
- Mysql Server 5.1

Instrucciones para usar la aplicación

La aplicación funciona sobre JAVA Enteprise Edition y ademas necesita tener un entorno de Apache Tomcat funcionando, asi como un servidor mysql, se pueden consultar los anexos para una guía rápida de instalación de cada uno de estos programas. Una vez que los requisitos previos sean correctos es necesario preparar el entorno de para poder ejecutar el programa.

1. Montaje de la base de datos.

Iniciamos el cliente mysql, la instrucción dependerá de si el servidor es remoto o si es local, en este manual se tratara el caso de servidor remoto con una computadora cliente usando windows. Entramos a la computadora servidor usando nuestro usuario shell y entramos al cliente de mysql

mysql -u servicios -p

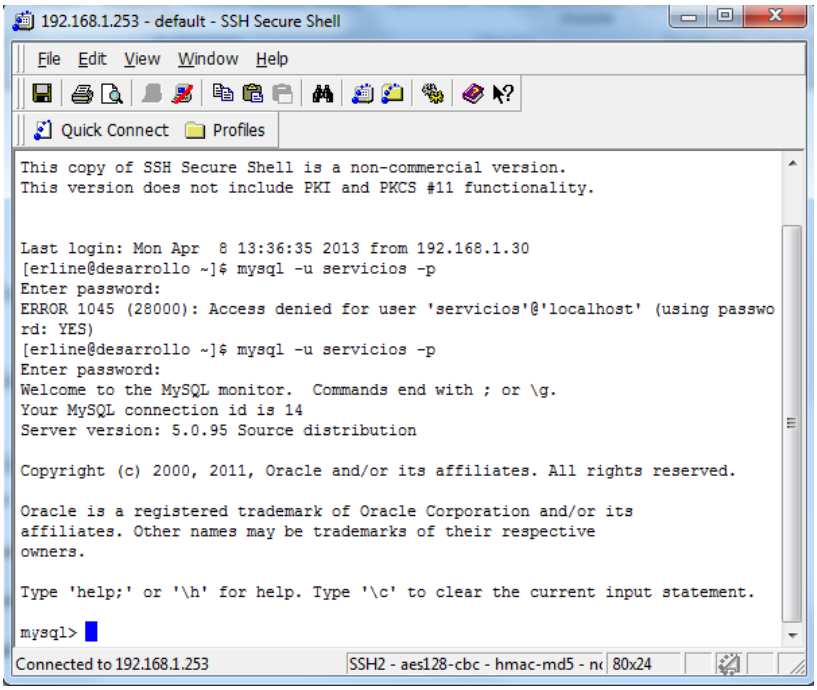


Figura 1 Pantalla principal de mysql

Creamos la base de datos servicios que es necesaria para ejecutar la aplicación

create database servicios;
use servicios;

Importamos las tablas y la estructura de la base de datos a partir del archivo .sql source "ruta del archivo.sql" la figura 2 muestra un ejemplo de ejecución de este comando

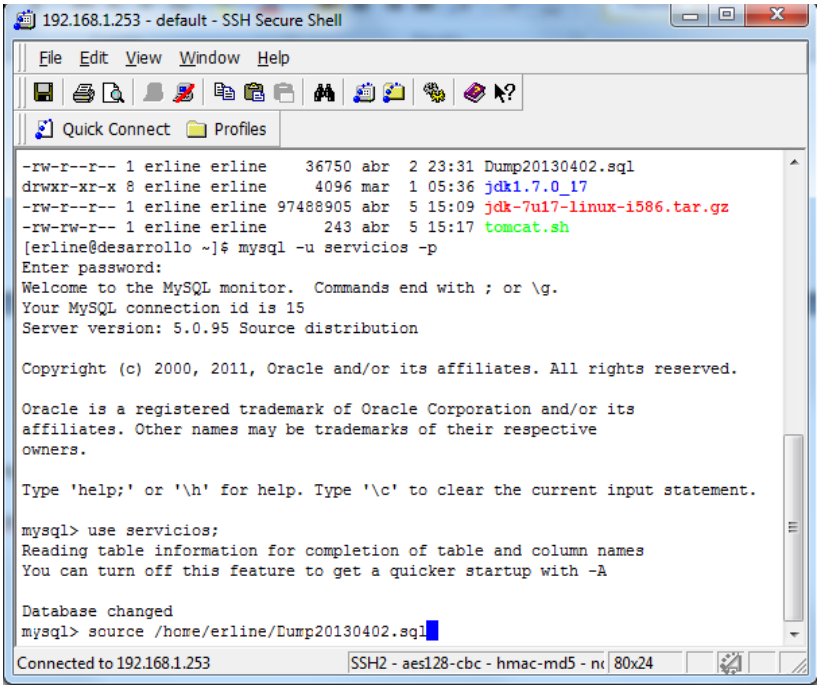


Figura 2 importacion de la base de datos

Cuando el comando haya finalizado la base de datos estará correctamente montada y estará lista para ejecutar el programa.

2. Preparación de la aplicación para su uso

Una vez que se tiene configurado el apache tomcat y que se comprobó que funciona correctamente debemos enviar el archivo .war al directorio de trabajo correspondiente, para esto hay 2 caminos: utilizar el archivo precompilado contenido en el CD o generar el archivo .war a partir del código fuente, esta segunda opción se recomienda si el servidor de ejecución y el servidor de base de datos no están ubicados en la misma computadora o si es necesario configurar la conexion mysql porque la configuracion predeterminada no funciona.

2.1. Montaje de la aplicación en la misma computadora de la base de datos.

Basta con mover el archivo .war del cd al directorio $CATALINA_HOME/webapps automáticamente tomcat montara la aplicación y estará lista para su ejecución.

2.2. Generación del archivo .war y montaje de la aplicación.

Es necesario contar con un software de desarrollo de JAVA Enterprise Edition, se recomienda el uso de Eclipse o algún derivado o Netbeans. Para el manual se utilizara Spring Tool Suite.

Se copia la carpeta del código fuente del proyecto a la carpeta de los proyectos del entorno de desarrollo elegido. Al abrir el proyecto mostrara el árbol de archivos del proyecto y el código fuente del mismo.

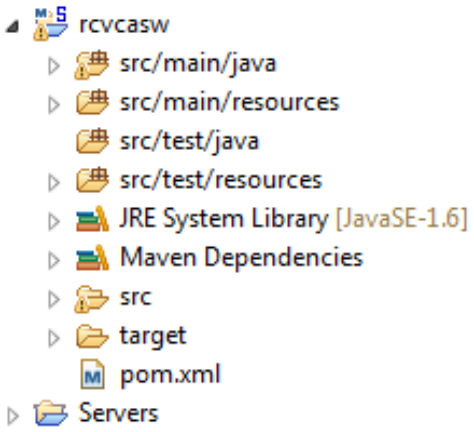


Figura 3 Vista del árbol de directorios del código fuente

Para cambiar los usuarios y conexiones basta con navegar hasta el archivo datasource.properties la ruta completa del archivo es:

src/main/webapp/web_inf/properties/datasource.properties

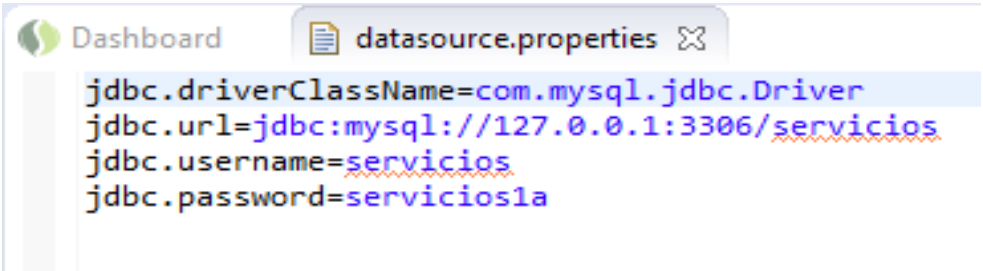
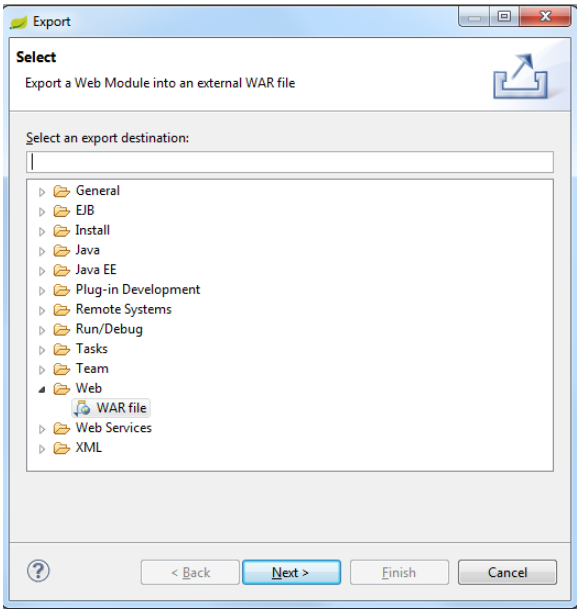


Figura 4 Contenido del archivo datasource.properties

Este archivo contiene la información necesaria para realizar la conexión a la base de datos, si el servidor es remoto se tiene que colocar la dirección remota y el usuario designado para el uso de la base de datos.

Creación del archivo .war

En el caso de eclipse y de sus derivados ejecutamos la utilidad exportar, click derecho en el proyecto y seleccionar exportar, seleccionamos web y escogemos el archivo .war, llenamos los datos que nos pide y el programa creara el archivo .war.

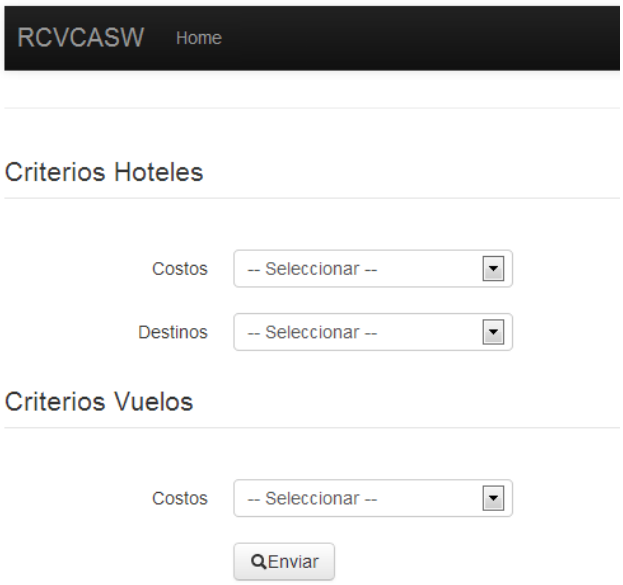


Cuando el archivo está listo basta con mover el archivo .war del cd al directorio $CATALINA_HOME/webapps automáticamente tomcat montara la aplicacion y estará lista para su ejecución.

3. Ejecución del programa

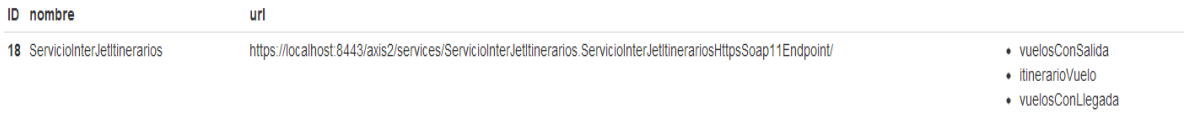
Entramos a la dirección de la instalación de tomcat y abrimos el proyecto agregando al final de la url el nombre del archivo .war sin la extensión. Se abrirá la ventana principal del programa.

Esta ventana muestra las palabras claves que servirán para filtrar los servicios web, para este caso se manejaran servicios de vuelos y de hoteles y sus filtros serán los costos y los destinos, la pestaña de destinos no aparece en los vuelos ya que se asume que se viajara al mismo lugar, una vez seleccionadas las características basta con presionar el botón enviar.



Interpretación de los resultados

Después de seleccionar los filtros el programa mostrara los resultados de hacer el filtrado y mostrara en pantalla los servicios que cumplen con las condiciones seleccionadas, el programa muestra 2 secciones claramente identificadas, en la primera, muestra toda la información referente a los servicios, el id (el cual sirve como identificación para la base de datos) el nombre, del servicio, la url de invocación del servicio y las operaciones que tiene el servicio.



Al darle click en el nombre de las operaciones el sistema muestra los parámetros de entradas y de salida de ese método en específico.

- vuelosConSalida
  - ns:lineaDetalleRequest
  - ns:lineaDetalleResponse
  - ns:lineasConDestinoFinalRequest
  - ns:lineasConDestinoFinalResponse
  - ns:verFrecuenciaDePasoRequest
  - ns:verFrecuenciaDePasoResponse
  - ns:lineasConDestinoInicioRequest
  - ns:lineasConDestinoInicioResponse
  - ns:paradasDeLineaRequest
  - ns:paradasDeLineaResponse
  - ns:horariosLineaRequest
  - ns:horariosLineaResponse
- itinerarioVuelo
  - ns:consultarDescuentosRequest
  - ns:consultarDescuentosResponse
  - ns:clasificacionDescuentosRequest
  - ns:clasificacionDescuentosResponse
  - ns:verDescuentoRequest
  - ns:verDescuentoResponse
- vuelosConLlegada
  - ns:obtenerDescientosSterenRequest
  - ns:obtenerDescientosSterenResponse
  - ns:verClasificacionesRequest
  - ns:verClasificacionesResponse
  - ns:detalleDescuentoRequest
  - ns:detalleDescuentoResponse

La parte de abajo de la pantalla muestra la composición, muestra el id del servicio y con cual otro servicio puede juntarse para resolver el problema.

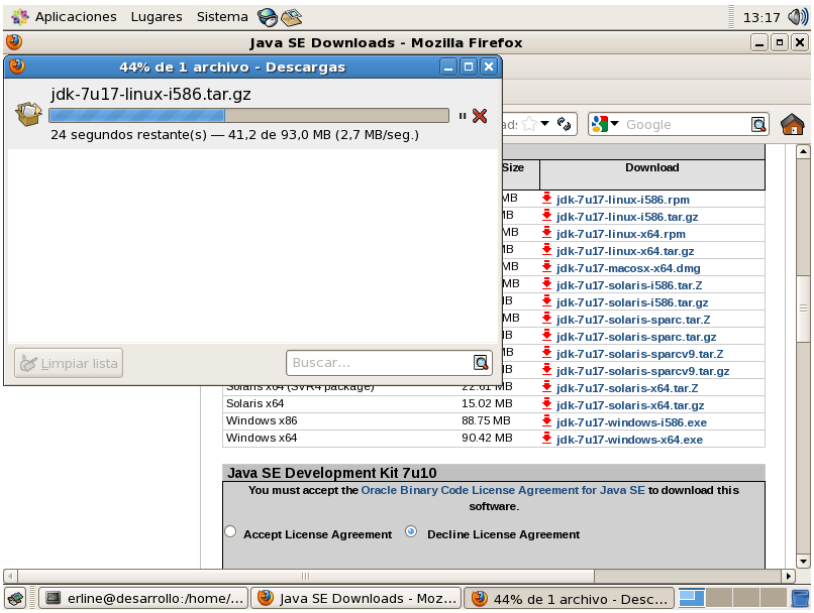
Combinacion de servicios

idVuelo	idHotel
27	18
31	18
39	18
27	22
31	22
39	22
27	44
31	44
39	44

Anexo 1 Instalación de JDK

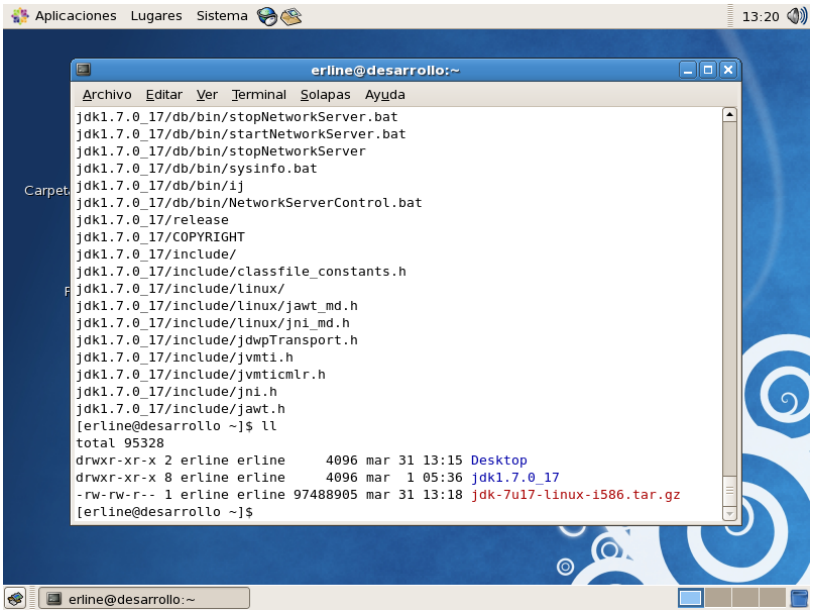
Se asume que el usuario no cuenta con privilegios de root.

1. Descargar el instalador del jdk de la pagina java.oracle.com, descarga la versión comprimida en .tar



2. Colocarse en el directorio home y extraer el instalador:

tar xvf instalador.tar.gz

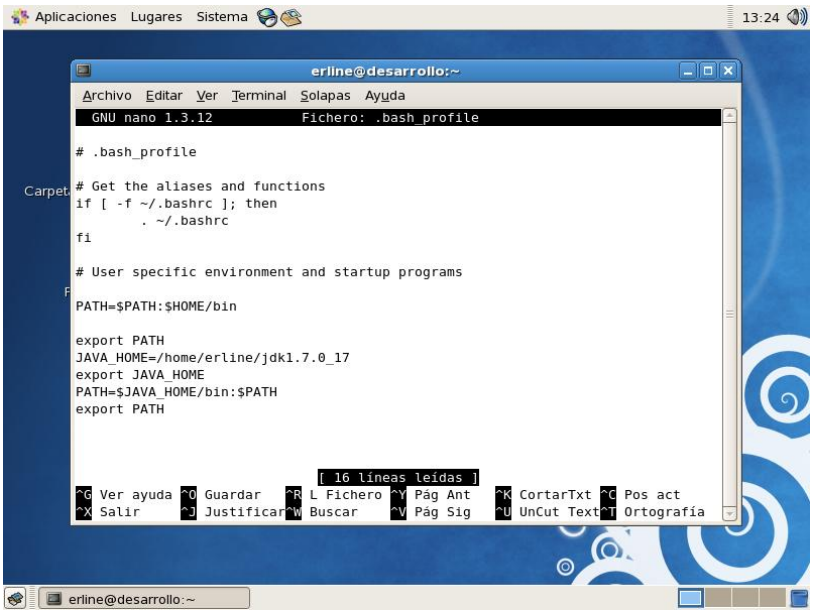


3. Crear las variables de entorno JAVA

nano .bash_profile y agregamos estas líneas.

JAVA_HOME=/home/usuario/java/jdk1.7.0_05

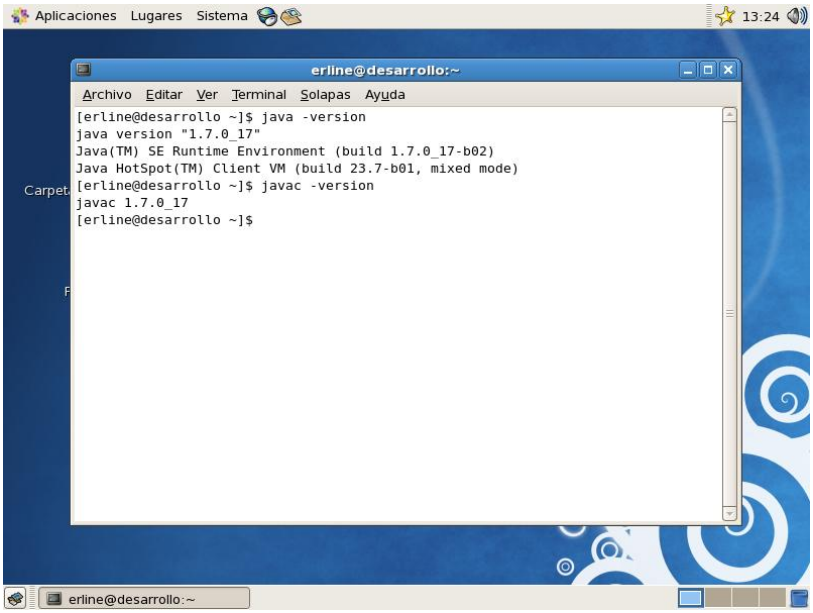
export JAVA_HOME
PATH=$JAVA_HOME/bin:$PATH
export PATH



Cerrar la sesión e iniciarla de nuevo

4. Verificar la instalación de java

java –version
javac-version

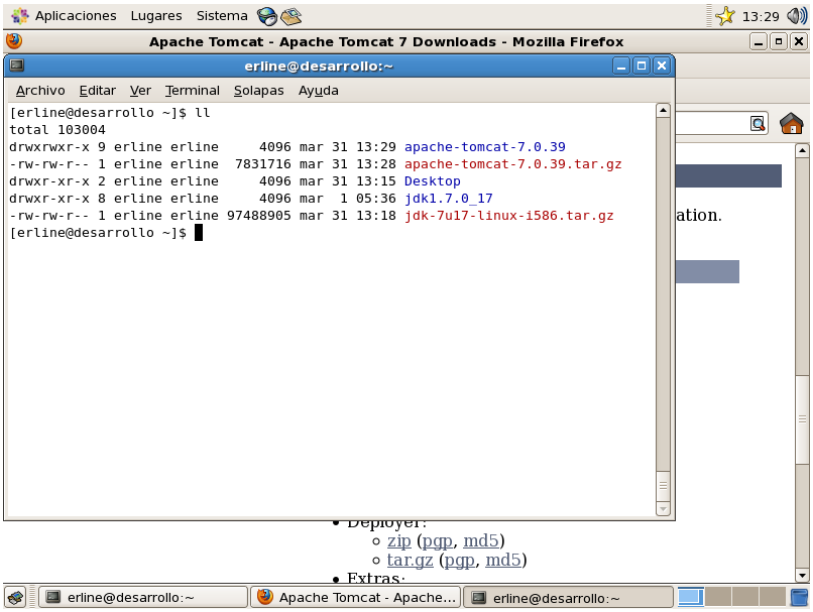


Si los comandos devuelven algún valor con la versión de Java se ah completado la instalación de Java correctamente.

Anexo 2 Instalación de Tomcat

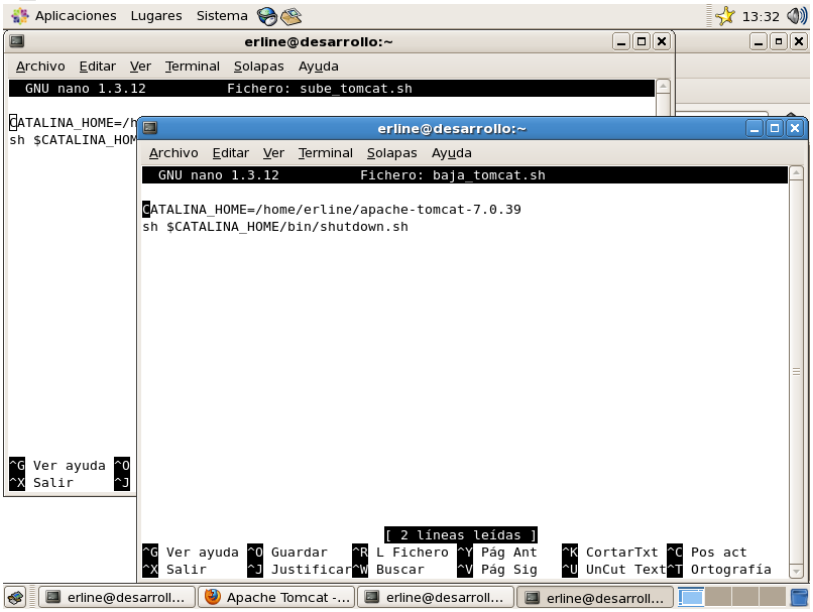
1. Descargar la versión tar.gz de apache tomcat versión 7 y descomprimir el archivo tar

tar xvf archivo.tar.gz



2. Creación de los scripts de ejecución de Tomcat

CATALINA_HOME=
sh $CATALINA_HOME/bin/startup.sh
CATALINA_HOME=
Sh $CATALINA_HOME/bin/shutdown.sh



3.-Configuracion de Tomcat

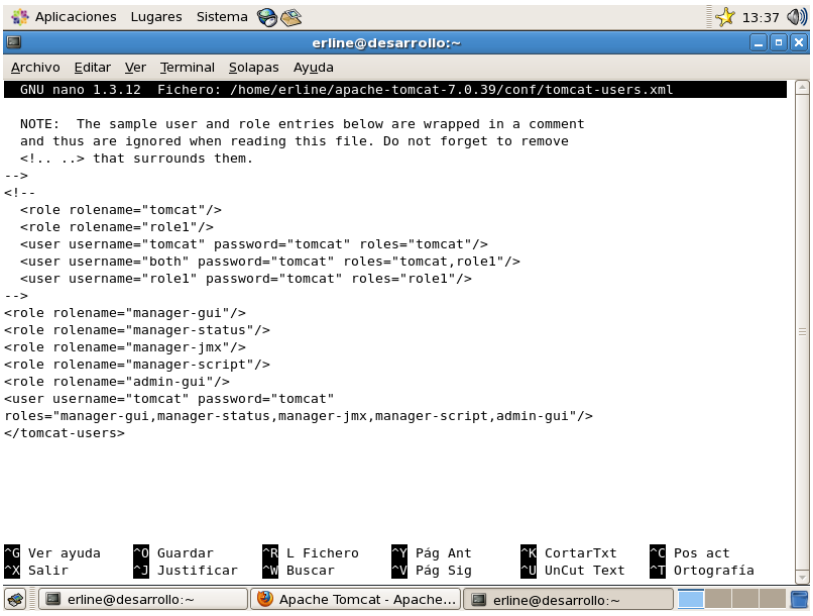
Tomcat requiere de algunos privilegios para su correcto funcionamiento, en esta guía se creara un usuario que realice la administración total del servidor

Buscar el archivo de configuración de Tomcat

nano $CATALINA_HOME/conf/tomcat-users.xml

Agregar el usuario y sus roles

<role rolename=”manager-gui”/>
<role rolename=”manager-status”/>
<role rolename=”manager-jmx”/>
<role rolename=”manager-script”/>
<role rolename=”admin-gui”/>
<user username=”usuario” password “password” roles=”manager-gui,manager-status,manager-jmx,manager-script,admin-gui”/>

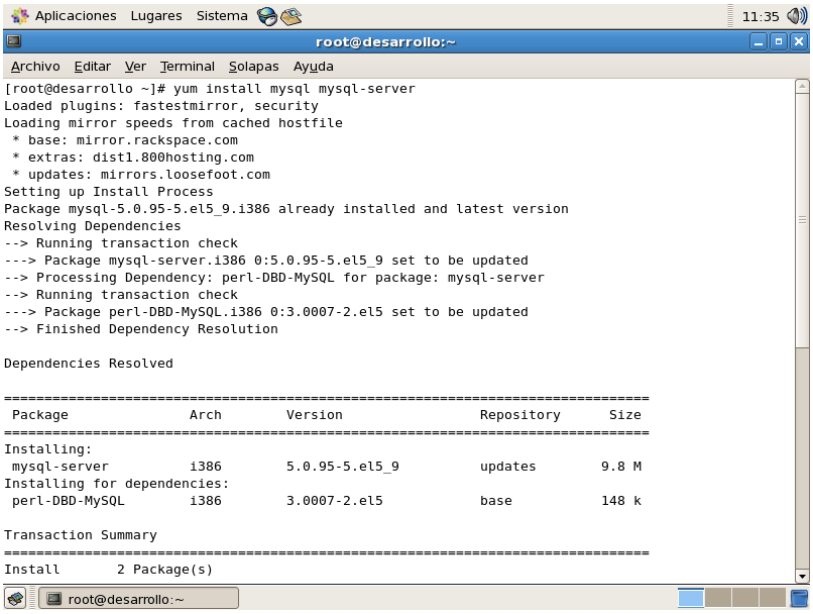


Guardamos el archivo y reiniciamos tomcat con los scripts creados anteriormente

Anexo 3 Instalación y Configuración de un servidor Mysql

1. Instalación Mysql-Server usando los repositorios de Centos

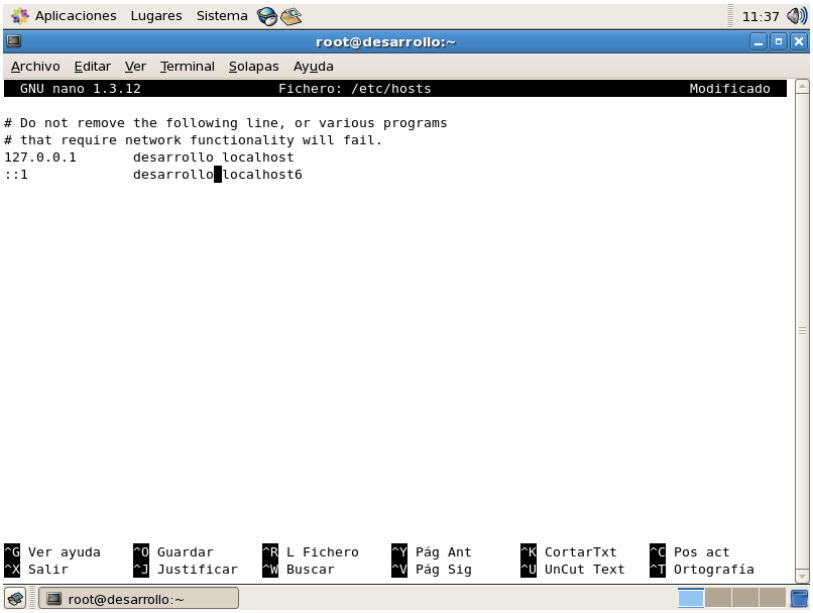
yum -y install mysql mysql-server



2. Modificación del archivo hosts para un arranque sin errores ni advertencias

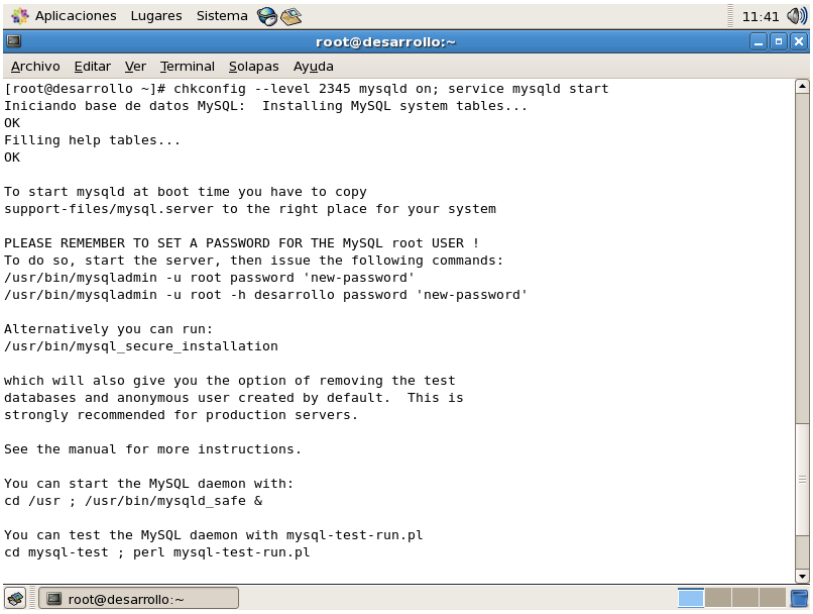
nano /etc/hosts

Eliminamos localhost.localdomain y ponemos el nombre de nuestra maquina



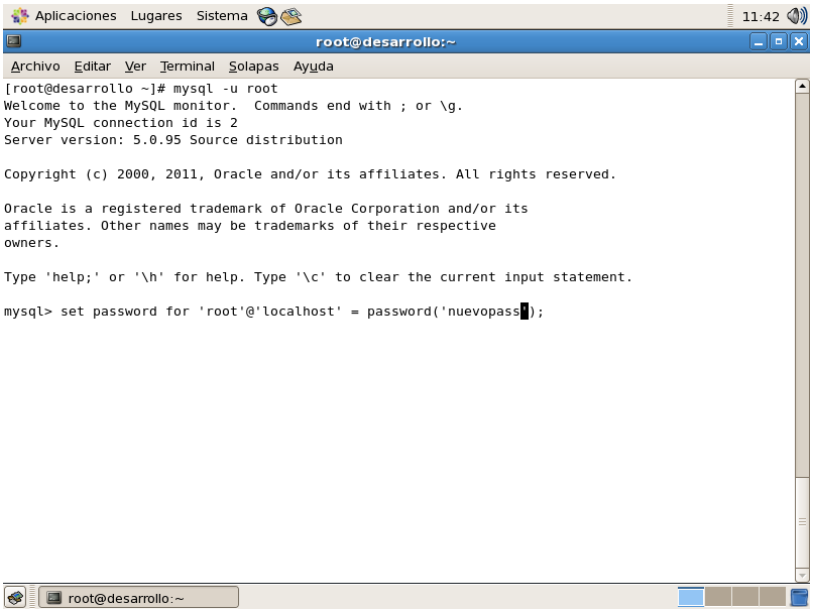
3. Iniciar el demonio de Mysql para arranque automático al iniciar el servidor

chkconfig – level 2345 msyqld on ; service mysqld start



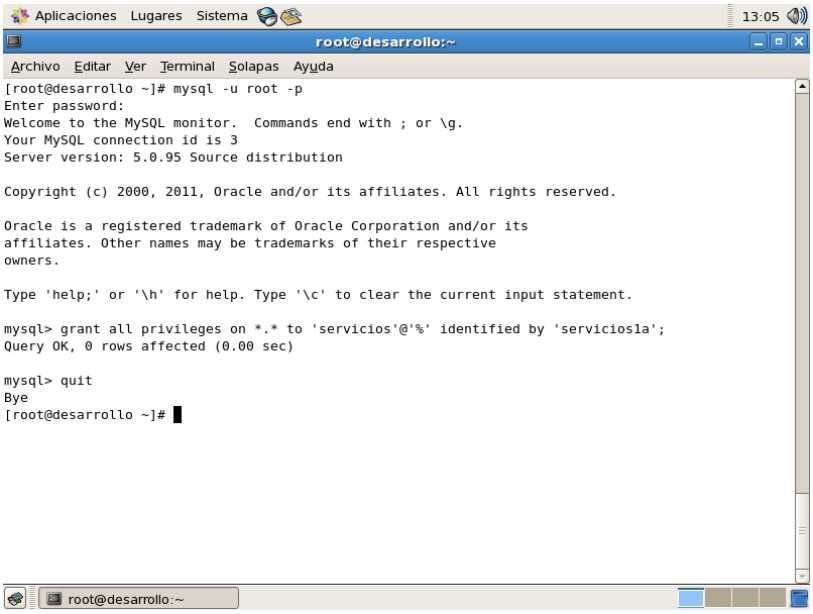
4. Establecer password para el usuario root

mysql –u root
Set password for 'root'@'localhost' = password ('password');
Flush privileges



5. Agregar un usuario remoto con permisos para todas las bases de datos

*Grant all privileges on *.* to 'usuario'@'%' identified by 'password';*

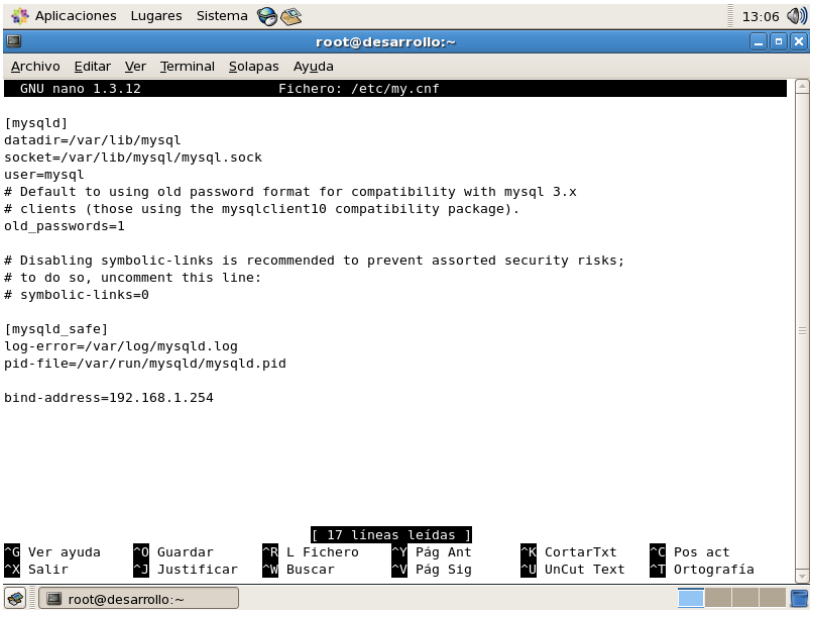


6. Permitir acceso remoto de mysql

nano /etc/my.cnf

Agregamos esta línea al archivo:

bind-address = ip del servidor



7. Reiniciar el servicio mysql

Service mysqld restart

La instalación de mysql ah terminado.