ANEXO C WSMINING: TECNOLOGÍAS Y HERRAMIENTAS DE DASARROLLO

1 Tecnología para gestión de redes

1.1 Protocolo Simple de Gestión de Red - SNMP

El protocolo SNMP fue diseñado en los años 80, y su principal objetivo fue integrar la gestión de diferentes tipos de redes mediante un diseño sencillo y que produjera poca sobrecarga en la red.

SNMP opera en el nivel de aplicación, utilizando el protocolo de transporte TCP/IP[1], por lo que ignora los aspectos específicos del hardware sobre el que funciona. La gestión se lleva a cabo al nivel de IP[2], por lo que se pueden controlar dispositivos que estén conectados en cualquier red accesible desde la Internet, y no únicamente aquellos localizados en la red local. Evidentemente, si alguno de los dispositivos de enrutamiento con el dispositivo remoto a controlar no funciona correctamente, no será posible su monitoreo ni reconfiguración.

El modelo SNMP de una red gestionada consta de cuatro componentes, ver figura 1: Nodos, Estaciones, Información y Un protocolo.

Los nodos, pueden ser hosts, enrutadores, proxys, impresoras u otros dispositivos. Para ser gestionado directamente por el SNMP, un nodo debe ser capaz de ejecutar un proceso de gestión SNMP, llamado *agente SNMP*. Cada agente mantiene una base de datos local de variables que describen su estado e historia y que afectan su operación.

La gestión de la red se hace desde *estaciones gestoras*, que son ordenadores con un software de gestión especial. La estación gestora contiene uno o más procesos que se comunican con los agentes a través de la red, emitiendo comandos y recibiendo respuestas.

[1] Para mayor información sobre el protocolo TCP/IP ver http://www.saulo.net/pub/tcpip/
[2] Para mayor información sobre IP ver http://www.monografias.com/trabajos7/protoip/protoip.shtml

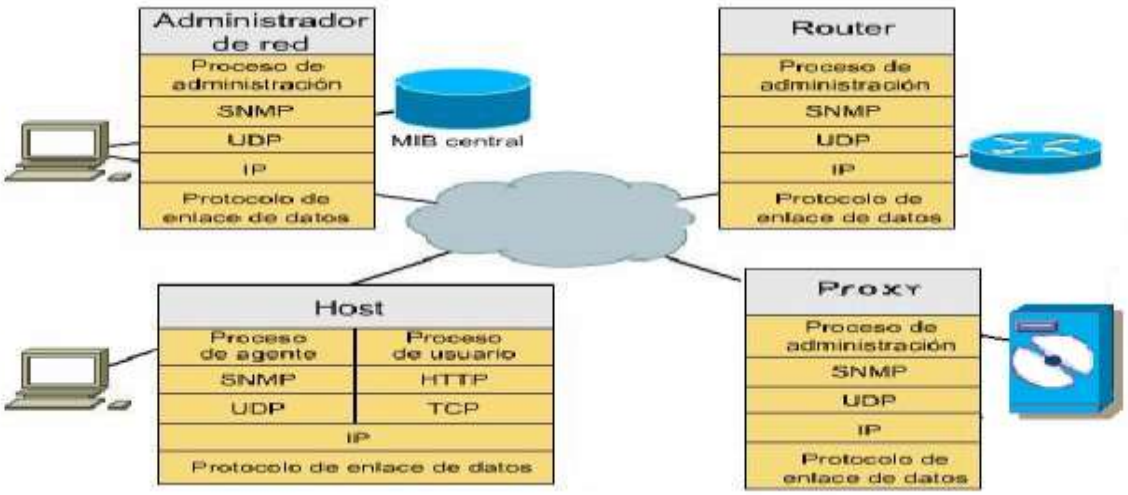

Figura 1Componentes de la Arquitectura SNMP

SNMP describe la información exacta de cada tipo de agente que tiene que gestionar la estación gestora y el formato con el que el agente tiene que proporcionarle los datos. Cada dispositivo mantiene una o más variables que describen su estado, estas variables se llaman *objetos*. El conjunto de todos los objetos posibles de una red, quedan almacenados en la estructura de datos llamada *MIB* (Management Information Base, Base de Información de Administración.

La estación gestora interactúa con los agentes usando SNMP. Este protocolo permite a la estación administradora consultar, y modificar, el estado de los objetos locales de un agente. La figura 2 muestra una estación gestora y varios agentes distribuidos en una red.

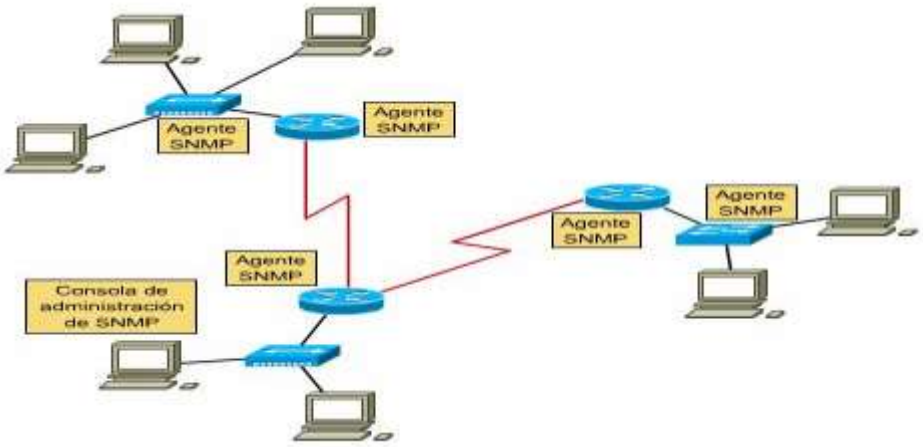

Figura 2 Agentes y estación administradora SNMP

En el ambiente SNMP, los errores, el congestionamiento, los fallos en la alimentación eléctrica o la caída de una línea son considerados como eventos significativos no planeados. Cada evento o suceso significativo se define en un módulo MIB. Cuando un agente nota que ha ocurrido un suceso significativo, de inmediato lo informa a todas las estaciones administradoras de su lista de configuración sin que hayan sido solicitados de forma explicita al gestor. Este tipo de informe es un comando especial de SNMP, llamado *trap*. Como la comunicación entre los nodos gestionados y la estación gestora no es confiable, la estación debe sondear ocasionalmente cada nodo.

En esencia, SNMP es un protocolo muy sencillo puesto que todas las operaciones se realizan bajo el paradigma de carga-y-almacenamiento (load-and-store), lo que permite un juego de comandos reducido. Un gestor puede realizar sólo dos tipos diferentes de operaciones sobre un agente: leer o escribir un valor de una variable en la MIB del agente. Estas dos operaciones se conocen como petición-de-lectura (get-request) y petición-de-escritura (set-request). Hay un comando para responder a una petición-de-lectura llamado respuesta-de-lectura (get-response), que es utilizado únicamente por el agente.

A pesar de su extenso uso, SNMP tiene algunas desventajas. La más importante es que se apoya en UDP. Puesto que UDP no tiene conexiones, no existe contabilidad inherente al enviar los mensajes entre el servidor y el agente. Otro problema es que SNMP proporciona un solo protocolo para mensajes, por lo que no pueden realizarse los mensajes de filtrado. Esto incrementa la carga del software receptor. Finalmente, SNMP casi siempre utiliza el sondeo en cierto grado, lo que ocupa una considerable cantidad de ancho de banda.

Un paquete de software servidor SNMP puede comunicarse con los agentes SNMP y transferir o solicitar diferentes tipos de información. Generalmente, el servidor solicita las estadísticas del agente, incluyendo el número de paquetes que se manejan, el estado del dispositivo, las condiciones especiales que están asociadas con el tipo de dispositivo (como las indicaciones de que se cayo el enlace se terminó el papel o la pérdida de la conexión en un módem) y la carga del procesador.

El servidor también puede enviar instrucciones al agente para modificar las entradas de su base de datos MIB. El servidor también puede enviar los límites o las condiciones bajo las cuales el agente SNMP debe generar un mensaje de interrupción, como cuando la carga del CPU alcanza el 90 por ciento.

2 Tecnología y Estándares de servicios Web

Las normas en las que se basa el desarrollo de servicios Web son tecnologías en proceso de evolución. Las principales son SOAP, WSDL, UDDI y WSIL (Web Services Inspection Language, Lenguaje de inspección de servicios Web).

2.1 SOAP.

Los autores de SOAP, admiten que originalmente se enfocaron en el "acceso a objetos", pero con el tiempo se deseó que SOAP sirviera a una audiencia mucho más amplia. Por esto, el enfoque en la especificación se aparto rápidamente de los objetos, para convertirse en un Framework de mensajería XML generalizado.

El cambio de enfoque creó un problema con la "O" en el acrónimo; sin embargo, debido a su popularidad y aceptación se decidió dejar la sigla igual. La definición oficial encontrada en la más reciente especificación ni siquiera menciona los objetos.

SOAP es un protocolo liviano que permite el intercambio de información estructurada en un ambiente descentralizado y distribuido. SOAP usa la tecnología XML para definir un Framework de mensajería extensible, el cual proporciona la construcción de mensajes que pueden ser intercambiados a través de una variedad de protocolos.

Figura 3 Arquitectura SOAP

SOAP es un protocolo de mensajes independiente del transporte. Los mensajes SOAP son documentos XML. SOAP utiliza mensajes unidireccionales, aunque es posible combinar mensajes en secuencias de solicitud y respuesta. La especificación SOAP define el formato del mensaje XML, pero no su contenido ni la forma en que se envía. No obstante, SOAP especifica la forma en que los mensajes SOAP se encaminan por medio de HTTP.

Figura 4 Mensaje SOAP

A continuación se proporciona un ejemplo de un mensaje SOAP sencillo enviado por HTTP que solicita el precio actual de las acciones de Borland:

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "urn:stock-quote-services"
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/
">
<SOAP-ENV:Body>
<m:GetLastTradePrice xmlns:m="Some-URI">
<symbol>BORL</symbol>
</m:GetLastTradePrice>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

En otras palabras, SOAP es un protocolo de comunicación simple y extensible basado en mensajes, que permiten el intercambio de información estructurada y tipada, entre aplicaciones en un entorno distribuido. SOAP define una manera de transportar mensajes XML de un punto a otro.

Un mensaje SOAP es un documento XML que posee un Envelope, el cual constituye el elemento raíz del mensaje, lo cual facilita a las aplicaciones la identificación de los “Mensajes SOAP”, a través del nombre de la raíz. Las aplicaciones también pueden determinar la versión de SOAP usada inspeccionando el nombre del namespace del elemento Envelope. El Envelope contiene un Header opcional, un Body obligatorio puesto que representa el payload o información del mensaje. El Body es un contenedor genérico que puede contener cualquier número de namespaces. En este elemento es donde van los datos que se envían. Por ejemplo, el siguiente mensaje

SOAP representa una transferencia de fondos entre cuentas bancarias donde se puede notar que no hay encabezado, por que no es obligatorio.

```
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
<x:TransferFondos xmlns:x="urn:examples-org:banking">
<desde>22-342439</desde>
<hacia>98-283843</hacia>
<cantidad>100.00</cantidad>
</x:TransferFondos>
</soap:Body>
</soap:Envelope>
```

Si el receptor soporta request/response y es posible procesar el mensaje satisfactoriamente, enviará otro mensaje SOAP como respuesta al origen. En este caso, la información de respuesta para el mensaje también estaría contenida en Body como se ilustra en el siguiente mensaje:

```
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
<x:TransferFondosResponse
xmlns:x="urn:examples-org:banking">
 <balances>
   <cuenta>
    <id>22-342439</id>
    <balance>33.45</balance>
  </cuenta>
  < cuenta >
    <id>98-283843</id>
   <balance>932.73</balance>
   </ cuenta >
 </balances>
</x:TransferFondosResponse>
</soap:Body>
</soap:Envelope>
```

[3]

[3] Para complementar la información ver el tutorial de SOAP http://www.w3schools.com/soap/default.asp

2.2 WSDL.

Los servicios Web sólo resultan útiles si otras aplicaciones pueden reconocer qué hacen y la forma de llamarlos. Los desarrolladores deben estar suficientemente informados sobre un servicio Web para poder escribir un programa cliente que lo llame. WSDL es un lenguaje basado en XML que se utiliza para definir servicios Web y describe la forma de acceder a ellos.

Concretamente, describe los contratos de datos y mensajes que ofrece un servicio Web. Los desarrolladores deben examinar el documento WSDL de los servicios Web para averiguar qué métodos disponibles y cómo se realizan las llamadas con los parámetros adecuados.

Los Servicios Web son para los ordenadores, lo que las páginas Web para los humanos. Páginas que contienen cierta información o realizan cierta tarea y con la que se comunican mediante XML. Cuando navegamos por un portal, siempre encontramos en alguna parte un índice, o algo parecido. El índice nos permite conocer las partes accesibles de portal y dirigirnos a la que nos interese. Con los Servicios Web, ocurre igual. El índice es el fichero WSDL que indica al ordenador que lo consulta, qué servicios dispone en su sitio. Además, no sólo indica cuales dispone, sino que además da una referencia precisa sobre ellos, para poder invocarlos usando los parámetros adecuados.

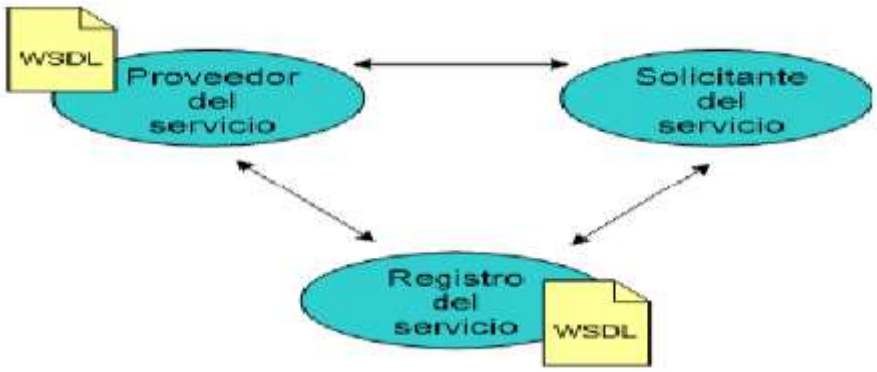

Figura 5 WSDL y Servicios Web

Un documento WSDL, define un servicio como una colección de terminales de red o puertos. Cada puerto se define de forma abstracta como un tipo de puerto, el cual soporta una colección de operaciones. Cada operación procesa un conjunto particular de mensajes. Una conexión relaciona un tipo de puerto con un protocolo específico y un formato de datos. Un puerto instancia un tipo de puerto y se conecta a una dirección de red específica. [4]

[4] Para complementar la información sobre WSDL ver http://www.w3schools.com/wsdl/default.asp

2.3 UDDI.

UDDI es una norma que trata sobre la descripción, la publicación y la localización de servicios Web.

Se trata de una especificación para el registro distribuido de información sobre servicios Web. Cuando se ha desarrollado un servicio Web y se ha creado un documento WSDL que lo describe, es necesario que exista una forma de proporcionar la información WSDL a los usuarios del servicio Web.

En cuando un servicio Web se publica en un registro UDDI, los posibles usuarios pueden buscar e informarse de la existencia de los servicios Web.

El contenido de un registro UDDI se puede comparar con las guías telefónicas. En las "páginas blancas" del registro figuran datos como el nombre, la dirección y el número de teléfono de la empresa que ofrece los servicios Web. Las "páginas amarillas" identifican el tipo de negocio y lo clasifican por sectores de actividad. Las "páginas verdes" proporcionan datos sobre los servicios Web que ofrece la empresa.

Al igual que en la Web tenemos buscadores, que nos llevan a las páginas que nos interesan, existe el concepto equivalente a nivel de Servicios Web, UDDI. UDDI es un Servicio Web en línea que se puede utilizar desde las aplicaciones para descubrir de forma dinámica otros servicios en línea, todos ellos perfectamente integrados en una interfaz XML simple.

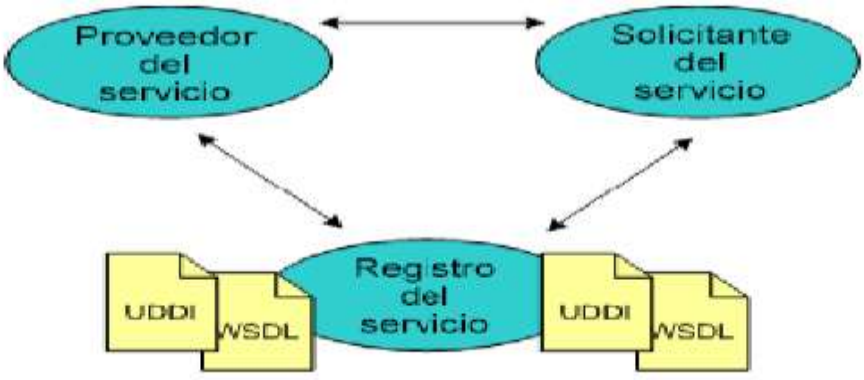

Figura 6 WDDI y Servicios Web

La especificación UDDI ha sido desarrollada por IBM, Microsoft y Ariba y está soportada por más de 300 empresas, incluyendo Oracle, Sun Microsystems y Nortel Networks. [5]

[5] Ver mayor información sobre UDDI en http://www.uddi.org

3 Tecnologías utilizadas para la construcción de WSMINING

3.1 Struts

Es un framework que implementa el patrón de arquitectura MVC (Modelo – Vista - Control) en Java. Un framework es la extensión de un lenguaje mediante una o más jerarquías de clases que implementan una funcionalidad y que (opcionalmente) pueden ser extendidas. El framework puede involucrar TagLibraries.

El patrón de arquitectura MVC es un patrón que define la organización independiente del Modelo (Objetos de Negocio), la Vista (interfaz con el usuario u otro sistema) y el Control (controlador del workflow de la aplicación: "si estoy aquí y me piden esto entonces hacer tal cosa, si sale bien mostrar esto y sino aquello otro").

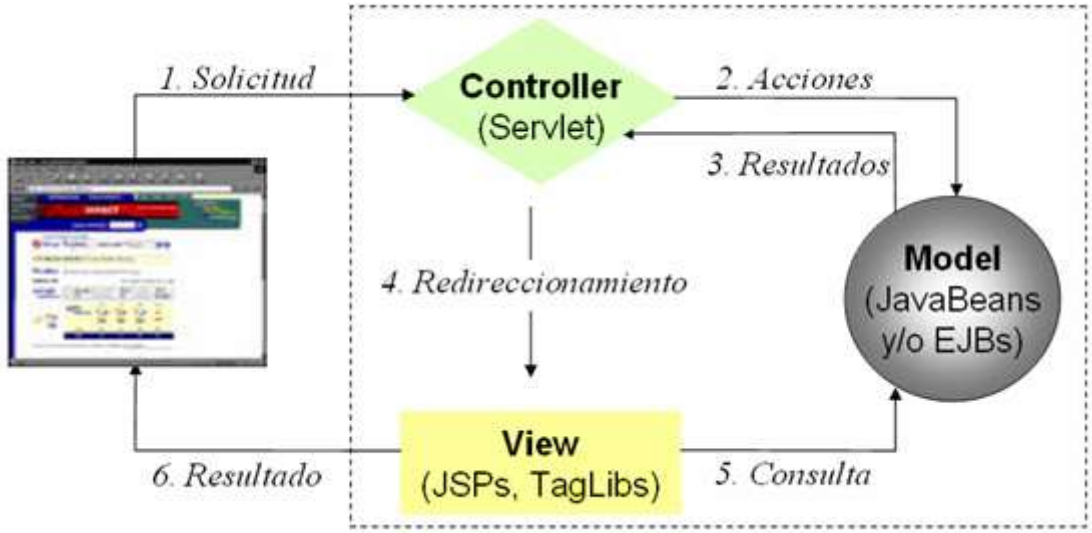

Figura 7 Arquitectura de Struts

El Control comprende la funcionalidad involucrada desde que un usuario genera un estímulo (clic en un link, envío de un formulario, etc.) hasta que se genera la interfaz de respuesta. Entre medio, llamará a los objetos de negocio del Modelo para que resuelvan funcionalidad propia de la lógica de negocio y según el resultado de la misma ejecutará el JSP (Java Server Page) que deba generar la interfaz resultante.

Struts[i] incluye un servlet que a partir de la configuración de struts-config.xml recibe las solicitudes del usuario, llama al Action Bean que corresponda y, según lo que éste retorne, ejecuta una JSP. Por consiguiente, las tareas que se deben realizar son:

1. Escribir una clase Action que extienda de org.apache.action.Action.
2. Configurar el **struts-config.xml** para que incluya el nuevo action mapping y sus posibles forwards de salida.

Por ejemplo:

```
<struts-config>
...
  <action-mappings>
...
    <action path="/logoff"
type="com.empresa.aplicacion.LogoffAction">
      <forward name="success" path="/index.jsp"/>
    </action>
  </action-mappings>
...
```

El patrón de arquitectura MVC que permite obtener una aplicación ordenada y completamente modular y la seguridad ofrecida en el encapsulamiento de los JSP, son dos de las más fuertes razones por la cual se decidió utilizar Struts. De esta forma, la aplicación queda preparada para posibles cambios y mejoras que permitan evolucionar sin tanto traumatismo.

3.2 Java Server Page

Una Java Server Page[6] es un archivo de texto, que combina tags HTML con nuevos tags de scripting Java.

Básicamente es un archivo HTML con código JAVA intercalado. El contenido de una JSP, se divide en dos categorías:

- ✓ Los elementos que son procesados en el servidor.
- ✓ Las plantillas de datos (HTML) que son ignoradas por el contenedor JSP.

Una JSP se ejecuta en el entorno de un contenedor JSP, que está instalado en un servidor Web.

Cuando el *browser* pide al contenedor una página, el contenedor encapsula el requerimiento y lo direcciona al JSP apropiado junto con un objeto respuesta. El JSP "trabaja" y modifica el objeto respuesta. Finalmente, el contenedor encapsula la respuesta y la devuelve al *browser*.

Los JSP están contemplados dentro de los Struts como los elementos que permiten mostrar las interfaces graficas a los usuarios, por tal motivo, es una tecnología que necesita ser implementada si se desea utilizar Struts.

[6] Para mayor información ver http://java.sun.com/products/jsp/

3.3 Componentes Java (Java Beans).

Un Java Bean[7] o bean es un componente hecho en software que se puede reutilizar y que puede ser manipulado visualmente por una herramienta de programación en lenguaje Java.

Para ello, se define un interfaz para el momento del diseño (design time) que permite a la herramienta de programación, interrogar al componente y conocer las propiedades (**properties**) que define y los tipos de sucesos (**events**) que puede generar en respuesta a diversas acciones.

Aunque los beans individuales pueden variar ampliamente en funcionalidad desde los más simples a los más complejos, todos ellos comparten las siguientes características:

- **Introspection**: Permite analizar a la herramienta de programación o IDE como trabaja el bean
- **Customization**: El programador puede alterar la apariencia y la conducta del bean.
- **Events**: Informa al IDE de los sucesos que puede generar en respuesta a las acciones del usuario o del sistema, y también los sucesos que puede manejar.
- **Properties**: Permite cambiar los valores de las propiedades del bean para personalizarlo (customization).
- **Persistence**: Se puede guardar el estado de los beans que han sido personalizados por el programador, cambiando los valores de sus propiedades.

En general, un bean es una clase que obedece ciertas reglas:

- Un bean debe tener un constructor por defecto (sin argumentos)
- Un bean tiene persistencia, es decir, implementar el interface *Serializable*.
- Un bean tiene introspección (**instrospection**). Los IDE reconocen ciertas pautas de diseño, nombres de las funciones miembros o métodos y definiciones de las clases, que permiten a la herramienta de programación mirar dentro del bean y conocer sus propiedades y su conducta.

3.4 JDBC (Java Database Connectivity)

JDBC[8] es una especificación de un conjunto de clases y métodos de operación que permiten a cualquier programa Java acceder a sistemas de bases de datos de forma homogénea. Lógicamente, al igual que ODBC, la aplicación de Java debe tener acceso a un controlador JDBC adecuado. Este controlador es el que implementa la funcionalidad de todas las clases de acceso a datos y proporciona la comunicación entre el API JDBC y la base de datos real.

[7] Para mayor información ver http://java.sun.com/products/javabeans/
[8] Para mayor información ver http://java.sun.com/products/jdbc/

Toda la conectividad de bases de datos de Java se basa en sentencias SQL, por lo que se hace imprescindible un conocimiento adecuado de SQL para realizar cualquier clase de operación de bases de datos. Aunque, afortunadamente, casi todos los entornos de desarrollo Java ofrecen componentes visuales que proporcionan una funcionalidad suficientemente potente sin necesidad de que sea necesario utilizar SQL, aunque para usar directamente el JDK se haga imprescindible.

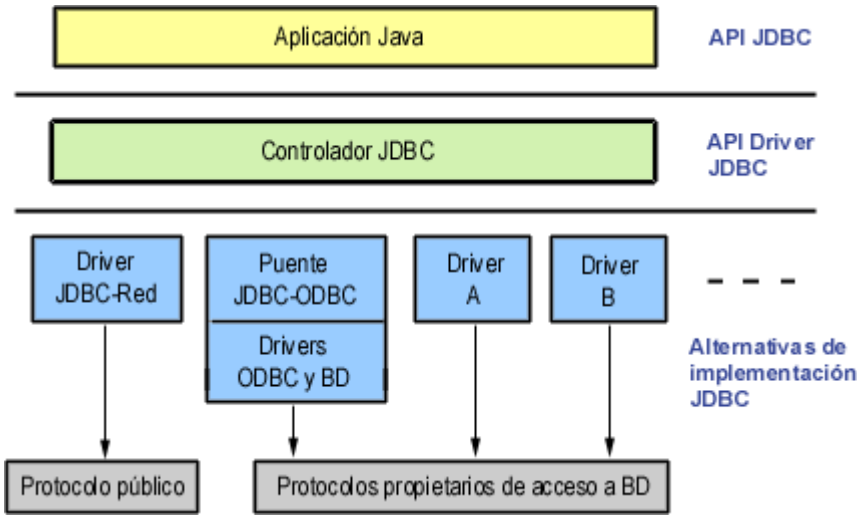

Figura 8 Arquitectura de JDBC

La aplicación esta completamente construida en lenguaje Java y es necesario establecer conexiones con diferentes bases de datos, por tal motivo JDBC es la mejor opción para obtener y enviar datos desde y hacia las bases de datos.

3.5 JMX (Java Mangement Extensions).

Las Extensiones de Gestión Java (JMX) han sido creadas para reunir los requerimientos de gestión dinámica en el mercado y para ofrecer las herramientas correctas a los diseñadores y desarrolladores de soluciones de gestión. Este nuevo estándar es apropiado para adaptar los sistemas, implementar nuevas soluciones de gestión y preparar los sistemas para el futuro. JMX es la respuesta definitiva a la gestión Java, para todos los usos de esta tecnología y donde la gestión sea necesaria.

JMX[ii] proporciona un camino simple y liviano para implementar objetos java. La implementación con JMX tiene muy pocos contratiempos porque es totalmente independiente de la infraestructura de gestión. Esto significa que un recurso puede ser administrable sin importar cómo está implementado su gestor, por ejemplo éste puede ser un NMS (Network Management System, Sistema de Gestión de Red) que utilice SNMP para interactuar con los agentes.

Usando JMX, adicionar capacidad de gestión a una aplicación Java es simple. La implementación compatible con JMX puede ser añadida a cualquier aplicación Java, algunas veces con tan sólo 4 o 5 líneas de código. Esto permite a los desarrolladores (cualquiera que sea su segmento del mercado), concentrarse en el núcleo de su negocio en lugar de gastar esfuerzo excesivo en adicionar capacidad de gestión. Esto efectivamente separa la creación de dispositivos y servicios del desarrollo de sistemas de gestión. JMX define cómo hacer que su producto sea gestionable, no cómo gestionarlo.

JMX define una arquitectura de gestión, API's y servicios de gestión, todos bajo una sola especificación, desarrollada por Sun Microsystems y los líderes de la industria en el área de gestión, siguiendo el JCP (Java Community Process).

La arquitectura de JMX se construye a partir de un modelo en tres niveles. Esto da flexibilidad permitiendo usar partes de la especificación individualmente por diferentes comunidades de desarrolladores que utilizan la tecnología de Java.

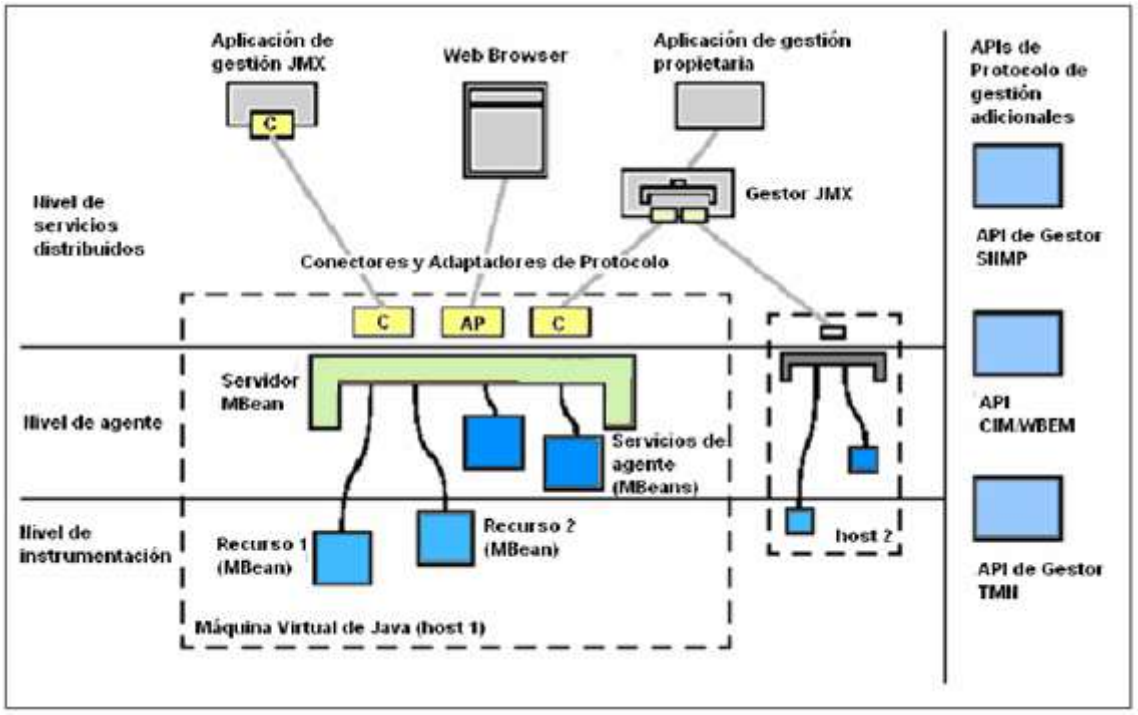

Figura 9 Arquitectura JMX

Como JMX es un conjunto de extensiones de gestión basadas en Java que se destaca por la incorporacion de interfaces que permiten interoperar con herramientas adicionales, tales como los Servicios Web, presenta facilidad de manejo y además por las caracteristicas ya mensionadas, es que JMX es una excelente opción para ser utilizada en nuestra aplicación.

3.6 Enterprise Java Beans (EJB)

La tecnología EJB[iii] se puede definir como un marco de trabajo de componentes basados en Java, del lado del servidor y reutilizables para aplicaciones distribuidas. Este marco de trabajo facilita el manejo centralizado de la lógica de negocio, el despliegue declarativo y la persistencia de objetos. Algunas de las características del marco de trabajo EJB son:

- Los EJBs son objetos Java ejecutados dentro de un entorno servidor.
- Los EJBs pueden distribuirse entre varias máquinas y se puede acceder a ellos de forma remota como si estuvieran en una máquina local.
- Los EJBs son manejados de forma centralizada por un contenedor.

Junto a la especificación EJB y la arquitectura hay un artefacto conocido como 'Enterprise Java Bean' (que, como al marco de trabajo, normalmente se le da el acrónimo EJB). Aunque los nombres del marco de trabajo EJB son muy similares a los nombres del marco de trabajo JavaBeans, las especificaciones de diseño para cada uno de ellos son muy diferentes.

Los nombres de las especificaciones EJB y JavaBeans y de sus componentes son muy similares, ahí es donde terminan las similitudes. La especificación y el modelo de componentes de JavaBeans y el modelo de componentes Enterprise JavaBeans tienen muy poco en común.

3.6.1 Entorno de Ejecución EJB.

Un JavaBean Enterprise es un objeto Java que expone algunos interfaces y que vive y se ejecuta dentro de un entorno de ejecución conocido como contenedor EJB. Un Enterprise JavaBean no puede existir fuera de un contenedor EJB. De hecho, la relación entre un contenedor EJB y un EJB es conocida como "inversión de control".

Un contenedor EJB es un entorno de ejecución para controlar beans enterprise. El contenedor almacena y maneja un bean enterprise de la misma manera que un motor servlet hospeda un servlet Java. El contenedor EJB ejemplariza y controla los beans enterprise y les proporciona servicios de bajo nivel.

Una aplicación cliente que desee interactuar con un bean enterprise no accede directamente a él. En lugar de eso, el bean está aislado de la aplicación cliente mediante el contenedor.

3.6.2 Servicios Proporcionados por un Contenedor EJB.

Un desarrollador que vaya a crear las clases e interfaces a partir de las cuales se creará un objeto bean enterprise puede asumir que el contenedor EJB tendrá disponibles los siguientes servicios de bajo nivel:

- Maneja las transacciones del bean.
- Proporciona seguridad al bean
- Proporciona persistencia para el bean
- Acceso remoto al bean
- Control del ciclo de vida del bean
- Repositorio de conexiones a la base de datos
- Repositorio de ejemplares del bean

Como el contenedor EJB maneja el paquete de servicios a nivel de infraestructura para un bean enterprise, un programador se puede concentrar en programar la lógica de negocio de ese bean.

3.6.3 Tipos de Beans Enterprise.

La arquitectura EJB define tres tipos de beans enterprise distintos:

- Beans dirigidos-por-mensaje.
- Beans de sesión.
- Beans de entidad.

A los beans de sesión y de entidad se les invoca síncronamente mediante un cliente bean enterprise. A los MDB (Message Driven Bean - beans dirigidos por mensaje) los invoca un contenedor de mensaje, como un tópico publicar/suscribir.

- **Beans Dirigidos por Mensaje**.

Un MDB es un bean enterprise que maneja de forma asíncrona los mensajes entrantes. Normalmente, un ejemplar de MDB actúa como un oyente de mensajes enviados desde un tópico publicar/subscribir. Un MDB puede recibir mensajes compatibles con JMS.

Un contenedor EJB maneja el ciclo de vida de un MDB, sin embargo, al contrario que los beans de sesión y entidad; los programas clientes no se dirigen al MDB con llamadas a métodos. En vez de eso, el MDB recibe mensajes de una fuente de mensajes a través de un método de retrollamada, onMessage.

- **Beans de Sesión.**

Un bean de sesión representa un único cliente y no se comparte entre clientes. Un cliente llama a los métodos del bean de sesión, que son dirigidos a través del contenedor EJB al bean enterprise. El bean de sesión realiza la lógica de negocio para el cliente y el contenedor devuelve el control al cliente. Un bean de sesión no persiste entre varias sesiones. Hay dos tipos de beans de sesión:

> **Beans de Sesión con Estado**: Un bean de sesión con estado mantiene un estado conversacional con un cliente durante la duración de una sesión. Esto implica que el bean de sesión con estado puede mantener ejemplares de

variables a través de varias llamadas desde un cliente durante una única sesión.

Beans de Sesión sin Estado: Un bean de sesión sin estado no mantiene un estado conversacional para cada cliente individual. Cada llamada a un bean de sesión sin estado debería considerarse como una petición a un ejemplar totalmente nuevo, ya que cualquier estado de las variables de ejemplar se perderá entre dos llamadas.

El contenedor EJB no persiste los beans de sesión sin estado en el almacenamiento secundario; por lo tanto un programador debe reconocer que todos los datos son temporales entre llamadas de un cliente. La naturaleza temporal de los beans de sesión sin estado permite a un contenedor EB reutilizar ejemplares de beans y por lo tanto, normalmente optimiza el rendimiento de los beans.

La siguiente imagen muestra como un contenedor EJB actúa como un Proxy entre el cliente EJB y el propio ejemplar de EJB. Esto le da al contendor la libertad de hacer invocaciones a los ejemplares del EJB y controlar así el ciclo de vida de cada ejemplar EJB.

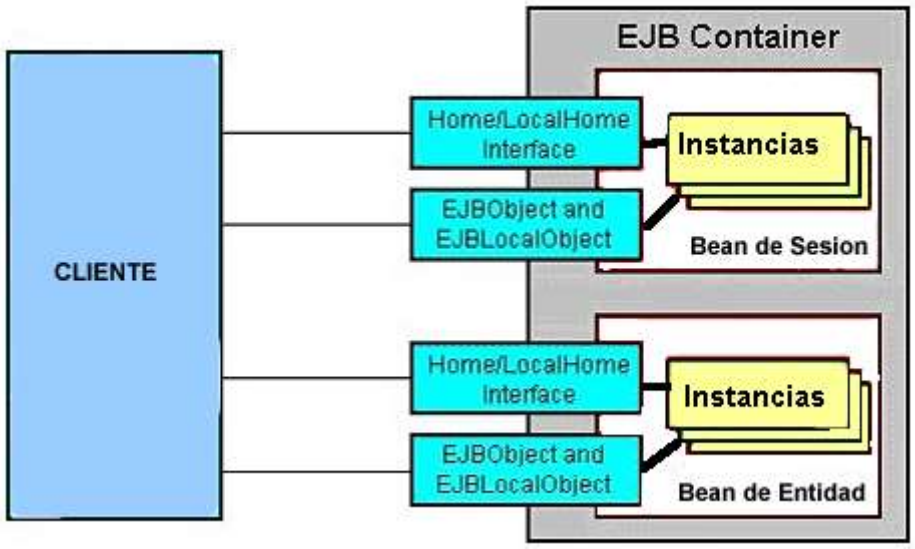

Figura 10 Contenedor EJB

- **Beans de Entidad.**

Un bean de entidad está pensado para representar la lógica de negocio de una entidad existente en un almacenamiento persistente, como una base de datos relacional. Los beans de entidad comparten algunas cualidades que podría encontrar en una base de datos relacional, por ejemplo:

- Los beans de entidad son persistentes: el estado de un bean de entidad existe más allá del ciclo de vida de la aplicación en la que se ha creado, o

incluso más allá del ciclo de vida del contenedor EJB. Esto implica que l contenedor EJB puede restaurar el bean de entidad a su estado original.

- Los beans de entidad permiten compartir accesos: los beans podrían compartirse entre varios clientes y la concurrencia la maneja el contenedor.
- Los beans de entidad tienen claves primarias: Existen clases Primary -key para identificar un ejemplar de un bean de entidad. La clave primaria contiene toda la información necesaria para encontrar un bean almacenado.
- Los beans de entidad podrían participar en relaciones: Se han presentado interfaces locales para manejar las relaciones entre beans.
- Los beans de entidad pueden participar en transacciones: como varios clientes pueden acceder y modificar los datos, es importante para los beans de entidad poder especificar las propiedades transaccionales para su interacción. Las propiedades de transacción es especifican declarativamente en descriptores de despliegue, y los límites de transacción los maneja el contenedor.

El mapeo objeto-relacional implícito en los beans de entidad requiere que un bean de entidad sea responsable de insertar, actualizar, seleccionar y eliminar datos dentro de la fuente de datos. Este proceso del manejo de la comunicación entre el componente y la fuente de datos se llama persistencia. En otras palabras, **persistencia** es *el proceso de escribir la información en una fuente de datos externa*. Hay dos tipos de persistencia para los beans de entidad:

Persistencia Manejada por el Bean (BMP): Con la persistencia manejada por el bean (BMP), el programador es el responsable de escribir dentro del bean todo el código para acceder a la fuente de datos. Este tipo de persistencia le da más flexibilidad al programador porque controla todos los accesos a la fuente de datos.

Persistencia Manejada por el Contenedor (CMP): Con la persistencia manejada por el contenedor EJB es el propio contenedor EJB el que maneja todos los accesos a la base de datos requeridos por el bean de entidad. Como resultado, el código de acceso a los datos del bean, no está acoplado a una fuente de datos específica. Esto libera al programador de tener que escribir código de acceso a los datos y permite que el bean de entidad se pueda desplegar en diferentes contenedores y/o contra diferentes fuentes de datos.

La siguiente imagen muestra el camino de una petición de cliente hasta la capa de aplicación. Una vez que el servlet controlador recibe la petición, la convierte a una petición de servicio de negocio y llama a los servicios de negocio apropiados en la capa de servicio de negocio. Los servicios de negocio utilizan uno o más beans de entidad para cargar y grabar datos desde los recursos de la capa de datos.

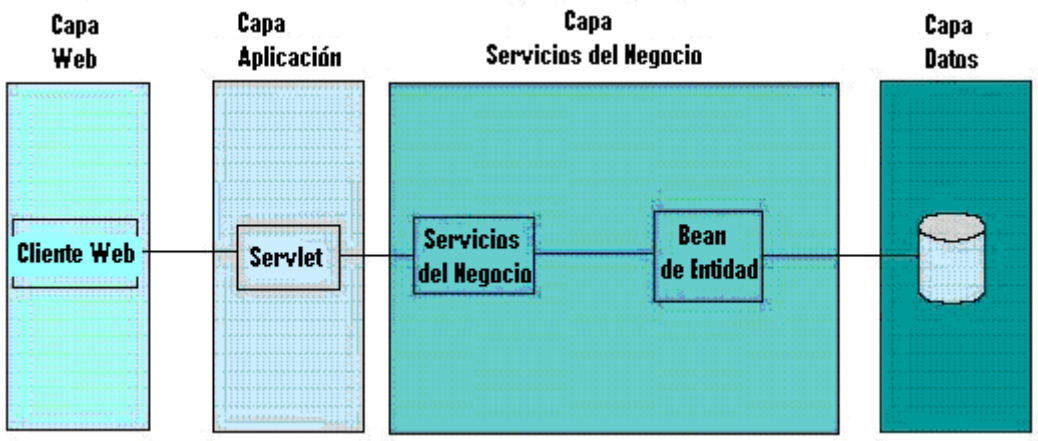

Figura 11 Petición desde un cliente a un EJB de Entidad

4 Herramientas de desarrollo utilizadas para WSMINING

4.1 Servidor de aplicaciones JBOSS.

JBoss es una implementación Open-Source de un "EJB Container"; es mediante este tipo de productos que es posible llevar acabo un desarrollo con EJB's "Enterprise Java Bean's". Este tipo de producto ("EJB Container") generalmente no es distribuido como producto individual y por esta razón se le pudiera considerar a "JBoss" como un producto diferente más no único.

La declaración anterior merece un poco más detalle, la gran gamma de productos en este ramo de Java (J2EE para ser más exactos) han sido comercializados como "Java Aplication Servers"

Como se observa en la siguiente gráfica un "Java Application Server" se encuentra compuesto por dos partes: un "Servlet Engine" y un "EJB Engine", dentro del "Servlet Engine" se ejecutan exclusivamente las clásicas aplicaciones de servidor (JSP's y Servlets), mientras el "EJB Engine (Container)" es reservado para aplicaciones desarrolladas alrededor de EJB's (Enterprise Java Bean's).

Casi todos los "Application Servers" en el mercado hoy en día son conocidos como "Fully J2EE Compliant", este término implica que se cumplen todas las especificaciones J2EE definidas por "Sun" y es aquí donde es notable la diferencia con JBoss.

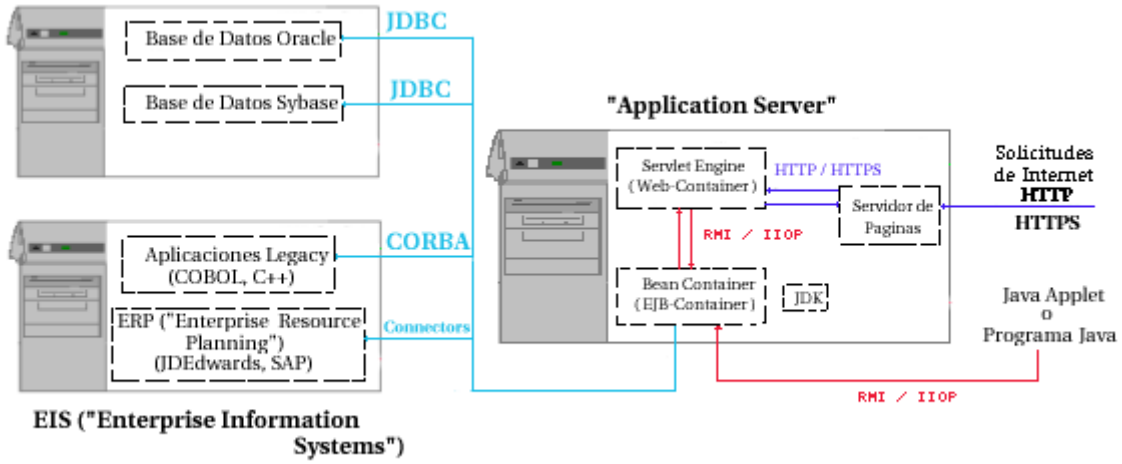

Figura 12 Arquitectura de un Java Application Server

No existe una clara distinción entre el "Web-Container" y "EJB Container", esto es, es posible ejecutar tanto JSP/Servlets así como EJB's, sin embargo, el ambiente se encuentra altamente integrado para que sea transparente (al menos para el programador final) la comunicación entre JSP/Servlets y EJB's.

El producto JBoss es únicamente un "EJB Container" y es por esto que generalmente se utiliza en conjunción con un "Web-Container", el "Web-Container" puede ser cualquiera disponible en el mercado, sin embargo, cuando obtenga JBoss se le ofrecerá la opción de incluir Tomcat o Jetty; lo anterior no restringe a JBoss para operar con otro "Web Container" como ServletExec, la única ventaja de utilizar aquellos "Web Containers" (Servlet Engines) incluidos con JBoss será en tiempo de coordinación/configuración entre JBoss|"x" Web-Container, y siendo que un ambiente utilizando EJB's es altamente complejo es preferible concentrarse en algo que ya ha sido utilizado y depurado.

4.2 Apache Tomcat

Tomcat es un contenedor de Servlets con un entorno JSP. Un contenedor de Servlets es un núcleo de ejecución que maneja e invoca servlets por cuenta del usuario. Podemos dividir los contenedores de Servlets en:

- **Contenedores de Servlets Stand-alone (Independientes):** Estos son una parte integral del servidor Web. Este es el caso cuando usando un servidor Web basado en Java, por ejemplo, el contenedor de servlets es parte de Java Web Server (actualmente sustituido por **iPlanet**). Este el modo por defecto usado por Tomcat. Sin embargo, la mayoría de los servidores, no están basados en Java, los que nos lleva los dos siguientes tipos de contenedores:

- **Contenedores de Servlets dentro-de-Proceso:** El contenedor Servlet es una combinación de un plugin para el servidor Web y una implementación de contenedor Java. El plugin del servidor Web abre una JVM (Máquina Virtual Java) dentro del espacio de direcciones del servidor Web y permite que el contenedor Java se ejecute en él. Si una cierta petición debería ejecutar un servlet, el plugin toma el control sobre la petición y lo pasa al contenedor Java (usando JNI). Un contenedor de este tipo es adecuado para servidores multi - hilo de un sólo proceso y proporciona un buen rendimiento pero está limitado en escalabilidad.

- **Contenedores de Servlets fuera-de-proceso:** El contenedor Servlet es una combinación de un plugin para el servidor Web y una implementación de contenedor Java que se ejecuta en una JVM fuera del servidor Web. El plugin del servidor Web y el JVM del contenedor Java se comunican usando algún mecanismo IPC (normalmente sockets TCP/IP). Si una cierta petición debería ejecutar un servlet, el plugin toma el control sobre la petición y lo pasa al contenedor Java (usando IPC). El tiempo de respuesta en este tipo de contenedores no es tan bueno como el anterior, pero obtiene mejores rendimientos en otras cosas (escalabilidad, estabilidad, etc.).

Tomcat puede utilizarse como un contenedor solitario (principalmente para desarrollo y depuración) o como plugin para un servidor Web existente (actualmente se soportan los servidores Apache, IIS y Netscape). Esto significa que siempre que despleguemos Tomcat tendremos que decidir cómo usarlo, y, si seleccionamos las dos últimas opciones, también necesitaremos instalar un adaptador de servidor Web.

4.3 Firebird

Firebird[9] es un proyecto de software libre cuyo código es una evolución del de Interbase. Es, por tanto, un gestor de bases de datos relacionales con un buen soporte del estándar ANSI SQL-92.

Algunas de las ventajas principales de Firebird:

- Arquitectura multigeneracional**.-** Es la forma en que Firebird administra la concurrencia en las actualizaciones en los datos, así como el manejo de las transacciones. Nos asegura que no habrá bloqueos a nivel de página de datos ni de registro, ya que cada vez que se abre una transacción, Firebird genera una copia de los datos para ese usuario.

- Triggers, o disparadores.- Firebird cuenta con una de las implementaciones de triggers más completas comparada con otras bases de datos. Los triggers permiten

[9] Para mayor información ver http://firebird.sourceforge.net/

la realización de acciones cada vez que se agrega, modifica o elimina un registro. De esta manera, podemos implantar reglas de negocio desde el nivel de la base de datos.

- Procedimientos almacenados.- Funcionan de manera similar a los triggers, con la diferencia de que pueden ser ejecutados de manera independiente a las acciones que se ejecuten sobre los registros.

- Integridad referencial.- Permite establecer reglas de integridad entre tablas, para que no violen los principios de las relaciones entre tablas maestro-detalle.

- Seguridad integrada.- Firebird mantiene su lista de usuarios, y es necesario que se registre el usuario cada vez que se conecta a la base de datos. Además, se pueden asignar permisos independientes de acceso, modificación inserción y eliminación a por tabla a cada usuario.

Firebird cuenta con muchas más características, como los generadores de números consecutivos, excepciones definidas por el usuario, además de que funciona en varias plataformas (en estos momentos Linux, Solaris, Mac OSX, HP-UX y Windows).

4.4 Weka

Como ya se a mencionado, WSMF es el framework a utilizar y esta construido totalmente en lenguaje JAVA, por tal motivo es práctico utilizar JAVA como el lenguaje utilizado en la construcción del Data Warehouse y Data Mining, y de esta forma poder ofrecer compatibilidad a toda la aplicación.

Existen varias herramientas de aprendizaje escritas en Java, pero WEKA (Waikato Environment for Knowledge Analysis) es quizá la más popular dentro del mundo Machine Learning. Es un paquete que por lejos es el mejor de todas las herramientas disponibles en Internet [iv]. Esta herramienta es ampliamente aceptada por los cursos de ciencia en computación en Machine Learning y por las universidades alrededor del mundo.

El programa WEKA [v] es una herramienta que permite realizar minería de datos con una interfaz gráfica lo que facilita su utilización. Además, permite una comparación con los distintos métodos que se utilizan para el pre-procesamiento, clasificación de información, clustering y meta-aprendizaje. WEKA proporciona una plataforma para evaluar un problema con distintas combinaciones de algoritmos y poder extraer conocimiento interesante.

La herramienta utiliza los paquetes definidos por WEKA para realizar el proceso de minería de datos. La ejecución de todo el proceso se hace de forma automática hasta

que el algoritmo entrega una salida. Una vez el usuario tiene la salida él puede visualizarla en su browser y saber tanto el comportamiento del SLA como del algoritmo utilizado.

4.4.1 Pre-procesamiento

Abriendo Archivos

La herramienta de WEKA presenta tres opciones para cargar el archivo con el cual se desea trabajar:

1 Open File…

2 Open URL…

3 Open DB…

La opción que se necesita es **Open DB…** porque después de diseñar y construir el Data Warehouse, hay que recuperar los datos desde una base de datos multidimensional y trabajar con ellos. Después de seleccionar esa opción es necesario ingresar información para especificar la tabla y la base de datos con la cual se desea trabajar, se tiene que especificar: **databaseURL**, **password, query, sparceData** y **username**, ver figura 13**.** El ingreso de estos parámetros se hace de forma automática por la aplicación, de hecho, el único parámetro que varía es **query**; puesto que depende de la tabla con que se quiera trabajar y está a su vez depende del acuerdo de nivel de servicio seleccionado. Los otros campos son comunes para toda la aplicación.

Como se menciona en el capitulo 4, la etapa de pre-procesamiento se realiza con la construcción del Data Warehouse. Por eso en esta etapa no es necesario modificar los atributos ni utilizar otra funcionalidad sino solamente cargarlos a la herramienta para empezar el proceso de minería de datos.

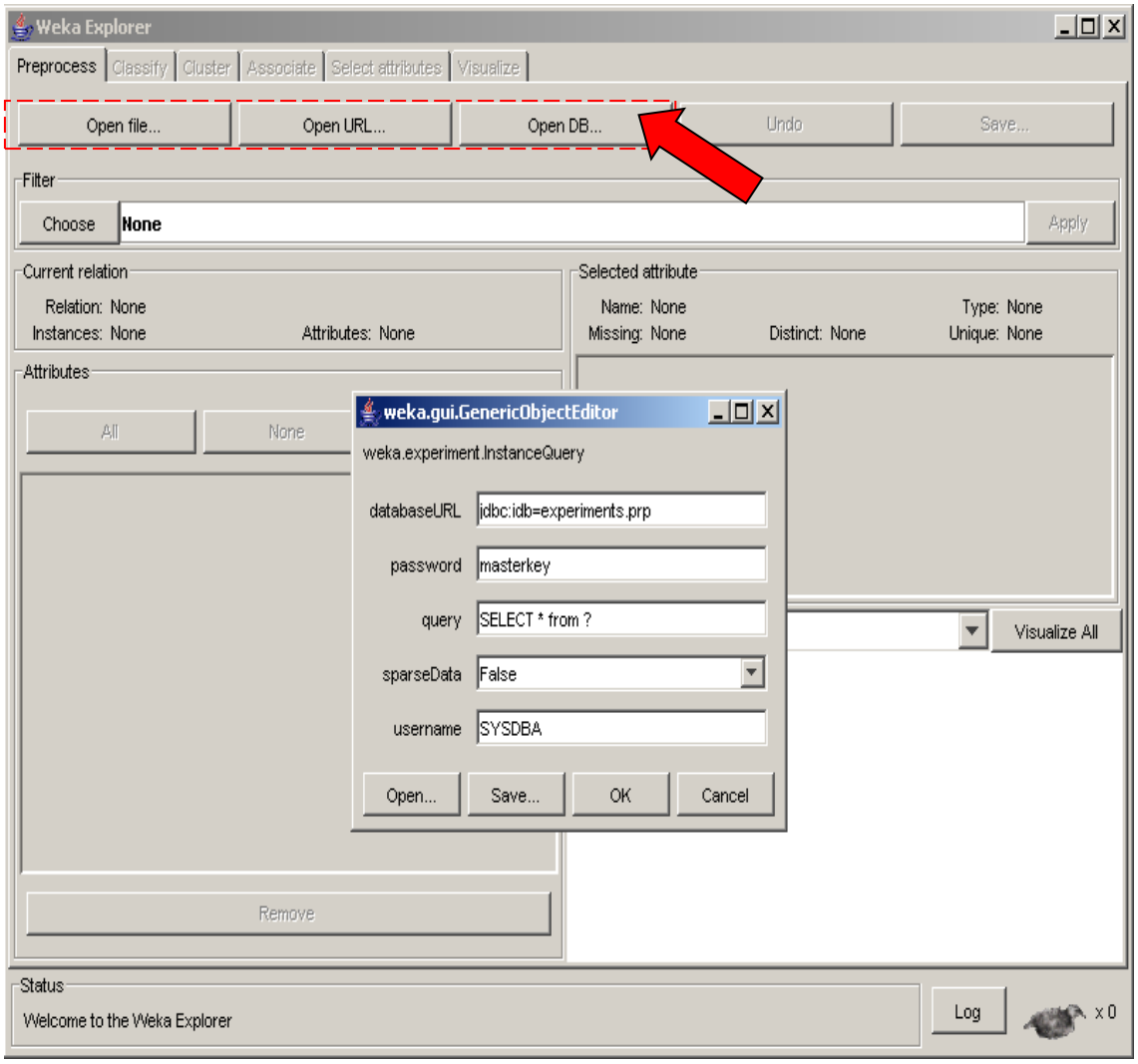

Figura 13 Seleccionar la Base de Datos

4.4.2 Clasificación

Seleccionar un Clasificador

El algoritmo que se utiliza, de acuerdo a los requerimientos del proyecto, es el árbol de decisión C4.5 (ver el capitulo 4). Esté algoritmo permite, rápidamente y con exactitud, analizar grandes cantidades de datos recolectados rutinariamente desde una red y almacenados en una base de datos que tiene tablas con pocos atributos.

El algoritmo J4.8, que es la implementación WEKA del árbol de decisión C4.5. J4.8 actualmente implementa la última y ligera versión mejorada llamada C4.5 Revisión 8, que es la última versión pública de esta familia de algoritmos antes de la C5.0, una implementación comercial.

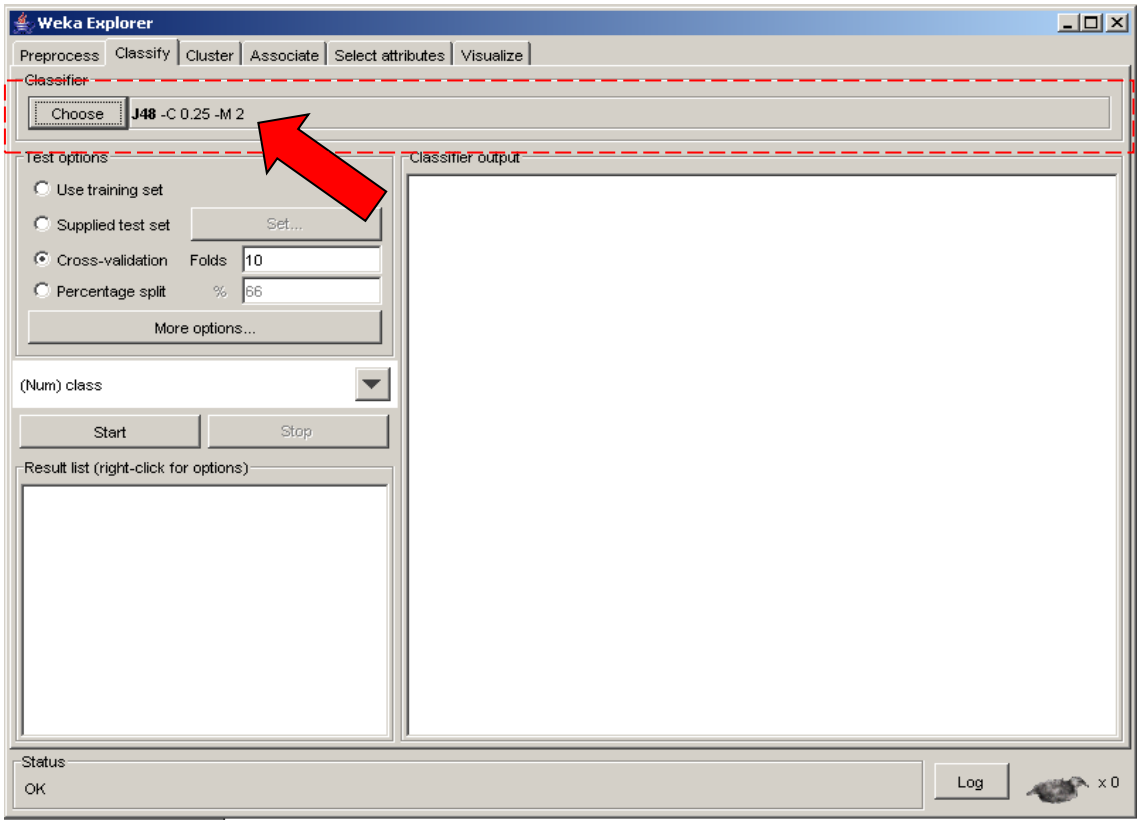

Figura 14 Seleccionar el algoritmo

Opción de prueba

El resultado obtenido, aplicando el clasificador J4.8 o cualquier otro, puede ser probado dependiendo de la *opción de prueba* (Test Options) seleccionada. La herramienta permite seleccionar entre cuatro opciones:

- Use training set
- Supplied test set
- Cross - Validation
- Porcentaje split

La opción de prueba utilizada por nosotros en la aplicación es la **Cross - Validation,** esta opción permite evaluar el clasificador por medio de una validación cruzada.

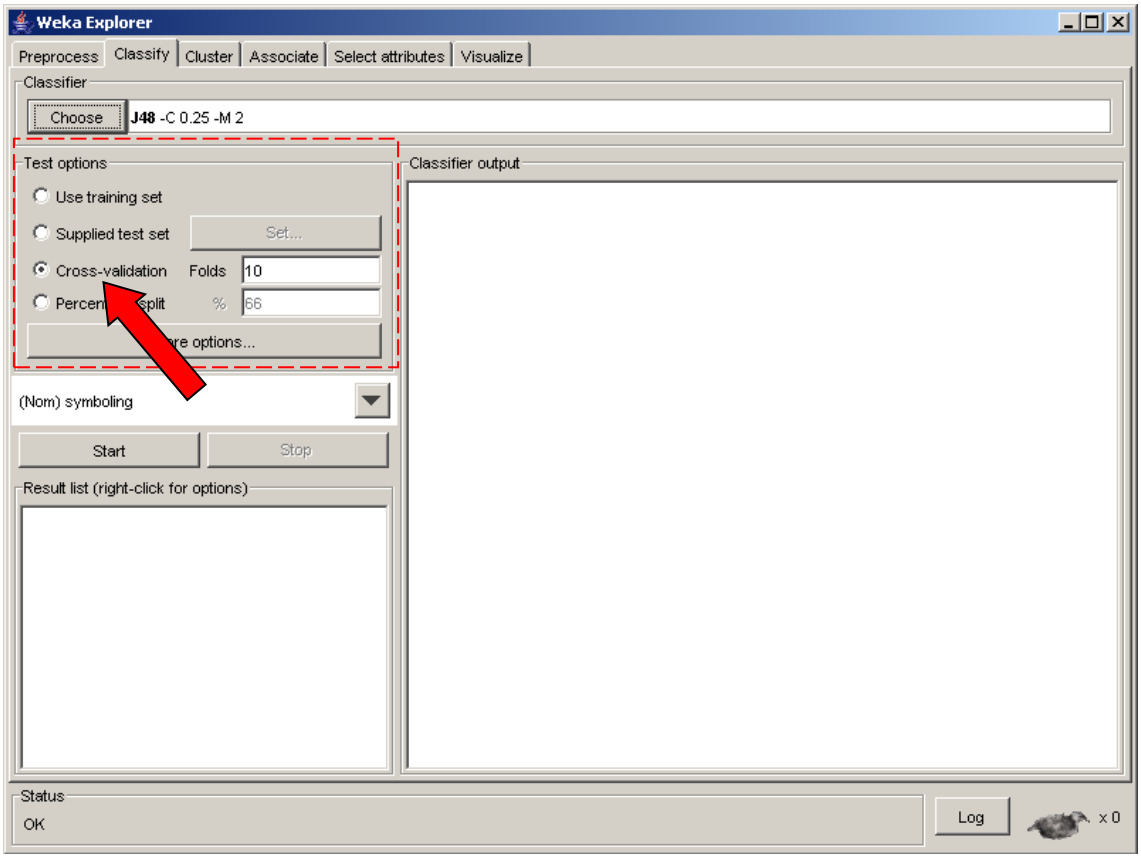

Figura 15 Opción de Prueba

4.4.3 El Atributo Class

Cuando WEKA se utiliza para clasificación usa por defecto el último atributo de la tabla seleccionada como atributo “Class”. El atributo Class es utilizado como el atributo objetivo para predicción. Es un atributo especial y debe ser previamente organizado para obtener buenos resultados. Ya que nosotros utilizamos el árbol de decisión como clasificador debemos definir siempre nuestro atributo *Class* nominal con el fin que solamente pueda tomar valores discretos y se pueda realizar una correcta predicción.

4.4.4 Comenzando el clasificador

Una vez el clasificador, la opción de prueba y el atributo *Class* sean definidos el proceso de aprendizaje puede empezar. Todos estos prerrequisitos la aplicación los define automáticamente cada vez que el usuario desea ver el estado del cumplimiento del SLA.

Cuando el clasificador comienza el algoritmo empieza a trabajar (junto con su opción de prueba) y después de unos pocos segundos tiene lista una respuesta. Es en este momento cuando varias cosas ocurren, la salida del clasificador entrega gran cantidad

de texto describiendo el resultado del algoritmo y las pruebas. Y varias opciones de visualización de los resultados aparecen, ver la figura 16.

4.4.5 El texto de salida del clasificador

La salida esta dividida en varias secciones con la siguiente información: *Run information, J4.8 pruned tree, Stratified cross – validation, Detailed Accurasy by class y Confusion Matrix.* En la figura 16 se puede observar dos tipos de información la *Run information* y *J4.8 pruned tree*. A continuación hablamos un poco de que información proporciona cada una de ellas:

- **Run information.** Una lista con información obtenida de las opciones del esquema de aprendizaje seleccionado, el nombre del esquema, las instancias, los atributos y el modo prueba que esta involucrado en el proceso.
- **Modelo de clasificación.** Una representación textual del modelo de clasificación que se ha obtenido en todo el proceso.
- **El resultado del modo de prueba seleccionado.** En nuestro caso, siempre va a ser la validación cruzada.
- **Resumen.** Una lista estadística que resume exactamente como el clasificador pudo predecir la verdad de *Class* de las instancias bajo el modo de prueba escogido.
- **Detalle de la exactitud para *Class.*** Más detalle de la *Class* bajo análisis de la exactitud de la predicción del clasificador.
- **Matriz de confusión.** Muestra como las instancias han sido asignadas a cada clase. Los elementos muestran el número de ejemplos probados cuya clase actual es la fila y cuya clase pronosticada es la columna.

Toda esta información es suministrada a los usuarios de nuestra aplicación agrupada en las secciones anteriores; donde cada una de ellas se visualiza en una tabla. El usuario puede escoger que sección del resultado desea ver. A continuación vamos a describir un típico ejemplo de minería de datos que nos permita comprender un poco mejor la información que nos esta entregando WEKA como resultado de la clasificación.

El ejemplo utiliza el tipo de clasificación J4.8, la validación cruzada como opción de prueba y el atributo de predicción nominal (toma solamente dos valores discretos Yes o No). Este ejemplo se acomoda perfectamente a nuestros requerimientos, lógicamente con atributos (dimensiones) diferentes, pero en esencia se tiene la misma

estructura de la aplicación. Se utiliza el mismo tipo de clasificador, validación cruzada y la variable de predicción toma dos valores complementarios.

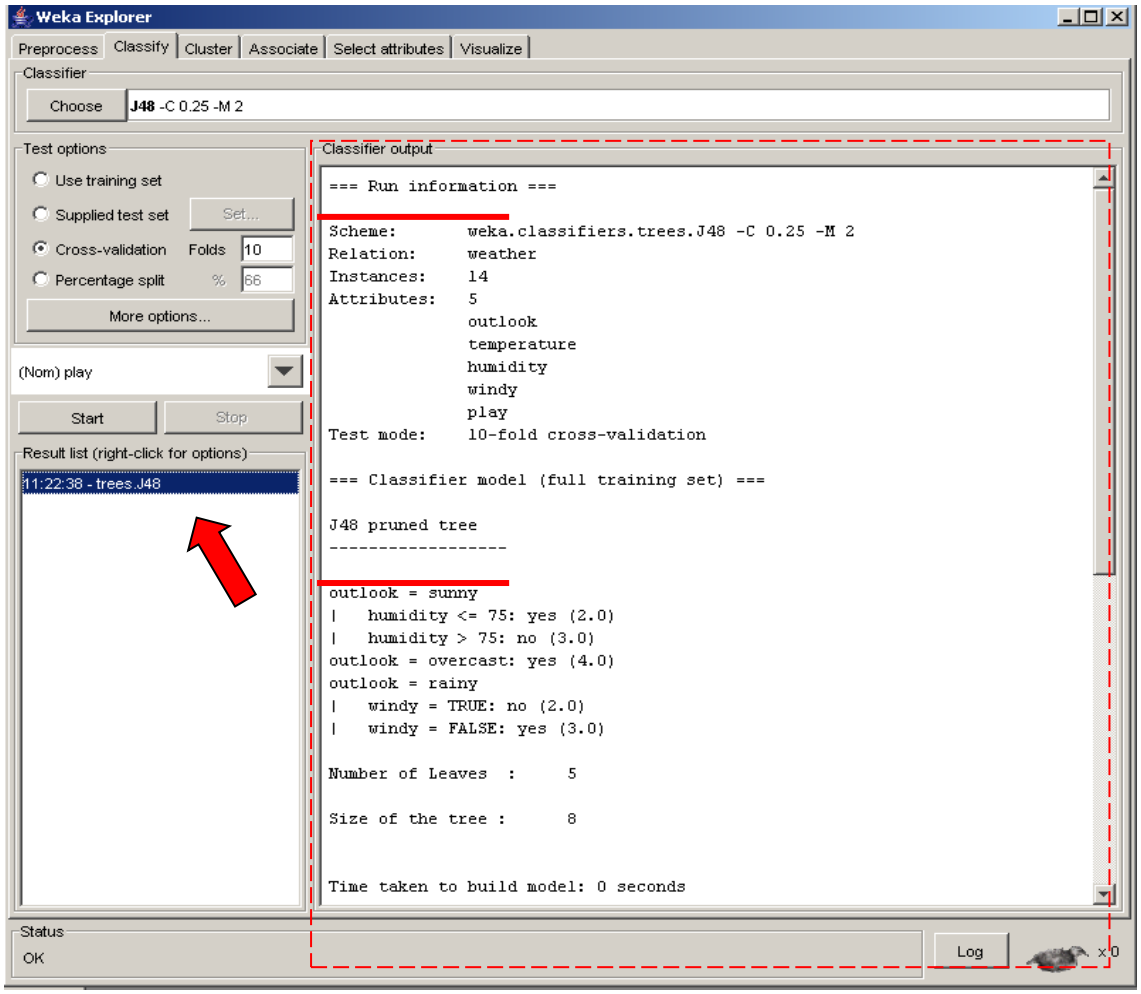

Figura 16 Salida del Clasificador J4.8

En el ejemplo se desea predecir la posibilidad de jugar tenis (atributo PLAY) de acuerdo a las condiciones climáticas (atributos OUTLOOK, HUMIDITY y WINDY). Es básicamente lo que se necesita, saber si un SLA se va a cumplir (Yes o No) dados unos parámetros.

Una vez el clasificador se ha ejecutado *El texto de salida* se puede ver en la figura 17. En la sección **J4.8 pruned tree** se ve claramente que es un árbol de decisión. *Outlook* es la raíz del árbol y determina la primera decisión. En caso que este nublado siempre podríamos jugar tenis. El número en paréntesis en el final de cada hoja es el número de ejemplos en esa hoja. En la grafica dentro de la misma sección podemos ver que él árbol de decisión tiene muy buena respuesta en cuanto a la velocidad de la construcción del modelo.

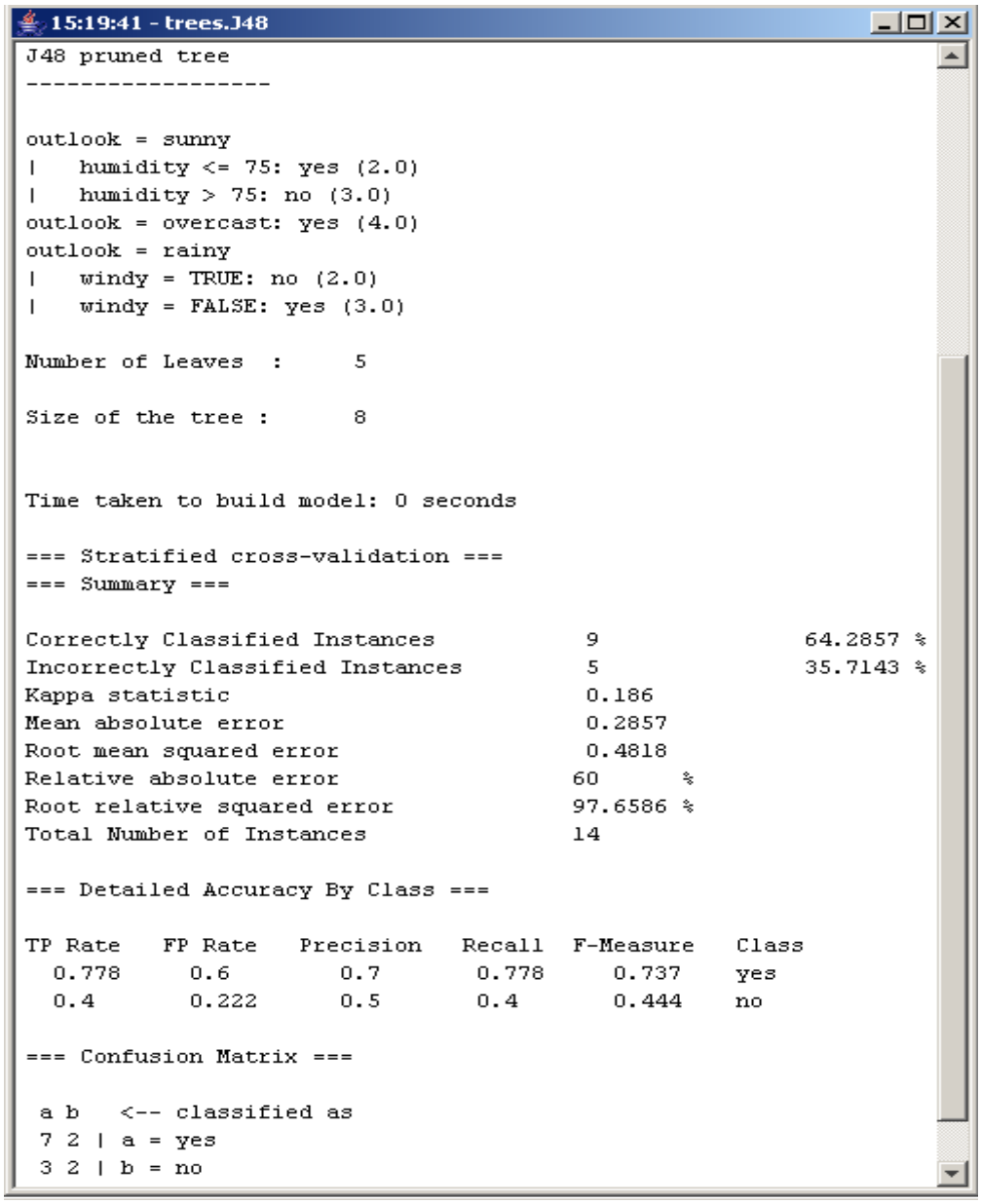

Figura 17 Texto de salida del clasificador

La sección **Stratified cross – validation** muestra información sobre la validación cruzada. La exactitud esta alrededor del 64%. La estadística kappa mide el acuerdo de predicción con la clase verdadera (1.0 significa completamente acordado). Los valores de error siguientes a la estadística kappa no son muy significativos para las tareas de clasificación, sin embargo para tareas de reqresión se pueden utilizar.

La matriz de confusión es comúnmente llamada *tabla de contingencia*. En el ejemplo se tienen dos clases, y por consiguiente una matriz de confusión de 2x2. El número de

instancias correctamente clasificadas es la suma de la diagonal en la matriz; todas las otras son incorrectamente clasificadas.

Dentro de la sección **Detailed Accuracy By Class** se tienen varios ítems que informan sobre la precisión por cada clase. El item *True Positive (TP)* es la proporción de ejemplos que son clasificados como clase *x*, entre todos los ejemplos que tienen clase *x.* Es equivalente a *Recall.* En la matriz de confusión, es el elemento de la diagonal dividido por la suma de la fila relevante, por ejemplo 7/(7+2)=0.778 para la clase *Yes* y 2/(3+2)=0.4 para la clase *No* para nuestra matriz.

El item *False Positive (FP)* es la proporción de ejemplos que son clasificados como clase *x*, pero pertenece a una diferente clase, entre todos los ejemplos que no son de clase *x.* En la matriz, es la suma de la columna de clase *x* menos el elemento de la diagonal, dividido por la suma de todas las otras clases, por ejemplo 3/5=0.6 para la clase *Yes* y 2/9=0.222 para la clase *No.*

La P*recisión* es la proporción de los ejemplos que correctamente tiene clase *x* entre todos esos que son clasificados como clase *x.* En la matriz, es el elemento de la diagonal dividido por la suma de la columna relevante, por ejemplo 7/(7+3)=0.7 para la clase *Yes* y 2/(2+2)=0.5 para la clase *No.*

La *F-Measure* es simplemente 2*Precision*Recall/(Precision+Recall), una medida que combina precision y recall.

Todas estas medidas (ítems) son útiles para comparar clasificadores.

4.4.5.1 La lista de resultado

WEKA tiene varias opciones para ver la salida del clasificador, en el punto anterior se mencionó la salida texto pero también cuenta con salidas graficas para un mejor entendimiento del clasificador. La mejor gráfica que representa la salida del algoritmo de árbol de decisión es **Visualize tree.** *Visualize tree* trae una representación gráfica de la estructura del modelo en árbol. La siguiente gráfica muestra el *Arbol* como resultado del ejemplo anterior.

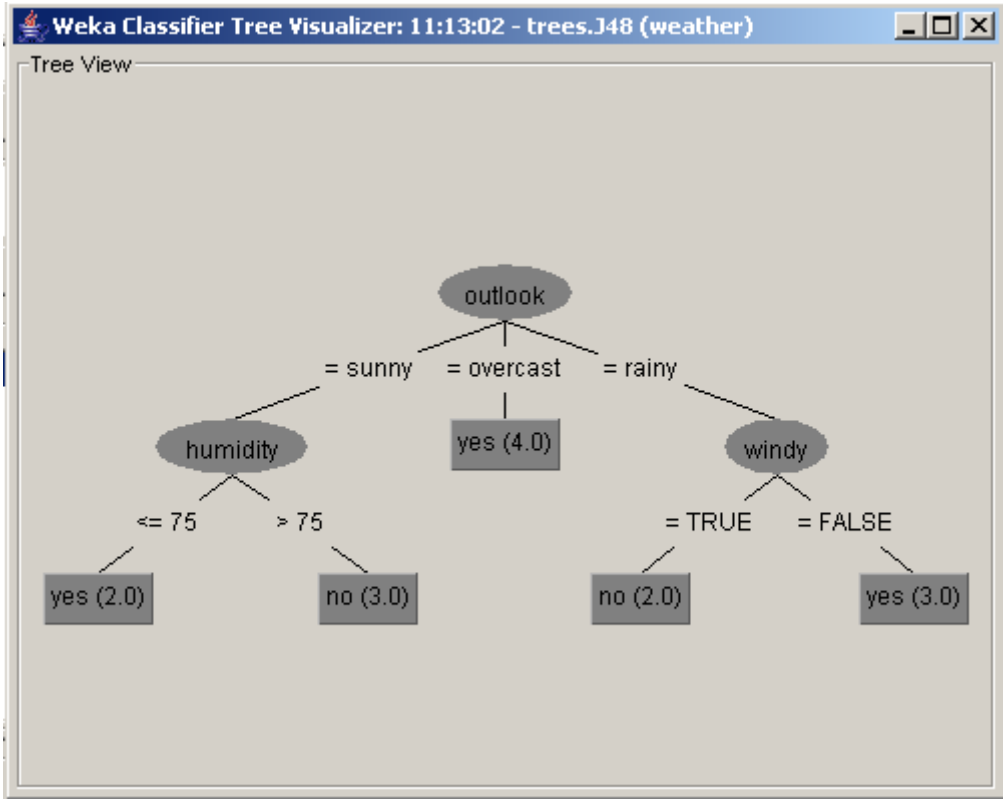

Figura 18 Visualización en árbol

Las “hojas” (representadas por rectángulos) son la terminación del árbol y toma los valores nominales de la variable de predicción. Los óvalos son las dimensiones y pueden tomar valores reales o nominales.

El ejemplo busca predecir las posibilidades de jugar tenis y para ello utiliza una variable que toma dos valores (Yes o No); en la figura son los rectángulos que muestra el valor y entre paréntesis el número de instancias del atributo para cada “rama” (bifurcaciones que puede tomar el árbol según los atributos). La raíz del árbol es determinada por el algoritmo y los datos de cada atributo, para el ejemplo es seleccionado el atributo Outlook que puede tomar tres valores Sunny, Overcas o Rainy,, cada uno de estos valores representa una rama del árbol.

La figura muestra que siempre se puede jugar tenis mientras que Outlook sea igual a Overcast. Si Outlook toma cualquiera de las otras dos opciones debe considerarse otra condición para determinar si se puede jugar o no.

Referencias

[i] Richard Hightower. Jakarta Struts Live, 2004

[ii] Sun Microsystems. Java Management Extensions SNMP Manager API

[iii] Oscar M. Caicedo. Desarrollo de Aplicaciones Web J2EE, 2003

[iv] Sione Palu. The Use of Java in Machine Learning, Articulo de Diciembre 2002.

[v] Universidad de Waikato Proyecto Weka http://www.cs.waikato.ac.nz/~ml/index.html

Bibliografía

1. *A Characterization of Data Mining Technologies and Processes* (Parsaye, K. The Journal of Data Warehousing. Otoño 1997).
2. *Data Mining and Knowledge Discovery* (Brand, E. and Gerritsen, R. DBMS Online. Febrero 1998).