Universidad Autónoma Metropolitana Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería

Licenciatura en Ingeniería en Computación

Proyecto tecnológico

Sistema de búsqueda de coincidencias de servicios web

Yara Lizbeth De La Cruz Hernández

208331774

Trimestre 14-I

Asesor: Maricela Claudia Bravo Contreras.

Profesor Asociado

Departamento de Sistemas

10 de Abril del 2014

Yo, Maricela Claudia Bravo Contreras., declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Firma

Yo, Yara Lizbeth de la Cruz Hernández, doy mi autorización a la Coordinación de Servicios de Información de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco, para publicar el presente documento en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Firma

Resumen.

El sistema realizado consiste de una aplicación web, que sirve para publicar o buscar servicios web, la cual requiere haber realizado previamente el registro del usuario dentro del sistema. El usuario puede seleccionar su registro para proveedor en caso que desee publicar algun servicio web en WSDL, o como cliente si desea buscar.
Cuando se realiza el registro del servicio el sistema se encarga de buscar alguna coincidencia y de no encontrarla en el momento esta mantiene el registro de la busqueda hasta que se encuentre alguna, una vez encontrada alguna coincidencia el sistema envia la notificación al cliente quien se encarga de decidir contactar al proveedor para obtener el servicio. La relacion que exista entre ellos ya sera externa al sistema.

Índice de contenido:

Índice de figuras.

Índice de códigos.

1 . Introducción

Existen en la Internet una gran cantidad de recursos de software disponibles en forma de servicios web para ser utilizados y reutilizados por programadores y agentes de software inteligentes. Los servicios web son componentes de software basados en un conjunto de protocolos, lenguajes y estándares interoperables de Internet, que resuelven los problemas de heterogeneidad entre aplicaciones empresariales desarrolladas en diferentes lenguajes de programación. Para que un desarrollador de aplicaciones empresariales pueda hacer uso de dichos servicios web, debe realizar búsquedas exhaustivas a través de buscadores comunes como Google y revisar largas listas de resultados para verificar si el servicio que requiere se encuentra dentro de la lista devuelta por el buscador tradicional. Esta tarea de buscar, revisar y descartar servicios implica gran cantidad de tiempo y es muy propensa al error humano.

En esta propuesta de proyecto terminal se propone diseñar y desarrollar un sistema que permita automatizar la búsqueda de servicios mediante la incorporación de un motor de búsquedas de coincidencias, que al recibir la descripción del servicio solicitado por el cliente realice las comparaciones necesarias con los servicios registrados por el proveedor; cuando exista una coincidencia entre los requerimientos del cliente y el servicio el sistema enviará los datos correspondientes del cliente y proveedor al sistema de notificaciones para informar a los respectivos involucrados sobre la coincidencia enviando al cliente la información del proveedor y viceversa. La comunicación entre ambos será externa.

Por ejemplo, supongamos que tres proveedores de servicios de banquetes publican sus servicios Web para la cotización de paquetes, el primer proveedor ofrece banquetes de comida Italiana, el mínimo de porciones es de cincuenta, el paquete puede incluir postre y aguas frutales; el segundo proveedor ofrece comida Mexicana y China el mínimo de porciones es de sesenta y el paquete incluye postre; el tercer proveedor ofrece paquetes de comida China y Árabe, no restringe la cantidad de porciones e incluye postre. Existe un cliente que quiere realizar una fiesta y esta en busca de un paquete de banquete y desea para sus invitados una comida Mexicana con postre y tendrá cien invitados.

El cliente realiza su registro de servicio y el motor comienza a realizar las comparaciones. Al realizar las comparaciones devuelve los datos del segundo proveedor y del cliente, el sistema de notificaciones envía los datos del cliente al proveedor y viceversa. El cliente se comunica con el proveedor de manera externa al sistema.

2 . Justificación

El trabajo que se desarrollará será útil en los siguientes casos:
1. Para desarrolladores de aplicaciones basadas en servicios web, ya que les facilitará la búsqueda y localización de servicios de manera precisa.
2. Para los proveedores de servicios web, porque podrán publicar y ofrecer sus servicios a una comunidad de usuarios más amplia,
3. Para agentes inteligentes, que podrán realizar sus búsquedas y publicaciones de servicios Web.
La elaboración de un motor de búsqueda permitirá a que la identificación de coincidencias se realice de manera automática ayudando al cliente tener una oferta de serviciao eficaz y más cercana a sus necesidades.

3 . Antecedentes

Proyectos terminales:
Extracción automatizada y representación de servicios web mediante ontologías [1]. Este proyecto terminal es similar con esta propuesta en el uso de ontologías para el manejo de servicios Web. Sin embargo, se diferencia en que el modelo ontológico de representación de servicios no está orientado al desarrollo de un motor de búsqueda de coincidencias.

Propuestas de proyecto terminal:
Clasificación de servicios web semánticos mediante ontologías [2]. La propuesta de Sánchez tiene relación con esta propuesta en el uso de ontologías. Pero, se diferencia en que el motor de búsqueda de coincidencias no es para realizar clasificación, su uso es para la comunicación entre cliente y proveedor.
Solución de problemas de mediana dificultad utilizando composición de servicios web [3]. Esta propuesta es semejante en el uso de servicios web y en la selección de servicios de acuerdo a los requerimientos. Se diferencia en que su objetivo principal es la solución de problemas mediante la composición de servicios.

Artículos:
Automatic Matchmaking of Web Services [4]. Este artículo se relaciona con mi propuesta en el uso de servicios web en WSDL y en la búsqueda de servios. Se diferencia en que no se realiza el registro de proveedores ni de clientes.
Flexible Matchmaking of Web Services. Using DAML-S Ontologies [5]. Este artículo se asemeja con mi propuesta en el uso de servicios web, pero se diferencia en que el lenguaje de descripción de ontologías es DAML-S, un lenguaje ontológico diferente de WSDL.
Semantic Matchmaking Algorithms [6]. Este artículo tiene en común con esta propuesta el uso de ontologías en OWL y los servicios web.

4 . Objetivos.

4.1.1 . Objetivo general.

Diseñar e implementar un sistema Web mediante el cual clientes (consumidores de servicios) realicen la publicación de solicitudes de servicios web, proveedores de servicios realicen la publicación de las descripciones de servicios web ofertados, y que a través de la implementación de un motor de búsqueda de coincidencias entre las solicitudes y las ofertas de servicios web se notifique automáticamente a los involucrados sobre la coincidencia.

4.1.2 . Objetivos especificos.

Diseñar e implementar un modulo a través del cual clientes (consumidores de servicios web) publiquen solicitudes de búsqueda incluyendo las restricciones y especificaciones de los servicios buscados.

Diseñar e implementar un modulo mediante el cual los proveedores de servicios web registren sus servicios y levanten la descripción del servicio en WSDL[1].

Diseñar e implementar un motor de búsqueda de coincidencias que esté constantemente revisando la lista de solicitudes de los clientes y la lista de servicios ofertados, para que al momento de detectar una coincidencia envíe un mensaje a ambos participantes.

Diseñar e implementar un método de búsqueda de coincidencias que implemente mediciones de similitud funcional[2] entre servicios web ofertados y solicitudes registradas.

5 . Analísis y modelado del sistema.

5.1. Arquitectura general del sistema.

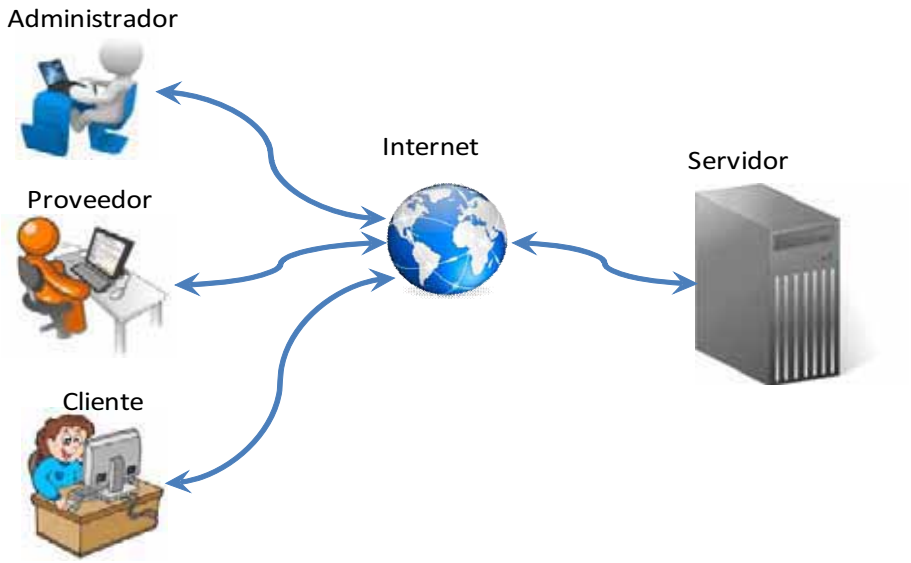


Figura 1: Arquitectura general del sistema.

En la Figura 1 se muestra la arquitectura del sistema, en la cual existen 3 tipos de usuarios uno es el administrador, otro el proveedor y el cliente cada uno cuenta con una interfaz de navegación distinta; el administrador como su nombre lo indica administra a los usuarios(proveedor y cliente), el cliente realiza su registro de datos y servicios al cual son enviados al servidor y este contesta cuando encuentra alguna coincidencia con el proveedor que registra sus datos y servicios en WSDL.
El servidor contiene la base de datos de los usuarios, servicios y busquedas realizadas, asi como el motor que ayuda a localizar las coincidencias de servicios.

[1]WSDL (web service description language por sus siglas en ingles) documento de la especificación de los servicios.
[2] La funcionalidad de un servicio Web considera entradas, salidas, pre-condiciones y efectos.

5.1.1 . Diseño por Capas.

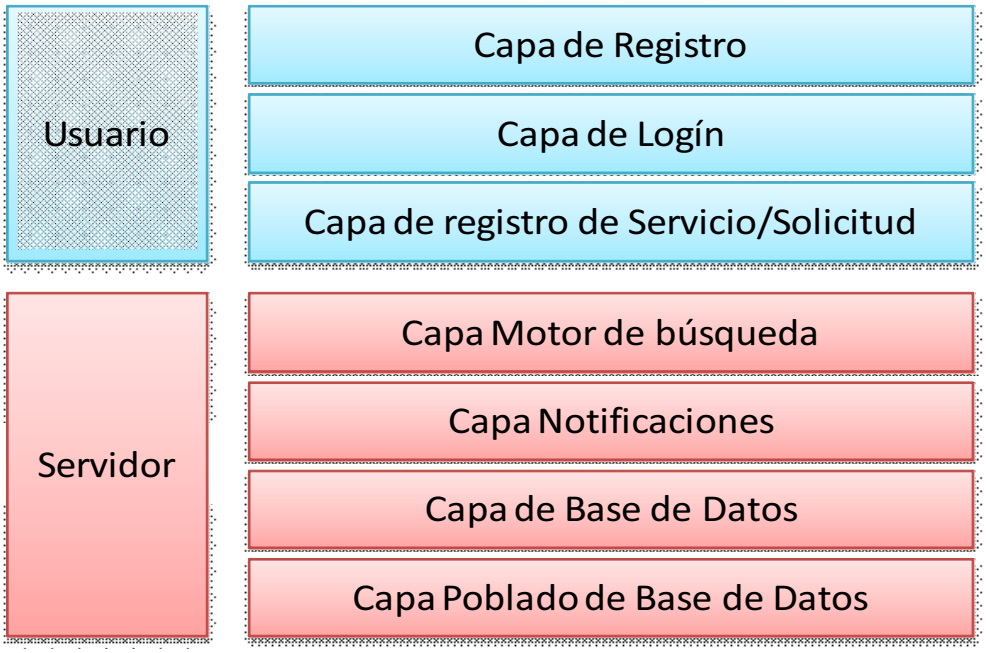


Figura 2: Capas de la arquitectura.

En la figuara 2 se muestra el diseño de capas de la arquitectura, las 3 capas correspondientes a los usuarios y 4 capas al servidor.

5.1.1.1 Capa de Registro.

En esta capa el usuario realiza su respectivo registro al sistema; el registro puede realizarlo como Proveedor o Cliente, esto introduciendo datos personales como nombre, dirección, e-mail, celular, así como también una contraseña para poder realizar el login.

5.1.1.2 Capa de Login.

El usuario ingresa su nombre y contraseña en los cuadros de texto, el sistema los procesa y devuelve en pantalla la sesión del cliente o proveedor logueado.

5.1.1.3 Capa de Registro de Servicio/Solicitud.

El usuario Cliente puede realizar su registro de solicitud introduciendo Nombre del servicio, de operacion y de parametros de entrada y salida.

El usuario Proveedor ofrece su servicio por medio de un archivo WSDL que el sistema parsea y guarada los datos necesarios en la base de datos.

5.1.1.4 Capa Motor de Búsqueda.

El servidor al recibir algún registro por parte del proveedor y alguno por el cliente este comienza a realizar las búsquedas de coincidencias y si encuentra alguna se envia la notificación.

5.1.1.5 Capa de Notificaciones.

El sistema al hallar alguna coincidencia se envia la notificación al cliente, que es el que decide contactar o no al proveedor.

5.1.1.6 Capa de Base de datos.

Los datos proporcionados por los usuarios son arreglados en una base de datos que con ayuda del administrador puede modificar y eliminar a usuarios.

5.1.1.7 Capa de Poblado de Base de datos.

La base de datos es poblada en partes, por lo que al hacer el registro los datos son almacenados, y asi sucesivamente en cada registro realizado.

5.2. Modelado de la Capa Usuario.

5.2.1 . Casos de uso de usuario.

Existen 3 tipos de usuarios que son Administrador, Cliente y Proveedor; cada uno tiene privilegios distintos es por ello que se describiran de la siguiente manera.

5.2.1.1 Casos de uso de Administrador.

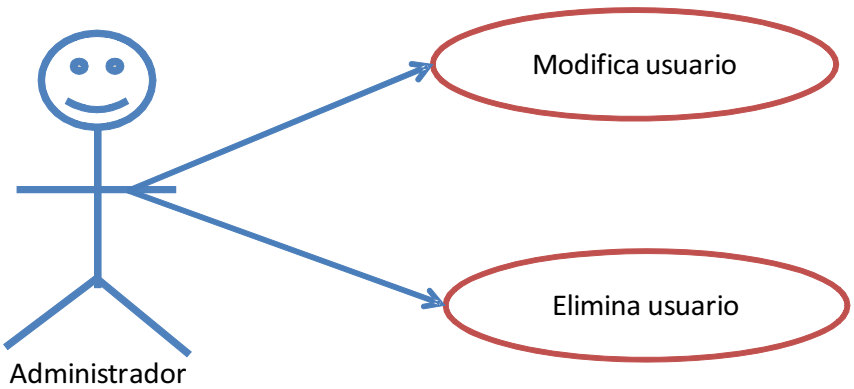


Figura 3. Casos de uso del Administrador.

Actores:

- Administrador: Es quien solo puede modificar y eliminar a los clientes y proveedores.

Casos de uso:

1. Modifica usuario. El administrador es el unico que puede modificar los datos de los proveedores y clientes registrador en el sistema.
2. Elimina usuario. De igual manera el administrador es el unico que puede eliminar a algun usuario ya sea cliente o proveedor.

5.2.1.2 Casos de uso de Cliente.

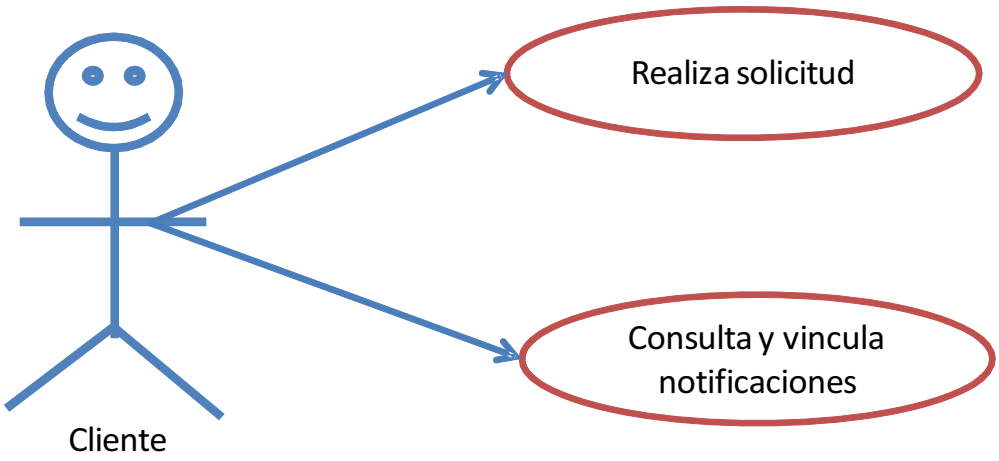


Figura 4. Casos de uso del Cliente.

Actores:

- Cliente. Es el que realiza alguna busqueda de servicio con solo introducir el nombre del servicio, de operaciones y parametros de entrada y salida.

Casos de uso:

1. Realiza solicitud. El cliente puede realizar muchas solicitudes introduciendo solo alguna parte de lo que desea.
2. Consulta y vincula notificaciones. El sistema le notifica de las coincidencias encontradas las cuales el cliente puede consultarlas y decide si las vincula para que tenga comunicación con el proveedor.

5.2.1.3 Casos de uso de Proveedor.

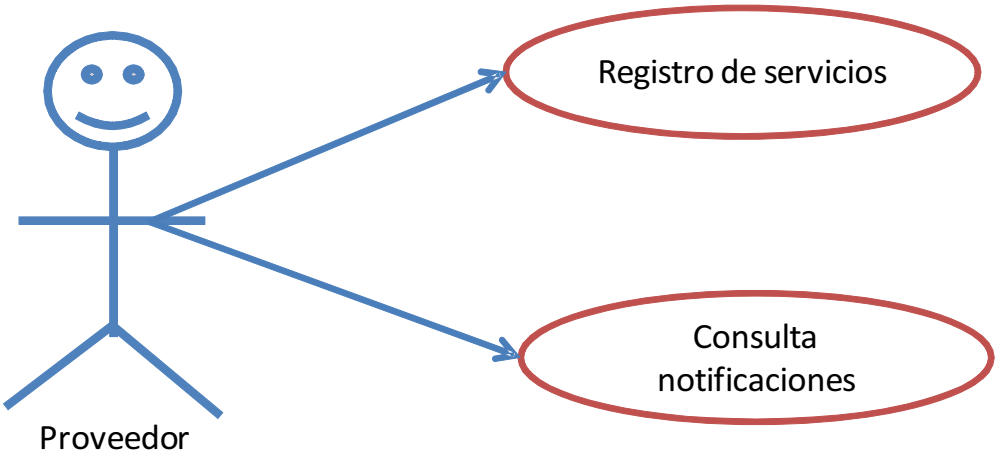


Figura 5. Casos de uso del Proveedor.

Actores:

- Proveedor: Es quien ofrece servicios en WSDL y esta en espera de las notificaciones del cliente.

Casos de uso:

1. Registro de servicios. El proveedor sube un archivo en WSDL y el sistema almacena la información en base de datos.
2. Consulta notificaciones. El cliente si lo deseo envia notificaciones al proveedor, esta las puede consultar en su sesión o al correo registrado.

5.3. Modelado de la capa Servidor.

5.3.1 . Motor de Búsquedas.

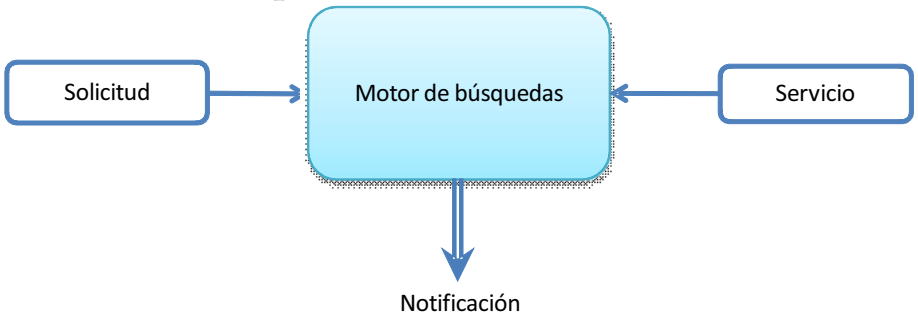


Figura 6. Diagrama del motor.

En el diagrama 6. se muestra el como el motor de busqueda de coincidencias se ejecuta, es decir, el motor comienza a realizar su función cuando le llega una solicitud por parte del cliente y una por el proveedor, este al encontrar una coincidencia la envia como notificación esto solo al cliente.

5.3.2 . Capa de Notificaciones.

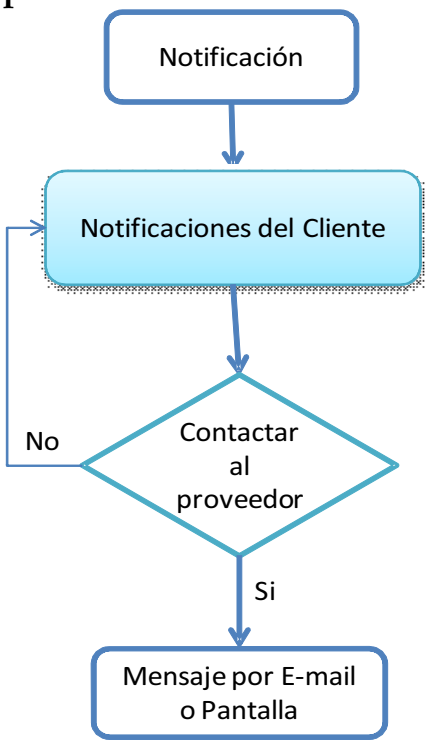


Figura 7. Diagrama de Notificaciones.

En el diagrama muestra la secuencia que se realiza al encontrar alguna coincidencia, esto es si al cliente le llega alguna notificación el decide si contacta al proveedor que le ofrece el servicio de acuerdo a lo que el solicitó, si decide enviar este podra enviar un mensaje via e-mail(proporcionado en el registro) y a su vez se enviara al sistema en al sesion del proveedor, en caso contrario esta notificación seguira mostrandose en pantalla del cliente, esto con el fin de contactarlo en otro momento.

5.4. Modelado de la Base de Datos.

La base de datos esta formada de la siguiente manera figura 8:

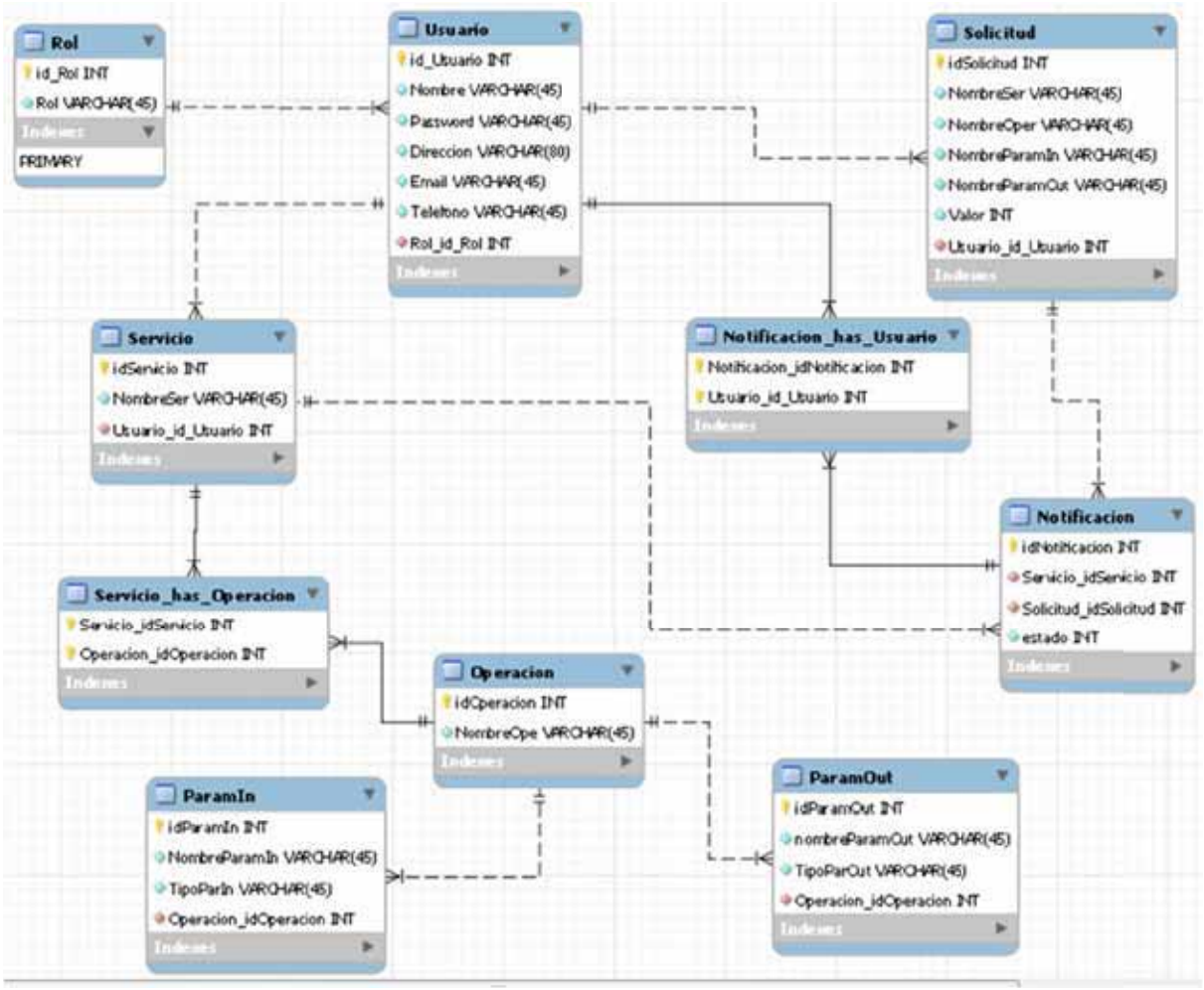


Figura 8. Diagrama Entidad-Relacion del sistema.

6 . Implementación.

Ya que se realizó el análisis y diseño del sistema, se realiza la implementación y codificación del sistema, para ello nos apoyamos en software libre, la programación realizada en lenguaje Java.

6.1. Tecnologias de implementación.

6.1.1 . IDE Netbeans 7.2.1

NetBeans[7] es un entorno de desarrollo integrado libre, hecho principalmente para el lenguaje de programacion Java. NetBeans IDE es un producto libre y gratuito sin restricciones de uso.

6.1.2 . PhpMyAdmin 4.0.3

PhpMyAdmin[8] es un software de código abierto, diseñado para manejar la administración y gestión de bases de datos MySQL a través de una interfaz gráfica de usuario

6.1.3 . JavaMail API.

JavaMail[9] es una expansión de java que facilita el envío y recepción de e-mail desde código java.
JavaMail implementa el protocolo SMTP (Simple Mail Transfer Protocol) así como los distintos tipos de conexión con servidores de correo -TLS, SSL, autentificación con usuario y password.

6.1.4 . PrimeFaces.

PrimeFaces[10] es una librería de componentes visuales para Java Server Faces (JSF) de código abierto que cuenta con gran cantidad de componentes que facilitan la creacioón de las aplicaciones Web.

6.1.5 . Apache Tomcat.

Apache Tomcat[11] es una implementación de software de código abierto que funciona con un contenedor de aplicaciones de *Java Servlet*2 y tecnologías *JavaServer Pages*3, se usa en la capa del servidor proporcionando soporte para el servicio web.

6.1.6 . Apache Axis2/Java

Apache Axis2/Java[12] es un motor para servicios web, es usado para el despliegue de servicios web dentro de Apache Tomcat.

6.2. Capa Usuario.

En esta capa se muestra como estan ligadas las difentes capas existentes y el seguimiento que realiza el usuario para llegar a ellas.

6.2.1 . Capa login.

Al iniciar el sistema la primera panatalla que muestra es la de login Figura 9, pero, para poder acceder es necesario tener un usuario y contraseña por lo que se debe realizar el registro. Para esto hay que dar clic en el Registrate.

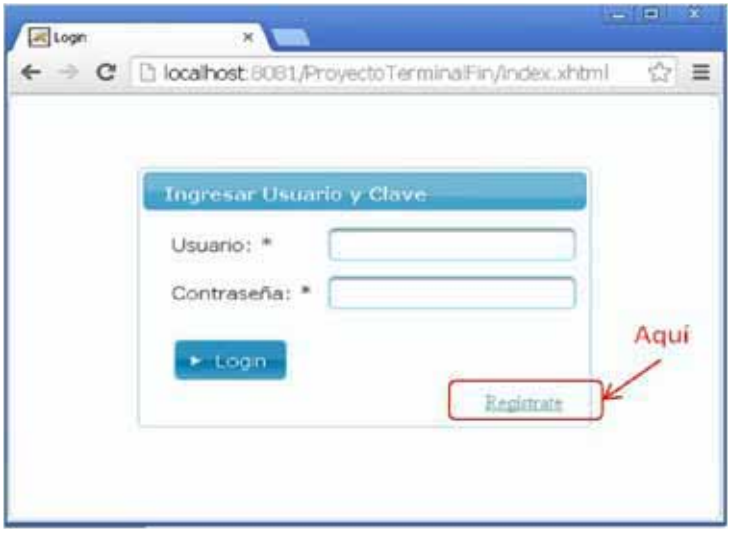


Figura 9. Pantalla de inicio login.

En el Codigo 1. muestra como la liga de Registrate manda a otra nueva pantalla

```
f:view  h:body
<h:body>
    <p:layout fullPage="true">
        <p:layoutUnit position="center" >
            <div class="conteiner">
                <p:growl id="growl" showDetail="true" life="3000" />
                <h:form id="formlogin">
                    <p:panel id="panel" header="Ingresar Usuario y Clave">
                        <h:panelGrid columns="2" cellpadding="5">
                            <p:outputLabel for="username" value="Usuario:" />
                            <p:inputText value="#{loginBean.username}"
                                         id="username" required="true" label="cliente" />

                            <p:outputLabel for="password" value="Contraseña:" />
                            <p:password value="#{loginBean.password}"
                                        id="password" required="true" label="password"/>

                            <f:facet name="footer">
                                <p:commandButton id="loginButton" value="Login" update="growl"
                                                 actionListener="#{loginBean.login()}" icon="ui-icon-play"
                                                 oncomplete="handleLoginRequest(xhr, status, args)"/>
                            </f:facet>
                            <p:growl id="messages" showDetail="true"/>
                        </h:panelGrid>
                        <h:link outcome="Registro" value="Registrate"
                    style="padding-left: 80%; font-family: Helvica; font-size: medium; color: cadetblue"/>
                        </p:panel>
                </h:form>
```

Codigo 1. Index del sistema.

6.2.2 . Capa de Registro de usuario.

Se puede realizar el registro como Cliente y como Proveedor. El unico apartado en el que cambia es el tipo de usuario que se seleccion en la parte superior. Figura 10.

Figura 10. Registro de Usuario.

En el Codigo 2. Se puede constatar como a excepción del domicilio los demás datos son requeridos para poder realizar su registro.

```
<h:form id="formReg">
    <p:panel header="Registro de nuevo Usuario">
        <h:panelGrid cellpadding="8" columns="2">
            <p:outputLabel id="rol"/>
            <p:selectOneMenu required="true" id="Tipo" label="Usuario" value="#{adminBean.selectedUsuario.rolIdRo}">
                <f:selectItem itemLabel="Tipo de Usuario" itemValue="" />
                <f:selectItem itemLabel="Cliente" itemValue="3"/>
                <f:selectItem itemLabel="Proveedor" itemValue="2"/>
            </p:selectOneMenu>
            <p:outputLabel value="Nombre:" />
            <p:inputText id="Nom" required="true" label="Nombre" size="45" value="#{adminBean.selectedUsuario.nombre}"/>
            <p:outputLabel value="Direccion:" />
            <p:inputText id="Direcc" size="60" value="#{adminBean.selectedUsuario.direccion}"/>
            <p:outputLabel value="E-mail:" />
            <p:inputText  id="mail" value="#{adminBean.selectedUsuario.email}" size="35" required="false" label="Email"
                          validatorMessage="Email: Error de formato" >
                <f:validateRegex
                    pattern="^[_A-Za-z0-9-\+]+(\.[_A-Za-z0-9-]+)*@[A-Za-z0-9-]+(\.[A-Za-z0-9]+)*(\.[A-Za-z]{2,})$" />
             </p:inputText>
             <p:outputLabel value="Celular:" />
             <p:inputMask required="true" id="Tel" mask="(99) 9999-9999" label="Celular" size="15"
                          value="#{adminBean.selectedUsuario.telefono}"/>
             <p:outputLabel value="Contraseña:" />
             <p:password id="Pass" label="Contraseña" required="true" feedback="true"  promptLabel="Introduce contraseña"
                         match="Pass2"  validatorMessage="Las contraseñas no conciden" weakLabel="Baja"
                         goodLabel="Buena" strongLabel="Excelente" value="#{adminBean.selectedUsuario.password}"/>
             <p:outputLabel value="Repetir contraseña:" />
             <p:password id="Pass2" label="Contraseña" required="true" feedback="true"  promptLabel="Introduce contraseña"
                         weakLabel="Baja"
                         goodLabel="Buena" strongLabel="Excelente"
                         value="#{adminBean.selectedUsuario.password}"/>

             <f:facet name="footer">
                 <p:separator/>
                 <p:commandButton id="ButtonAcept" value="Guardar" actionListener="#{adminBean.btncreateUsuario()}"
                                  update=":growl" icon="ui-icon-disk" oncomplete="handleLoginRequest(xhr, status, args)"/>
                 <p:commandButton id="ButtonCancel" value="Cancelar" icon="ui-icon-close"
                                  actionListener="#{loginBean.logout()}"/>
             </f:facet>
             <p:growl id="messages" showDetail="true"/>
         </h:panelGrid>
     </p:panel>
 </h:form>
iv>
```

Codigo 2. Registro de Usuarios

6.2.3 . Sesión del usuario Proveedor.

En la sesión del proveedor nos muestra una tabla con 4 apartados los cuales son Datos, Nuevo, Servicios y Notificaciones.

En la pestaña de Datos nos muestra los datos del proveedor proporciondados en el registro. Estos Datos son obtenidos desde la base de datos ver Codigo 3.

En el Código 4. Se ve como para la pestaña de Nuevo nos permirá subir muchos archivos en formato WSDL para ser procesados.

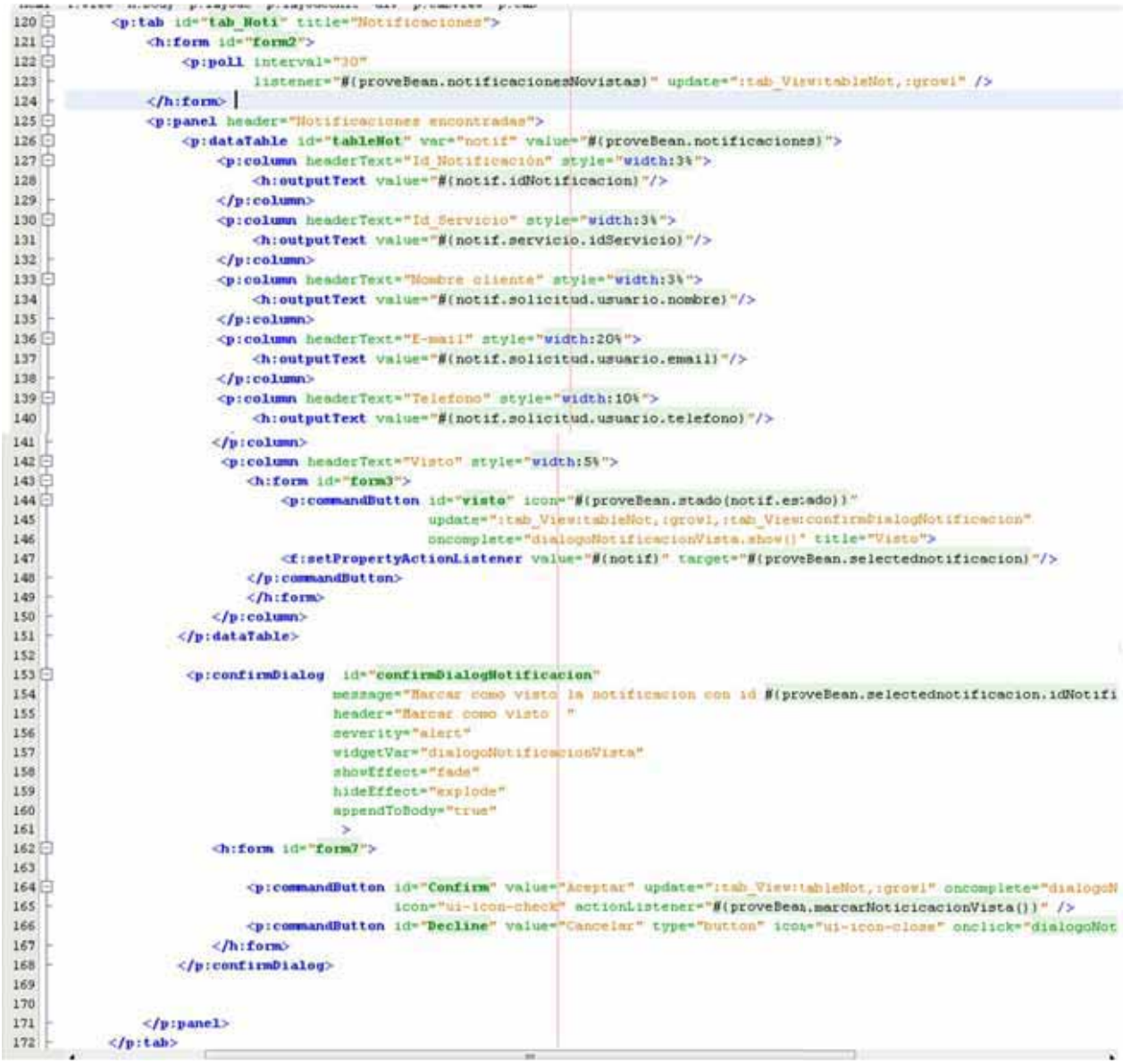
```
30                      <p:tabView id="tab_View">
31                          <p:tab id="tab_Datos" title="Datos">
32                              <p:panel header="Datos personales" style="width: 6in">
33                                  <h:panelGrid cellpadding="10" columns="2">
34                                      <p:outputLabel value="Nombre:" />
35                                      <p:outputLabel value="#{UserBean.nombre}"/>
36                                      <p:outputLabel value="Direccion:" />
37                                      <p:outputLabel value="#{UserBean.direccion}"/>
38                                      <p:outputLabel value="E-mail:" />
39                                      <p:outputLabel value="#{UserBean.mail}"/>
40                                      <p:outputLabel value="Celular:" />
41                                      <p:outputLabel value="#{UserBean.telefono}"/>
42                                      <p:outputLabel value="Contraseña:" />
43                                      <p:outputLabel value="#{UserBean.password}"/>
44                                  </h:panelGrid>
45                              </p:panel>
46                          </p:tab>
47
```

Código 3. Datos del proveedor.

```
48          <p:tab id="tab_Nuevo" title="Nuevo">
49              <p:outputPanel id="panel-toolbar" layout="block"
50                             styleClass="ui-widget-header ui-corner-all"
51                             style="margin-bottom: 10px;">
52                  <h:form id="toolbar-form">
53                      <h:panelGrid columns="7" style="margin-left: 15px;">
54                          <h:outputText value="Selecciona archivo(s) WSDL "/>
55                          <p:spacer width="20"/>
56                          <p:outputPanel layout="block" >
57                              <p:commandButton styleClass="btn" id="btn-publar" value="Guardar"
58                                               update="panel-toolbar,:tab_View:lista-servicios"
59                                               icon="ui-icon-disk"
60                                               actionListener="#{proveBean.guardarServicios}"/>
61                              <p:commandButton styleClass="btn" global="false" id="btn-terminar"
62                                               value="Terminar"
63                                               update=":panel-toolbar,:tab_View:form1,:tab_View:list2"
64                                               disabled="j" icon="ui-icon-check"
65                                               actionListener="#{proveBean.terminarProceso}"/>
66                          </p:outputPanel>
67                          <p:spacer width="20"/>
68                          <p:commandButton styleClass="btn" global="false" id="btn-cancelar"
69                                           value="Cancelar" icon="ui-icon-cancel"
70                                           update=":panel-toolbar,:tab_View:form1,:tab_View:list2"
71                                           actionListener="#{proveBean.cancelarProceso}"/>
72                      </h:panelGrid>
73                  </h:form>
74              </p:outputPanel>
75              <p:messages id="msje" closable="true" autoUpdate="true"/>
76              <h:form id="form1" enctype="multipart/form-data">
77                  <p:scrollPanel style="max-height: 500px;border: none;margin-bottom: 10px;" mode="native">
78
79                      <p:fileUpload  id="subirArchivos" fileUploadListener="#{proveBean.subirArchivos}"
80                                     invalidFileMessage="archivo no valido"
81                                     mode="advanced"
82                                     invalidSizeMessage="tamaño no valido"
83                                     label="Buscar"
84                                     cancelLabel="Cancelar"
85                                     uploadLabel="Subir"
86                                     multiple="true"
87                                     sizeLimit="700000"
88                                     allowTypes="/(\.|\/)(wsdl|WSDL)$/"
89                                     update=":tab_View:list2" >
90                      </p:fileUpload>
91
92
93                  </p:scrollPanel>
94              </h:form>
95              <p:outputPanel id="list2" layout="block">
96                  <p:dataList  rendered="#{proveBean.nomArchivos.size()>0}" value="#{proveBean.nomArchivos}" var="nom
97                               paginator="true" rows="10" rowIndexVar="indice" paginatorPosition="top"
98                               paginatorTemplate=" {CurrentPageReport}  {FirstPageLink} {PreviousPageLink} {PageLinks}
99                               currentPageReportTemplate="Pág. {currentPage} de {totalPages}">
100                     <f:facet name="header">
101                         <h:outputText value="Total de Archivos Cargados: #{proveBean.nomArchivos.size()}"/>
102                     </f:facet>
103                     <h:outputText value="#{indice}.- #{ nombre}"/>
104                 </p:dataList>
105             </p:outputPanel>
106
107         </p:tab>
```

Codigo 4. Archivos en WSDL.

En la pestaña de notificaciones se nos presenta una lista de coincidencias encontradas despues de que el cliente las ha recibido y este haya deseado contactarlo Codigo 5.



```
<p:tab id="tab_Noti" title="Notificaciones">
    <h:form id="form2">
        <p:poll interval="30"
            listener="#{proveBean.notificacionesNovistas}" update=":tab_View:tableNot,:growl" />
    </h:form>
    <p:panel header="Notificaciones encontradas">
        <p:dataTable id="tableNot" var="notif" value="#{proveBean.notificaciones}">
            <p:column headerText="Id_Notificación" style="width:3%">
                <h:outputText value="#{notif.idNotificacion}"/>
            </p:column>
            <p:column headerText="Id_Servicio" style="width:3%">
                <h:outputText value="#{notif.servicio.idServicio}"/>
            </p:column>
            <p:column headerText="Nombre cliente" style="width:3%">
                <h:outputText value="#{notif.solicitud.usuario.nombre}"/>
            </p:column>
            <p:column headerText="E-mail" style="width:20%">
                <h:outputText value="#{notif.solicitud.usuario.email}"/>
            </p:column>
            <p:column headerText="Telefono" style="width:10%">
                <h:outputText value="#{notif.solicitud.usuario.telefono}"/>
            </p:column>
             <p:column headerText="Visto" style="width:5%">
                <h:form id="form3">
                    <p:commandButton id="visto" icon="#{proveBean.stado(notif.estado)}"
                                     update=":tab_View:tableNot,:growl,:tab_View:confirmDialogNotificacion"
                                     oncomplete="dialogoNotificacionVista.show()" title="Visto">
                     <f:setPropertyActionListener value="#{notif}" target="#{proveBean.selectednotificacion}"/>
                    </p:commandButton>
                    </h:form>
                </p:column>
            </p:dataTable>

             <p:confirmDialog  id="confirmDialogNotificacion"
                               message="Marcar como visto la notificacion con id #{proveBean.selectednotificacion.idNotifi
                               header="Marcar como visto  "
                               severity="alert"
                               widgetVar="dialogoNotificacionVista"
                               showEffect="fade"
                               hideEffect="explode"
                               appendToBody="true"
                                >
                <h:form id="form7">

                    <p:commandButton id="Confirm" value="Aceptar" update=":tab_View:tableNot,:growl" oncomplete="dialogoN
                                     icon="ui-icon-check" actionListener="#{proveBean.marcarNoticicacionVista()}" />
                    <p:commandButton id="Decline" value="Cancelar" type="button" icon="ui-icon-close" onclick="dialogoNot
                </h:form>
            </p:confirmDialog>


    </p:panel>
</p:tab>
```

Código 5. Notificaciones al proveedor.

6.2.4 . Sesión del Usuario Cliente.

En la sesión del cliente te muestra una tabla con 4 apartados que son Datos, Nuevo, Solicitudes y Notificaciones.

En la Figura 11. Nos muestra el apartado Datos en ella estan visibles los datos que se proporcionaron en el registro. El código es el mismo que en el proveedor.

Figura 11. Datos del cliente.

En la Figura 12.. Se visualiza el apartado Nuevo en el cual el cliente puede registrar una nueva solicitud.

Figura 12. Nueva Solicitud.

Para este apartado de registro de nueva solicitud el codigo es similar al del registro de cliente, ver Código 6.

```
<p:tab id="tabNewReg" title="Nuevo">
    <h:form id="formSer" style="margin-left: 23%; margin-top: 1%;width: 51%">
        <p:panel header="Registra una nueva solicitud">
            <h:panelGrid cellpadding="8">
                <p:outputLabel value="Nombre del servicio:" />
                <p:inputText id="NomSer" size="45" required="true" value="#{clienteBean.selectedsolicitud.nombreSer}"
                             label="Servicio"/>
                <p:outputLabel value="Nombre de operacion:" />
                <p:inputText id="NomOper" size="45" required="true" value="#{clienteBean.selectedsolicitud.nombreOper}"
                             label="Operacion"/>
                <p:outputLabel value="Nombre de parametro de entrada:" />
                <p:inputText id="NomParIn" size="45" value="#{clienteBean.selectedsolicitud.nombreParamIn}"
                             label="Parametro In"/>
                <p:outputLabel value="Nombre de parametro de Salida:" />
                <p:inputText id="NomParOut" size="45" value="#{clienteBean.selectedsolicitud.nombreParamOut}"
                             label="Parametro Out"/>

                <f:facet name="footer">
                    <p:separator/>
                    <p:commandButton id="ButtonGuardar" value="Guardar" actionListener="#{clienteBean.btncreateSolicitud()}
                                     update=":tabview:tab_solicitudes_user,:growl,:tabview:formSer" icon="ui-icon-disk" />
                    <p:commandButton id="ButtonCancelar" value="Cancelar" icon="ui-icon-close" actionListener="#"/>
                </f:facet>
                <p:growl id="messages" showDetail="true"/>
            </h:panelGrid>
        </p:panel>
    </h:form>
</p:tab>
```

Código 6. Registro de solicitud.

La Figura 13. Nos presenta las solicitudes realizadas anteriormente.

Figura 13. Historial de solicitudes.

Por ultimo en la Figura 14. Se nos muestran las notificaciones, es decir si existe alguna coincidencia.

Figura 14. Notificaciones del cliente.

6.3. Capa del motor de búsquedas.

El motor de búsquedas de coincidencia comienza ha hacer la busqueda en las solicitudes ya que esta cuenta con un valor cuando esta en 0 es porque no se ha encontrado coincidencia para esa solicitud y en 1 cuando ya fue atendida, es por ello que el motor realiza las consultas de acuerdo al valor en 0(Ver Codigo 7), posteriormente realiza las búsquedas del nombre y de la operación(Ver Código 8) si encuentra alguna coincidencia este envia las notificaciones a los respectivos usuarios

```
22      public List<Solicitud> allSolicitudNoAtendidasDelCliente(int id) {
23          List<Solicitud> listSol = null;
24          Session sesion = HibernateUtil.getSessionFactory().getCurrentSession();
25          try {
26              sesion.beginTransaction();
27              String sql = "FROM Solicitud  WHERE Valor = '0' and Usuario_id_Usuario = '" + id + "'";
28              listSol = sesion.createQuery(sql).list();
29
30              //sesion.beginTransaction().commit();
31          } catch (Exception e) {
32              sesion.beginTransaction().rollback();
33          }
34          return listSol;
35      }
36
37      public List<Servicio> obtenerServicio(String m) {
38          List<Servicio> ser = null;
39          Session sesion = HibernateUtil.getSessionFactory().getCurrentSession();
40          try {
41              sesion.beginTransaction();
42              String sql = "from Servicio where NombreSer like '%" + m + "%'";
43              ser = sesion.createQuery(sql).list();
44
45              //sesion.beginTransaction().commit();
46          } catch (Exception e) {
47              sesion.beginTransaction().rollback();
48          }
49          return ser;
50
51      }
```

Codigo 7. Código de las busquedas SQL.

```
53      public List<Servicio> obtenerServicioDesdeOperacion(String m) {
54          List<Operacion> op = null;
55          List<Servicio> ser = null;
56          Session sesion = HibernateUtil.getSessionFactory().getCurrentSession();
57          try {
58              sesion.beginTransaction();
59              ser = new ArrayList<Servicio>(0);
60              String sql = "from Operacion where NombreOpe like '%" + m + "%'";
61              op = sesion.createQuery(sql).list();
62
63              for (Operacion opera : op) {
64                  // System.out.println("Nombre de la operacion: "+opera.getNombreOpe());
65
66                  // System.out.println("operacion pertenece al/los servicios: ");
67                  Set<Servicio> servicios = opera.getServicios();
68                  for (Servicio servicio : servicios) {
69                      //System.out.println("nombre del servicio: "+servicio.getNombreSer());
70                      ser.add(servicio);
71                  }
72              }
73
74              Set<Servicio> set = new HashSet<Servicio>(ser);
75              ser.clear();
76              ser = new ArrayList<Servicio>(set);
77
78
79              //sesion.beginTransaction().commit();
80          } catch (Exception e) {
81              sesion.beginTransaction().rollback();
82          }
83          return ser;
84      }
```

Codigo 8. Obtener búsqueda de operaciones.

Al igual realiza la busqueda de coincidencias con los parametros de entrada se muestra el código 9 y de salida en el código 10.

```
86      public List<Servicio> obtenerServicioDesdeParamin(String m) {
87          List<Paramin> pi = null;
88          List<Servicio> ser = null;
89          Session sesion = HibernateUtil.getSessionFactory().getCurrentSession();
90          try {
91              ser = new ArrayList<Servicio>(0);
92              sesion.beginTransaction();
93              String sql = "FROM Paramin  WHERE NombreParamIn like '%" + m + "%'";
94              pi = sesion.createQuery(sql).list();
95
96              Set<Paramin> set = new HashSet<Paramin>(pi);
97              for (Paramin paramin : set) {
98                  Set<Servicio> servicios = paramin.getOperacion().getServicios();
99                  for (Servicio servicio : servicios) {
100                     ser.add(servicio);
101                 }
102             }
103             Set<Servicio> serv = new HashSet<Servicio>(ser);
104             ser.clear();
105             ser = new ArrayList<Servicio>(serv);
106             //sesion.beginTransaction().commit();
107         } catch (Exception e) {
108             sesion.beginTransaction().rollback();
109         }
110         return ser;
111
112     }
113
```

Código 9. Busquedas de coincidencia en parametros In

```
114     public List<Servicio> obtenerServicioDesdeParamout(String m) {
            List<Paramout> po = null;
116         List<Servicio> ser = null;
117         Session sesion = HibernateUtil.getSessionFactory().getCurrentSession();
118         try {
119             ser = new ArrayList<Servicio>(0);
120             sesion.beginTransaction();
121             String sql = "FROM Paramout  WHERE NombreParamOut like '%" + m + "%'";
122             po = sesion.createQuery(sql).list();
123
124             Set<Paramout> set = new HashSet<Paramout>(po);
125             for (Paramout paramout : set) {
126                 Set<Servicio> servicios = paramout.getOperacion().getServicios();
127                 for (Servicio servicio : servicios) {
128                     ser.add(servicio);
129                 }
130             }
131             Set<Servicio> serv = new HashSet<Servicio>(ser);
132             ser.clear();
133             ser = new ArrayList<Servicio>(serv);
134             //sesion.beginTransaction().commit();
135         } catch (Exception e) {
136             sesion.beginTransaction().rollback();
137         }
138         return ser;
139
140     }
```

Código 10. Busquedas de coincidencia en parametros Out

6.4. Capa de envio de Notificaciones.

En esta capa las coincidencias encontradas son enviadas al cliente, modificando el valor antes mencionado de 0 sin coincidencia y 1 con coincidencia. Esto se puede visualizar en el siguiente Código 11.

```
142     public Notificacion agregarNotificacion(Notificacion notificacion) {
143         Session sesion = HibernateUtil.getSessionFactory().getCurrentSession();
144         try {
145             sesion.beginTransaction();
146             sesion.save(notificacion);
147             sesion.beginTransaction().commit();
148
149         } catch (Exception e) {
150
151             sesion.beginTransaction().rollback();
152         }
153         return notificacion;
154     }
155
156
157     public Solicitud modificarSolicitud(Solicitud solicitud) {
158         Session sesion = HibernateUtil.getSessionFactory().getCurrentSession();
159         try {
160             sesion.beginTransaction();
161             sesion.update(solicitud);
162             sesion.beginTransaction().commit();
163
164         } catch (Exception e) {
165
166             sesion.beginTransaction().rollback();
167         }
168         return solicitud;
169     }
170 }
```

Código 11. Envio de notificaciones y modifica solicitud.

El envio de notificaciones es para el cliente en la siguiente clase se muestra(Codigo 12).

```java
public void crearNotificacionCliente(){
    System.out.println("\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\");
    System.out.println("Verificando las solicitudes y generando notificaciones");
     ConsultaMotor consulta = new ConsultaMotor();
    List<Solicitud> allSolicitudNoAtendidasDelCliente =
            consulta.allSolicitudNoAtendidasDelCliente(usuario.getIdUsuario());

    for (Solicitud solicitud : allSolicitudNoAtendidasDelCliente) {
        List<Servicio> obtenerServicio = consulta.obtenerServicio(solicitud.getNombreSer());
        List<Servicio> obtenerOperacion = consulta.obtenerServicioDesdeOperacion(solicitud.getNombreOper());
        List<Servicio> obtenerServicioDesdeParamin = consulta.obtenerServicioDesdeParamin(solicitud.getNombreParamIn());
        List<Servicio> obtenerServicioDesdeParamout = consulta.obtenerServicioDesdeParamout(solicitud.getNombreParamOut());

         Set<Servicio> encontrados = new HashSet<Servicio>(0);
         encontrados.addAll(obtenerServicio);
         encontrados.addAll(obtenerOperacion);
         encontrados.addAll(obtenerServicioDesdeParamin);
         encontrados.addAll(obtenerServicioDesdeParamout);

         if (!encontrados.isEmpty()) {
             //System.out.println("        Se obtuvieron estos servicios");
             solicitud.setValor(1);
             consulta.modificarSolicitud(solicitud) ;
             for (Servicio servicio : encontrados) {
                 /*
                  * se crea por cada servicio una notificacion
                  */
                 Notificacion notificacion = new Notificacion();
                 Set<Usuario> us = new HashSet<Usuario>();
                 us.add(usuario);
                 notificacion.setUsuarios(us);
                 notificacion.setEstado(0);
                 notificacion.setServicio(servicio);
                 notificacion.setSolicitud(solicitud);
                 notificacion.setPara(usuario.getIdUsuario());

                 if (consulta.agregarNotificacion(notificacion).getIdNotificacion() != null) {

                     System.out.println("****   Notificacion Agregada   ****");
                 }
             }
         }
     }
}
```

Código 12. Crear notificación.

6.5. Sesión Usuario Administrador.

Como se menciono anteriomente, el Administrador solo podra modificar y eliminar a los usuarios, si este deseara realizar alguna solicitud o ofrecer algun servicio, tendria que registrarse con el tipo de usuario deseado.

En el siguiente código 13 se muestra las forma de como podra modificar al usuario y las respectivas ventanas ya con el sistema ejecutado en la figura 15.

```
<p:dialog header="Modificar Usuario"
          widgetVar="dialogUsuarioUpdate"
          resizable="false"
          id="dlgUsuarioUpdate"
          showEffect="fade" hideEffect="explode" modal="true"
          appendToBody="true" >
    <h:form id="formUpdate">
        <h:panelGrid id="display" columns="2" cellpadding="5" style="margin: 0 auto">
            <p:selectOneMenu required="true" id="Tipo" label="Usuario" value="#{adminBean.selectedUsuario.rolIdRol}">
                <f:selectItem itemLabel="Tipo de Usuario" itemValue="" />
                <f:selectItem itemLabel="Cliente" itemValue="1"/>
                <f:selectItem itemLabel="Proveedor" itemValue="2"/>
                <f:selectItem itemLabel="Administrador" itemValue="3"/>
            </p:selectOneMenu>
            <p:outputLabel/>
            <p:outputLabel value="Nombre:" />
            <p:inputText size="45" value="#{adminBean.selectedUsuario.nombre}"/>
            <p:outputLabel value="Direccion:" />
            <p:inputText size="65" value="#{adminBean.selectedUsuario.direccion}"/>
            <p:outputLabel value="E-mail:" />
            <p:inputText  id="mail" value="#{adminBean.selectedUsuario.email}" size="35" required="false" label="Email"
                          validatorMessage="Email: Error de formato" >
                <f:validateRegex
                    pattern="^[_A-Za-z0-9-\+]+(\.[_A-Za-z0-9-]+)*@[A-Za-z0-9-]+(\.[A-Za-z0-9]+)*(\.[A-Za-z]{2,})$" />
            </p:inputText>
            <p:outputLabel value="Celular" />
            <p:inputMask mask="(99) 9999-9999" size="15" value="#{adminBean.selectedUsuario.telefono}"/>
            <p:outputLabel value="Contraseña" />
            <p:inputText value="#{adminBean.selectedUsuario.password}"/>
            <f:facet name="footer">
                <p:separator/>
                <p:commandButton id="btnAceptar" icon="ui-icon-check" value="Aceptar" update=":users, :msg"
                                 oncomplete="dialogUsuarioUpdate.hide()" title="Guardar" actionListener="#{adminBean.btnU
                <p:commandButton id="btnCancelar" icon="ui-icon-circle-close" value="Cancelar"
                                 oncomplete="dialogUsuarioUpdate.hide()" title="Cancelar"/>
            </f:facet>
        </h:panelGrid>
    </h:form>
</p:dialog>
```

Código 13. Modificar usuario.

Figura 15. Ventana Modificar.

Tambien en el código 14 muestra como elimina y la ventana que muestra con el sistema en ejecución figura 16.

```
<h:form id="formDelete">
    <h:inputHidden value="#{adminBean.selectedUsuario.idUsuario}"/>
    <p:commandButton id="Confirm" value="Aceptar" update=":users,:msg" oncomplete="dialogUsuarioDelete.hide()"
                     icon="ui-icon-check" actionListener="#{adminBean.btnDeleteUsuario()}" />
    <p:commandButton id="Decline" value="Cancelar" type="button" icon="ui-icon-close"
                     onclick="dialogUsuarioDelete.hide()"/>
</h:form>
```

Código 14. Eliminar Usuario.

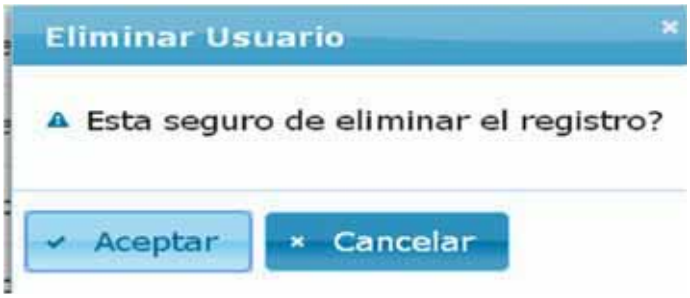


Figura 16. Confirmar eliminar.

7 . Pruebas y resultados.

Para realizar las pruebas al sistema, se creó un usuario Cliente, uno Proveedor y el respectivo administrador.

7.1. Administrador.

El administrador registrado es: **Usuario:** Admin.
Contraseña: Admin.

Realizamos el login Figura 17

Figura 17. Login Administrador.

En la figura 18 nos muestra la sesión del Administrador y en la ultima columna se encuentran los botones que permiten modificar o eliminar.

Id	Nombre	Contraseña	Dirección	E-mail	Telefono	Rol
1	admin	admin	a	a	5512457896	1
2	cliente	cliente	a	cliente@cliente.com	(55) 1111-2222	3
3	proveedor	proveedor	a	proveedor@proveedor.com	(55) 2222-1111	2
4	proveedor2	proveedor2	a	proveedor2@proveedor.com	(55) 2222-2222	2
5	cliente2	cliente2	calle	algo@algo.com	(44) 7777-7777	3
6	cliente3	cliente3	cliente3	cliente@cliente.com	(88) 8888-8888	3
7	yara	yara	av5 de mayo	yaren_lizdith@hotmail.com	(55) 2145-2307	3
8	lizbeth	lizbeth	av 5 de mayo	yaraliz1990@gmail.com	(55) 2145-2307	2

Figura 18. Sesión de Administrador

7.2. Cliente.

El Cliente registrado es:

Usuario: yara.
Contraseña: yara.

Realizamos el login Figura 19:

Figura 19. Login Cliente

La sesión del cliente logueado en la Figura 20.

Figura 20. Sesión del Cliente.

Realizamos una nueva solicitud los datos que se muestan en la Figura 21. estos datos son veridicos, es decir, si existe un servicio. El nombre de los parametros de entrada y de salida no son necesarios.

Figura 21. registro de una nueva solicitud.

En la Figura 22 nos muestra los servicios ya realizados.

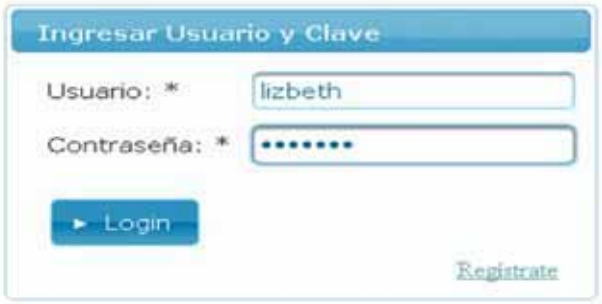
Figura 22. Muestra el historial del cliente.

7.3. Proveedor.

El proveedor registrado es:

Usuario: lizbeth
Contraseña: lizbeth

Realizamos el login Figura23 :



En la Figura 24 nos muestra la sesión del proveedor.

Realizaremos un registro de un nuevo servicio. En la pestaña Nuevo, damos clic en el boton Buscar seleccionamos el o los archivos en WSDL que deamos ofertar Figura 25, posteriormente damos clic en subir Figura 26, luego guardar cuando te muestre el mensaje de realizado correctamente (Figura 27) das clic en terminar.

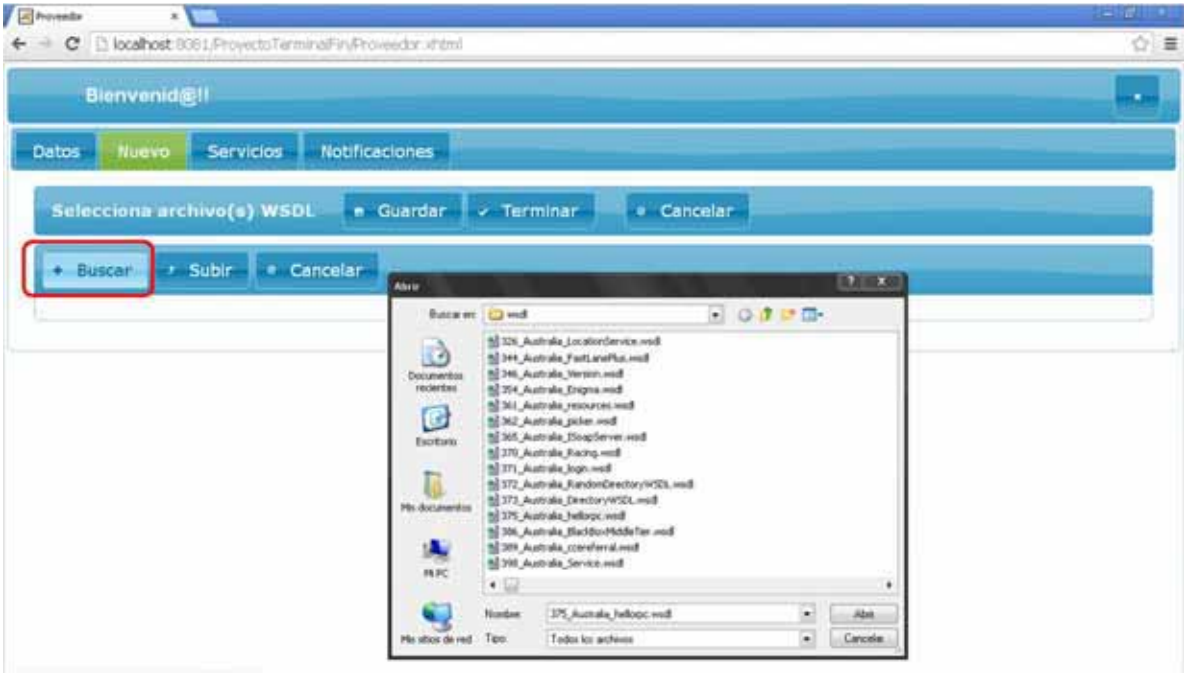


Figura 25. Seleccionar archivos.

Figura 26. Subir Archivos.

Figura 27. Proceso concluido

Al finalizar, al dar clic a la pestaña Servicios nos muestra el historial del servidor esto se visualiza en la figura 28.

Figura 28. Historial del Proveedor.

7.4. Notificaciones

Dentro de la prueba que estamos realizando. Al cliente ya le llego la notificacion correspondiente a la solicitud realizada, la podemos ver en la figura 29.
A la cual le podemo activar e visto y de igual manera enviarle un correo al proveedor para contactarlo como se muestra en la figura 30.

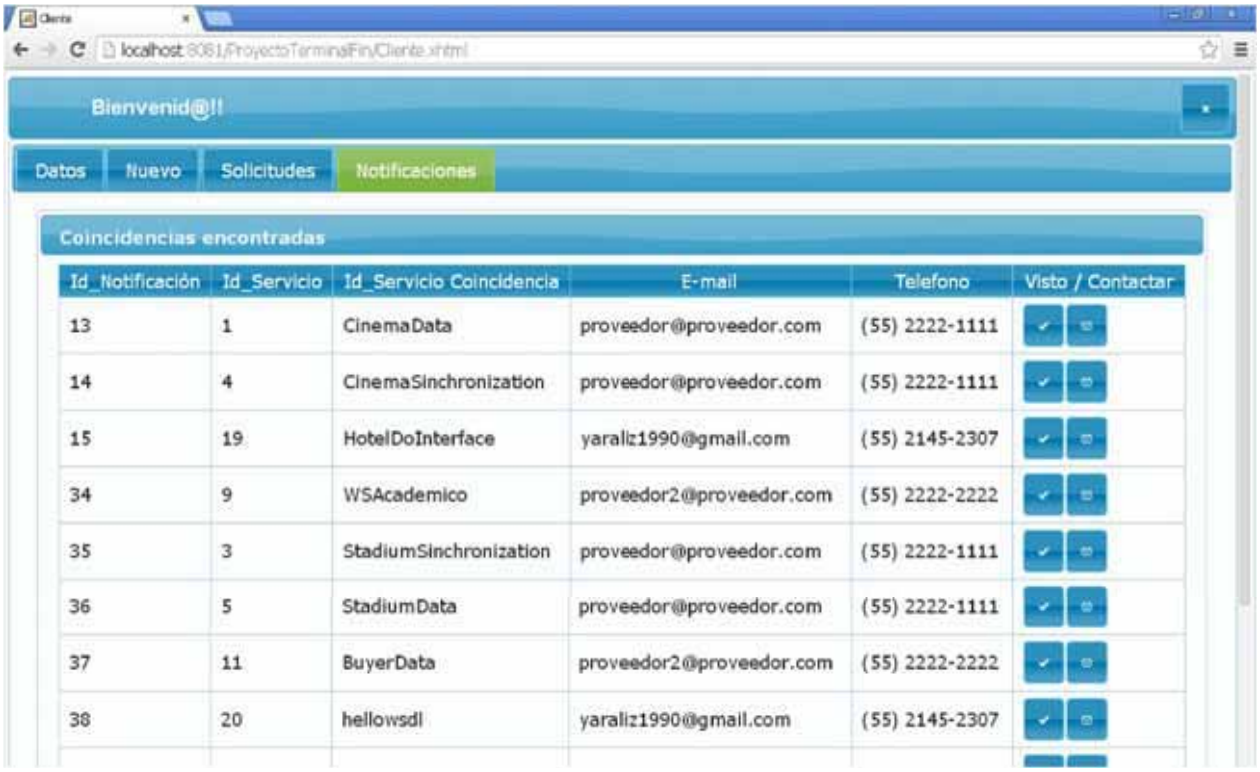


Figura 29. Notificaciones llegadas al cliente.

Se marca la notificacion como visto y se le puede enviar un correo esto es ver Figura 30 y 31. En la Figura 31 se puede dar cuenta que se tiene que dar clic al boton enviar e-mail posteriormete en aceptar.

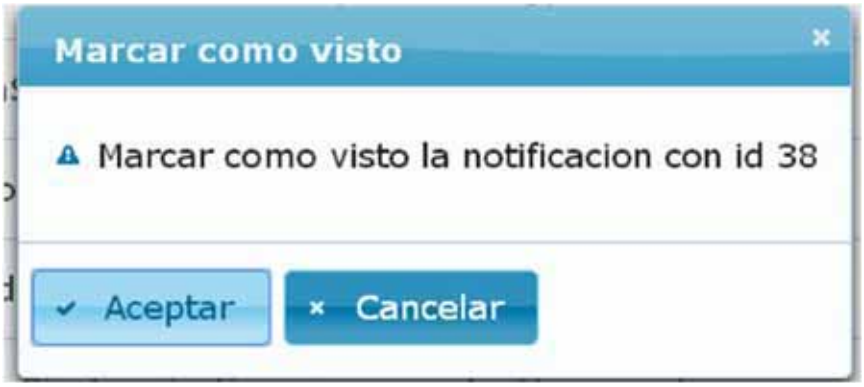


Figura 30. Aceptar para marcar como visto.

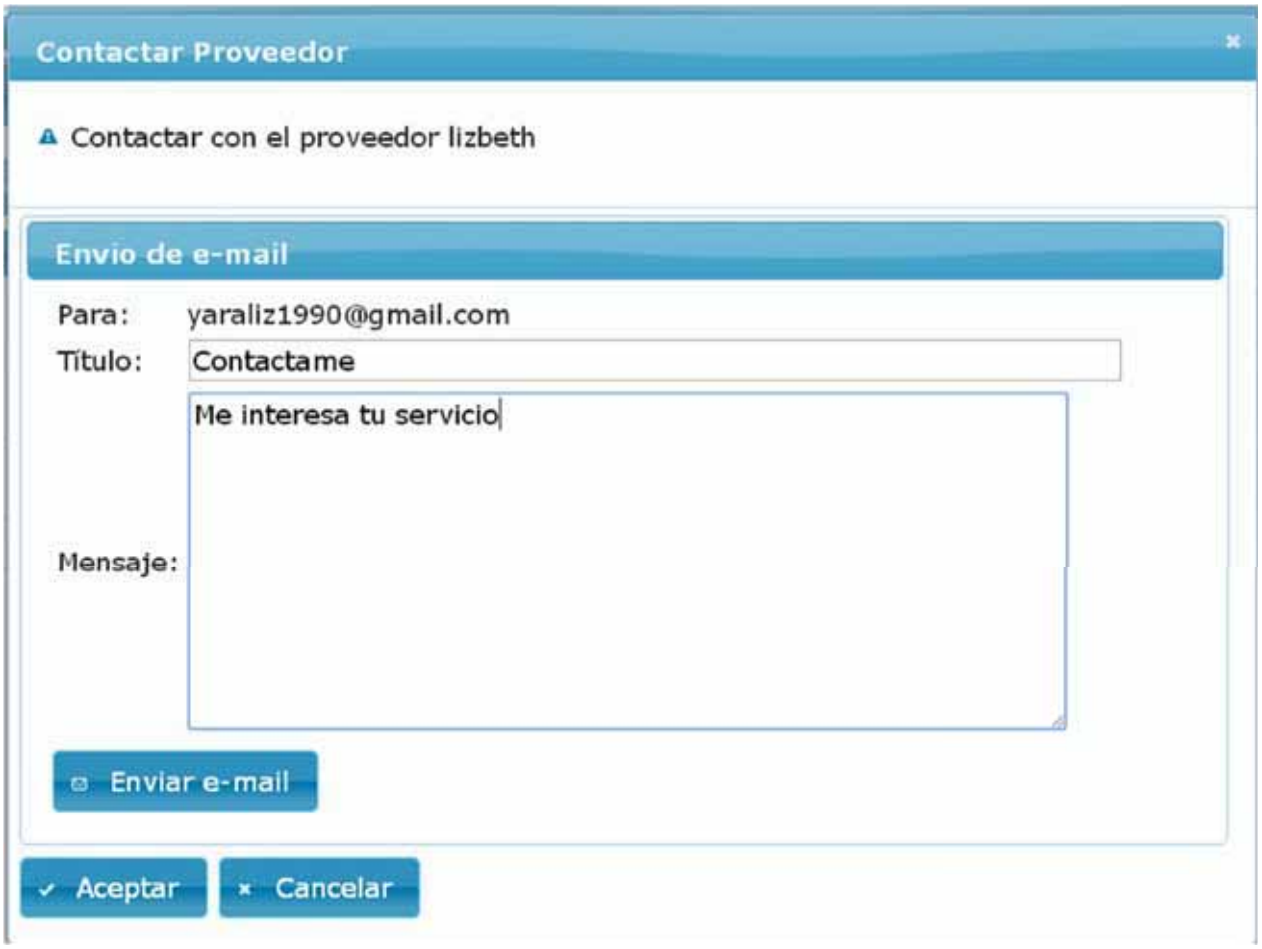


Figura 31. Se envia el correo para contactar al proveedor.

Por ultimo, como la contactaciçon entre ellos es externo al sistema, el correo que se le hace llegar al proveedor es como este: ver figura 32.

Figura 32. E-mail recibido por el proveedor.

8 . Conclusión.

El objetivo de este proyecto terminal fue crear un sistema web, mediante el cual clientes que pueden ser pesonas que no cuenten con los conocimientos acerca de un servicio en WSDL puedan solicitarlo, con solo poner el nombre del servicio que se desea y alguna operación que contenga, y por el otro lado que se encuentre algun experto ó conocedor del tema que ofrezca ese servicio y puedan tener contacto para satisfacer sus requerimientos.

El experto es el que provee los servicios en WSDL, estos archivos son Parseados por una parte del sistema en el cual se extraen el nombre del servicio, las operaciones contenidas, así como sus parametros de entrada y salida.

En ese momento el motor entra en accion comparando lo que se extrae del WSDL con los condiciones que el cliente pide. Las notificaciones se hacen presentes cuando al menos el nombre y alguna operación coincidan, es por eso que para el cliente no es obligatorio dar los nombres de parametros de entrada y salida.

El motor de busqueda implementado realiza esta búsquedas automaticamente y se las envia al cliente , el cual decide si el contactar al proveedor. De esta manera si le ha llegado alguna notificación y no cubrio con todas las espectativas este puede seguir buscando de hasta encontrar el deseado.

El motor de busquedas comienza comparando lo que el cliente requiere esto es para desechar solicitudes ya atendidas, con esto ahorramos tiempo en la búsqueda de coincidencias.

9 . Trabajo a futuro.

Este proyecto tiene muchas ramas por donde crecer, por un lado el sistema estaba pensado para enviar notificaciones por medio de mensajes SMS, debido a la economía esto no se pudo llevar a cabo .

También, puede ser mejorado para que sea una aplicación movil, es decir, migrar a *Android*, puede ser también para *Apple-IOS* o *BlackBerry OS* y con el uso de nuevas tecnologías; así las notificaciones le llegarían en tiempo.

Otra mejoría es que se agregue la contactación dentro del sistema y ya no fuera externo asi se le daría seguimiento a las coincidencias encontradas por el motor.

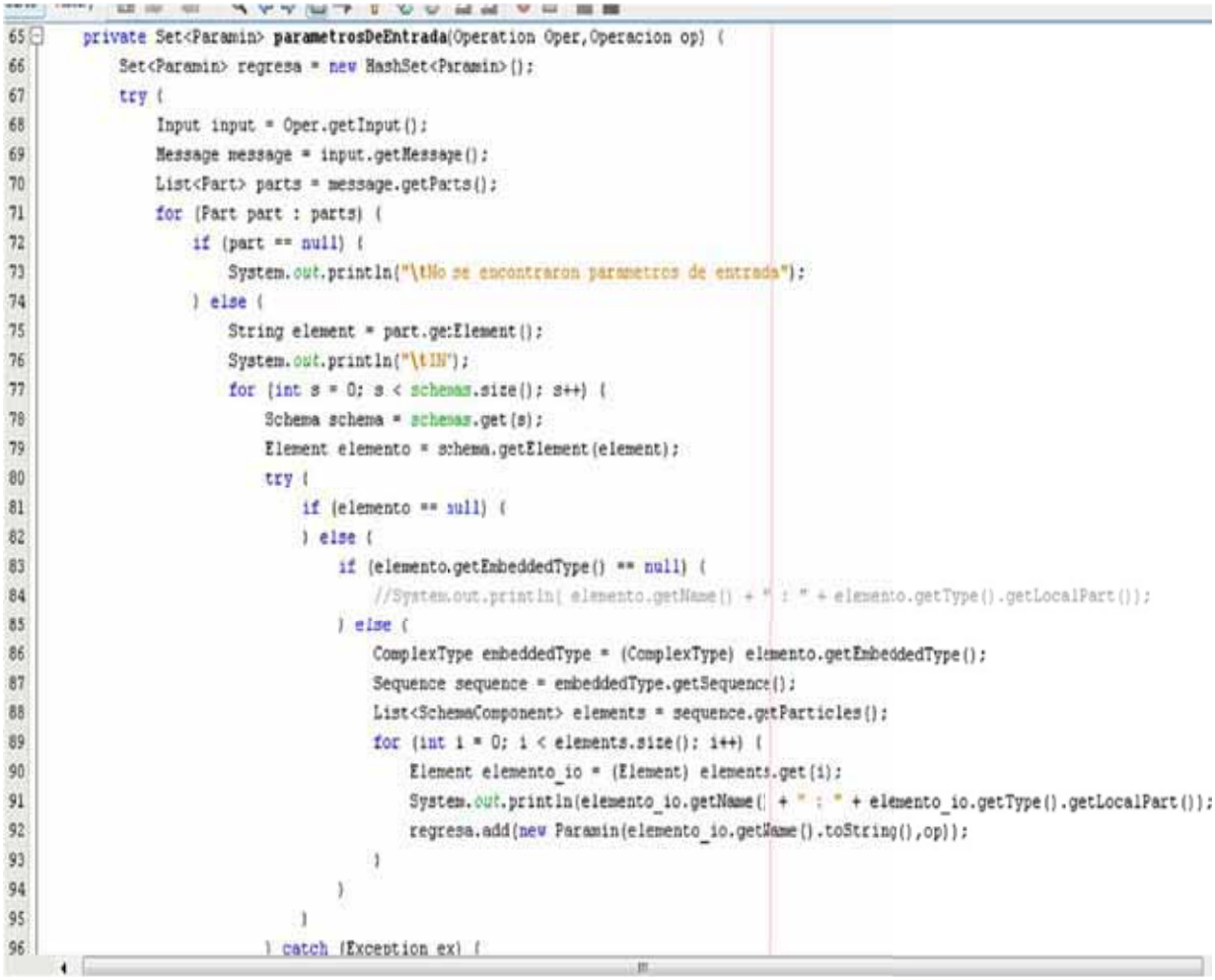
10 . Anexo

```
22  public class ConsultaWsdl {
23
24
25      public ConsultaWsdl() {
26      }
27
28      public Servicio CreateServicio(Servicio servicio) {
29
30          Session sesion = HibernateUtil.getSessionFactory().getCurrentSession();
31          try {
32              sesion.beginTransaction();
33              sesion.save(servicio);
34              Set<Operacion> operaciones = servicio.getOperaciones();
35              for (Operacion operacion : operaciones) {
36                  Set<Paramin> paramin = operacion.getParamin();
37                  for (Paramin paramin1 : paramin) {
38                      sesion.save(paramin1);
39                  }
40                  Set<Paramout> paramout = operacion.getParamout();
41                  for (Paramout paramout1 : paramout) {
42                      sesion.save(paramout1);
43                  }
44              }
45
46              sesion.beginTransaction().commit();
47          } catch ( JDBCException e) {
48              sesion.beginTransaction().rollback();
49              servicio.setError( e.getSQLException().getMessage());
50          }
51          return servicio;
52      }
53
```

Código 15. Clase Consulta WSDL

```
28      public void ParseWsdl(String target) {
29          servicio = new Servicio();
30
31          WSDLParser parser = new WSDLParser();
32          defs = parser.parse(target);
33          this.schemas = defs.getSchemas();
34          List<Service> services = defs.getServices();
35
36          for (Service service : services) {
37              Service ServTarget = service;
38              String name = ServTarget.getName();
39              System.out.println("Archivo: " + target);
40              System.out.println("Nombre del servicio: " + name);
41              String namespace = defs.getTargetNamespace();
42              System.out.println("Uri: " + namespace);
43              servicio.setNombreSer(name);
44
45              Set<Operacion> operaciones = new HashSet<Operacion>();
46              for (Operation op : defs.getOperations()) {
47                  Operacion operacion = new Operacion();
48                  operacion.setNombreOpe(op.getName());
49                  System.out.println(" OPERACION -" + op.getName());
50                  int total = 0;
51                  Set<Paramin> listain = parametrosDeEntrada(op,operacion);
52                  Set<Paramout> listaout = ParametrosDeSalida(op,operacion);
53                  operacion.setParamin(listain);
54                  operacion.setParamout(listaout);
55
56                  //VerEntradasSalidas(op);
57                  total++;
58                  operaciones.add(operacion);
59              }
```

Código 16. Parser WSDL



```
65  private Set<Paramin> parametrosDeEntrada(Operation Oper,Operacion op) {
66      Set<Paramin> regresa = new HashSet<Paramin>();
67      try {
68          Input input = Oper.getInput();
69          Message message = input.getMessage();
70          List<Part> parts = message.getParts();
71          for (Part part : parts) {
72              if (part == null) {
73                  System.out.println("\tNo se encontraron parametros de entrada");
74              } else {
75                  String element = part.getElement();
76                  System.out.println("\tIN");
77                  for (int s = 0; s < schemas.size(); s++) {
78                      Schema schema = schemas.get(s);
79                      Element elemento = schema.getElement(element);
80                      try {
81                          if (elemento == null) {
82                          } else {
83                              if (elemento.getEmbeddedType() == null) {
84                                  //System.out.println( elemento.getName() + " : " + elemento.getType().getLocalPart());
85                              } else {
86                                  ComplexType embeddedType = (ComplexType) elemento.getEmbeddedType();
87                                  Sequence sequence = embeddedType.getSequence();
88                                  List<SchemaComponent> elements = sequence.getParticles();
89                                  for (int i = 0; i < elements.size(); i++) {
90                                      Element elemento_io = (Element) elements.get(i);
91                                      System.out.println(elemento_io.getName() + " : " + elemento_io.getType().getLocalPart());
92                                      regresa.add(new Paramin(elemento_io.getName().toString(),op));
93                                  }
94                              }
95                          }
96                      } catch (Exception ex) {
```

Código 17. Clase para obtener los parametros de entrada.

```
106    private Set<Paramout> ParametrosDeSalida(Operation Oper,Operacion op) {
107        Set<Paramout> regresa = new HashSet<Paramout>();
108        try {
109            Output output = Oper.getOutput();
110            Message message1 = output.getMessage();
111            List<Part> parts1 = message1.getParts();
112
113            for (Part part : parts1) {
114                if (part == null) {
115                    System.out.println("\tNo se encontraron parametros de salida");
116                } else {
117                    System.out.println("\tOUT");
118                    String element = part.getElement();
119                    for (int s = 0; s < schemas.size(); s++) {
120                        Schema schema = schemas.get(s);
121                        Element elemento = schema.getElement(element);
122                        try {
123                            if (elemento == null) {
124                            } else {
125                                if (elemento.getEmbeddedType() == null) {
126                                } else {
127                                    ComplexType embeddedType = (ComplexType) elemento.getEmbeddedType();
128                                    Sequence sequence = embeddedType.getSequence();
129                                    List<SchemaComponent> elements = sequence.getParticles();
130                                    for (int i = 0; i < elements.size(); i++) {
131                                        Element elemento_io = (Element) elements.get(i);
132                                        System.out.println(elemento_io.getName() + " : " + elemento_io.getType().getLocalPart());
133                                        regresa.add(new Paramout(elemento_io.getName().toString(),op));
134                                    }
135                                }
136                            }
137                        } catch (Exception ex) {
```

Código 18. Clase para obtener los parametros de salida.

```
30      */
31      public PoblarWsdlIX(Set<File> coleccion) {
32          this.coleccion = coleccion;
33      }
34
35      public List<error> poblarBaseDatos(int usuarioIdUsuario) {
36          String ok="";
37          int index = 0;
38          wsdl = new ParserWSDL();
39          for (File file : coleccion) {
40              index++;
41              System.out.println("\n\n#######################################");
42              System.out.println("Archivo : " + index);
43              wsdl.init(file.getPath());
44              Servicio servicio = wsdl.getServicio();
45              servicio.setUsuario(new ConsultaUsuario().obtenerUsuarioId(usuarioIdUsuario));
46             //servicio.setUsuarioIdUsuario(usuarioIdUsuario);
47              ConsultaWsdl consulta = new ConsultaWsdl();
48              Servicio CreateServicio = consulta.CreateServicio(servicio);
49              if (CreateServicio.getError() != null) {
50                   errores.add(new error(CreateServicio.getError() + " in "+ file.getName()));
51              }
52          }
53          return this.errores;
54      }
55
56      public void init(List<error> errores) {
57         this.errores = errores;
58      }
```

Código 19. Clase de Poblado de la Base de datos.

11 . Bibliografía

[1] J. Pascual Martínez. "Extracción automatizada y representación de servicios Web mediante ontologías", Proyecto terminal, Universidad Autónoma Metropolitana Azcapotzalco, D.F., México, 2011.

[2] E. Estrada Sánchez. "Clasificación de servicios web semánticos mediante ontologías", propuesta de proyecto terminal, Universidad Autónoma Metropolitana Azcapotzalco, D.F., México, 2012.

[3] D.A. Malagón Mercado "Solución de problemas de mediana dificultad utilizando composición de servicios web", propuesta de proyecto terminal, Universidad Autónoma Metropolitana Azcapotzalco, D.F., México, 2011.

[4] S. Agarwal, and R. Studer, "Automatic Matchmaking of Web Services," Proc. of the IEEE 4th International Conference on Web Services, 2006, pp. 45-54.

[5] A. Brogi and and S. Corfini and R. Popesc, "Flexible Matchmaking of Web Services," Using DAML-S Ontologies. Proceedings of the ICSOC; 2004, pp 30- 45.

[6] U. Bellur, "Semantic Matchmaking Algorithms," Proceedings of the IEEE International Conference,2008, pp 120-128.

[7] IDE Netbeans, Netbeans 7.2.1. Disponible en https://www.netbeans.org/

[8] PhpMyAdmin, PhpMyAdmin 4.0.3. Disponible en https://www.phpmyadmin.net/

[9] JavaMail API, JavaMail API. Disponible en http://www.oracle.com/technetwork/java/javamail/

[10] PrimeFaces, PrimeFaces 3.5. Disponible en https://primefaces.org/

[11] Apache Tomcat, Apache Tomcat 2013. Disponible en https://tomcat.apache.org/

[12] Apache Axis2/ Java, Apache Axis2/Java. Dispobible en https://axis.apache.org/axis2/java/core/

Universidad Autónoma Metropolitana Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería

Licenciatura en Ingeniería en Computación

Manual de usuario del proyecto:
Sistema de búsqueda de coincidencias de servicios web

Yara Lizbeth De La Cruz Hernández

208331774

Asesor: Maricela Claudia Bravo Contreras.

Profesor Asociado

Departamento de Sistemas

Abril 2014

Indice

Manual de usuario.

1. Interfaz del sistema.

En la figura 1 nos muestra la interfaz inicial del sistema.

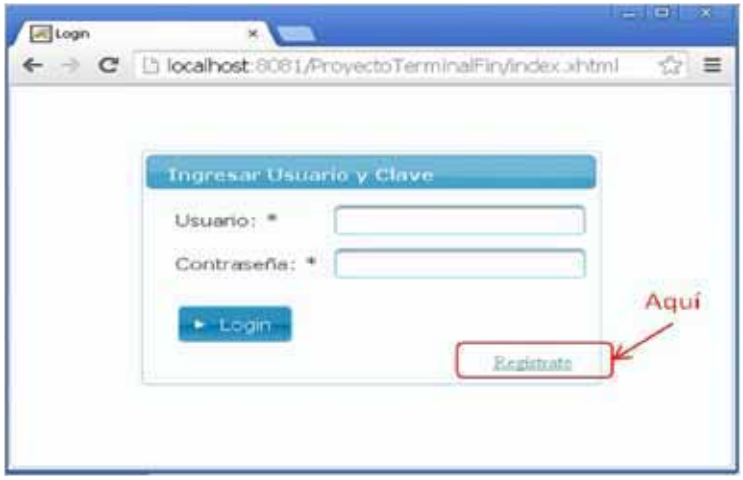


Figura 1.

2. Registro de usuario.

En ella das clic en Regístrate. Y te muestra la Figura 2.

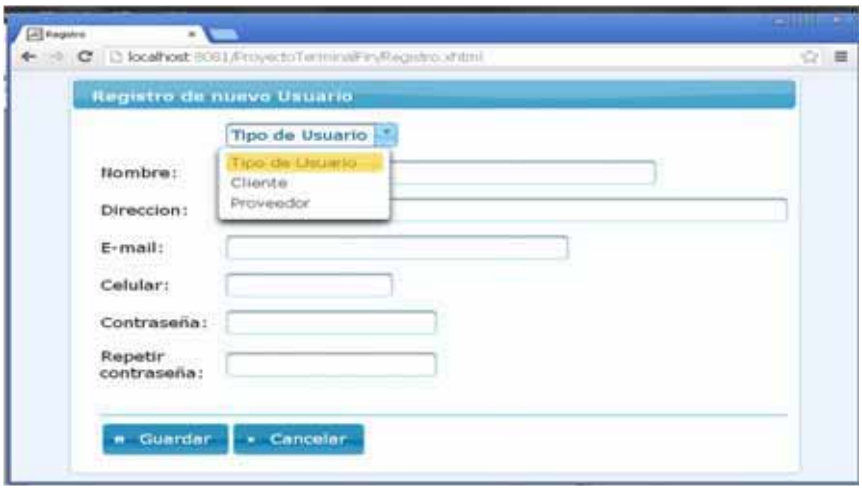


Figura 2.

En el cual realizar tu registro, das clic en guardar y se enlaza a login en el cual ya podras acceder sin complicaciones.

3. Sesión Cliente.

Si realizas tu registro como cliente, la Figura 3 te muestra como sera tu sesión.

Figura 3.

4. Registro de solicitud.

Como usuario cliente podrás solicitar algún servicio, en la pestaña Nuevo al finalizar das clic en guardar, se muestra la Figura 4.

Figura 4.

5. Historial del cliente.

Al guardar este se almacena en la pestaña solicitudes, ver Figura 5.

Figura 5.

6. Notificaciones del cliente

y en la pestaña notificaciones solo hay que esperar a que el sistema te devuelva alguna información, una vez echo esto puedes marcar como leído o enviar mensaje dando clic en los iconos correspondientes. Ver Figura 6 y 7.

Figura 6.

Figura 7.

7. Sesión Proveedor.

Si realizas tu registro como proveedor, la Figura 8 te muestra como sera tu sesión.

Figura 8.

8. Registro de servicio.

Como usuario proveedor podrás ofrecer uno o muchos servicios, en la pestaña Nuevo Seleccionas el archivo o archivos a publicar, das clic en subir, posteriormente a guardar y por ultimo terminar, Figura 9 muestra como esta diseñada.

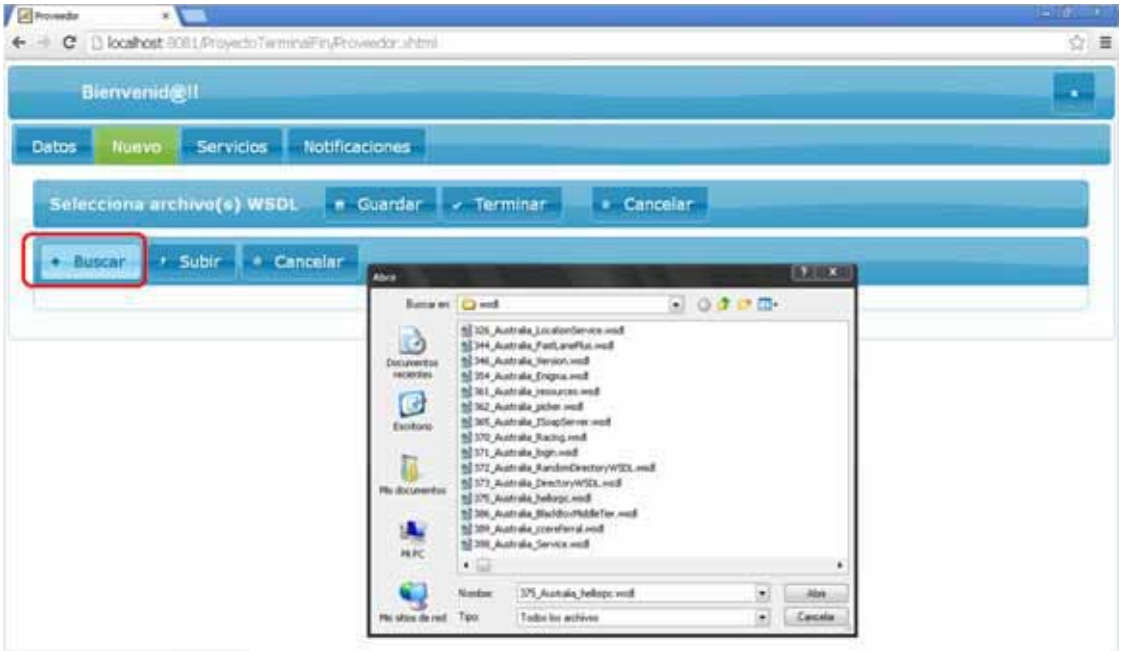


Figura 9.

9. Historial del proveedor.

Al guardar este se almacena en la pestaña solicitudes, ver Figura 10.

Figura 10.

10. Notificaciones del proveedor

y en la pestaña notificaciones solo hay que esperar a que el cliente te devuelva alguna información, una vez echo esto puedes marcar como leído dando clic en los iconos correspondientes. Ver Figura 11.

Figura 11.