8.DISEÑO EN JAVA DE LA LIBRERÍA PARA EL ACCESO Y MONITORIZACIÓN DE LA BASE DE DATOS: MARTEALERT.

8.1.Introducción.

El objetivo es diseñar una librería con métodos y clases que provean de soporte para visualizar, agrupar, gestionar, ordenar, filtrar y obtener informes de mensajes, mostrándolos en tiempo real, desde una fecha, o agrupados por parejas orígen-destino. En definitiva se trata de reproducir una aplicación similar a la herramienta Prewikka pero que sea capaz de subsanar los problemas encontrados dicha interfaz gráfica de Prelude. Con una interfaz en Java se puede conseguir una velocidad de procesamiento superior a la de una interfaz web, además de proveer de la estructura para futuras ampliaciones reutilizando código gracias a la herencia de clases.

La información obtenida de la base de datos se guardará en un objeto ***jTable*** que estará formado por un vector bidimensional para los datos y otro vector de cadenas de caracteres para los nombres de las columnas.

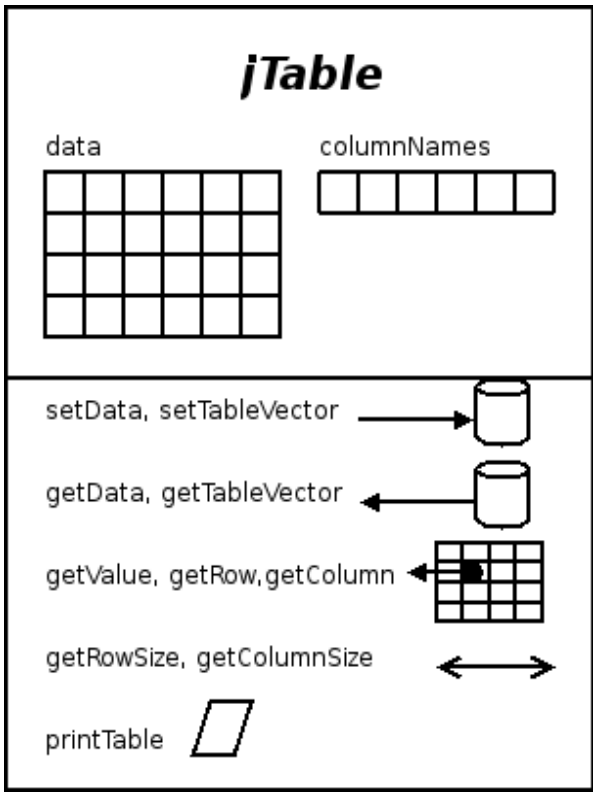


Figura 37.- Datos y métodos principales de la clase jTable.

El constructor podrá no tener parámetros, creando los vectores *data* y *columnNames* vacíos, o inicializándolos si se le pasa a dicho constructor la conexión y el comando SQL en formato cadena de caracteres. La conexión será un objeto *Connection* y el comando SQL es el que obtiene la tabla que queremos almacenar en *data*.

Usaremos el entorno de programación Netbeans 4.1, con librería para postgresql. Ésta se encuentra en formato ".jar" en http://turbosql.nihonsoft.org/index.html.

8.2.JDBC.

JDBC es el acrónimo de Java Database Connectivity, un API[26] que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java independientemente del sistema de operación donde se ejecute o de la base de datos a la cual se accede utilizando el lenguaje SQL del modelo de base de datos que se utilice.

La API JDBC está formada por el núcleo JDBC y la API de extensión. El núcleo define principalmente las interfaces estándar para realizar las siguientes funciones:

- *Crear una conexión con la base de datos*.
- *Crear "statements"*[27]
- *Acceder al conjunto de resultados.*
- *Realizar peticiones a la base de datos y devolver meta-datos como el nombre de las columnas, longitud en columnas de la tabla, etc.*

Las clases e interfaces del núcleo se definen en el paquete java.sql y están disponibles con la versión 2 de la plataforma Java, edición estándar, J2SE. La API de extensión define interfaces más sofisticadas para manejar recursos, transacciones distribuidas, factores de la conexión, etc. Estas clases pertenecen al paquete javax.sql y está disponible para la edición *enterprise* J2EE.

8.2.1.Uso del driver PostgreSQL JDBC.

La API JDBC define interfaces sólo para los objetos usados para ejecutar varias tareas relacionadas con bases de datos, como abrir y cerrar conexiones, ejecutar *statements* SQL, y devolver los resultados. No obstante, aunque podamos utilizar estos objetos, no se provee de clases que implementen dichas tareas. Las implementaciones software se llaman *JDBC drivers*, las cuales transforman las llamadas JDBC estándares a la llamada del gestor de recursos externo de la API.

En la figura 37 se muestra como un cliente de base de datos escrito en Java accede a un gestor de recursos externo usando la API JDBC y el driver JDBC.

[26] Interfaz de programación de aplicación, acrónimo de Application Programming Interface.

[27] Por cada conexión se podrán crear distintos *statements* o "subconexiones" a la base de datos. Una conexión viene determinada por la dirección de la máquina destino, el puerto, el protocol y el nombre de la base de datos. Un *statement* se creará petición que se haga en esa conexión.

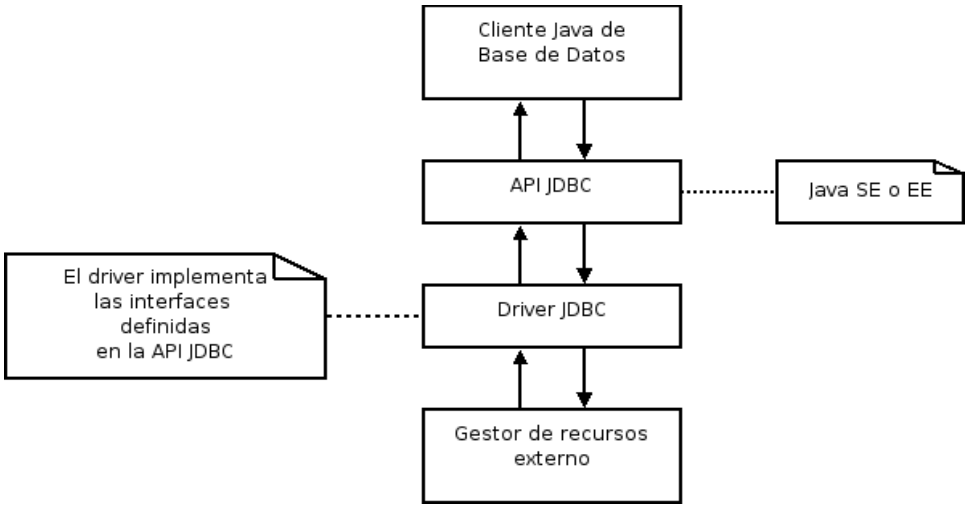


Figura 38.- Definición de la API JDBC y capas de implementación

Dependiendo del mecanismo de implementación, los drivers JDBC se pueden clasificar ampliamente en cuatro tipos:

- *Tipo 1*: Son drivers que implementan la API JDBC como capa más alta de una API de bajo nivel como ODBC. Estos drivers no son portables normalmente porque dependen de las librerías nativas.
- *Tipo 2*: Están escritos en una mezcla de Java y código nativo. Son drivers que usan APIs de fabricantes específicos para acceder a los datos. Tampoco son portables, porque dependen del código nativo, pero son muy eficientes para bases de datos locales.
- *Tipo 3*: Están escritos completamente en Java en el lado cliente, el cual se comunica con un servidor intermedio para acceder a los datos (middleware). Las llamadas al servidor intermedio son independientes de la base de datos, aunque luego el servidor haga llamadas específicas para acceder a los datos.
- *Tipo 4*: Están escritos puramente en Java. Imlementan las interfaces JDBC y traducen las llamadas específicas JDBC a llamadas de acceso a datos específicas de un fabricante.

De estos cuatro tipos, aunque PostgreSQL los permite todos, el más aconsejable es el último y el usado en esta librería *martealert*, ya que permiten definir un protocolo propio de red y transferencia de datos para el gestor de recursos, además que implementa clases con métodos de fácil y específico manejo para acceder a datos concretos, con lo que la eficiencia aumenta.

8.2.2.Instalación.

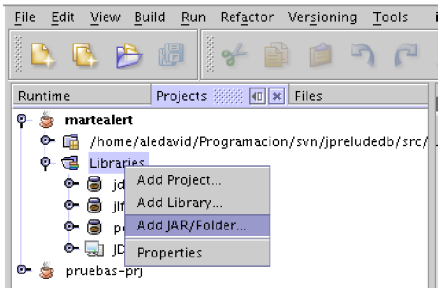
Los drivers para PostgreSQL pueden descargarse de http://jdbc.postgresql.org. Hay que tener cuidado a la hora de seleccionar la combinación apropiada del driver JDBC, JRE (Java Runtime Environment) y versión de PostgreSQL. Normalmente se tratará del JDBC-3.

Desde Gentoo Linux es suficiente con ejecutar lo siguiente para instalar jdbc sobre JDK.

```
# emerge jdbc3-postgresql
```

Este comando no sólo descarga el paquete sino que compila las fuentes y las instala en nuestro sistema.

Una vez que se tenga el fichero del driver (postgresql-8.0.309.jdbc3.jar o algo similar) hay que añadirlo al CLASSPATH, alterando el entorno por medio de una copia del fichero en el subdirectorio /lib/ext de JRE, o bien usando la opción **-cp** del comando java cuando se ejecute el programa. También se puede descargar de la dirección anterior la fuente de JDBC.



Desde NetBeans, se instala el programa descargado pinchando en la pestaña de proyectos *Projects*, *Libraries*, botón derecho y añadir librería o fichero JAR.

8.3.Estructura de clases.

La estructura de clases se basará en una clase abstracta (*martealertTable*) que heredará de *jTable* y dependerá de *martealertHashFilter*, ya que el objeto *martealertTable* está formado por un filtro *martealertHashFilter*. A continuación el diagrama con las relaciones de dependencia y herencia.

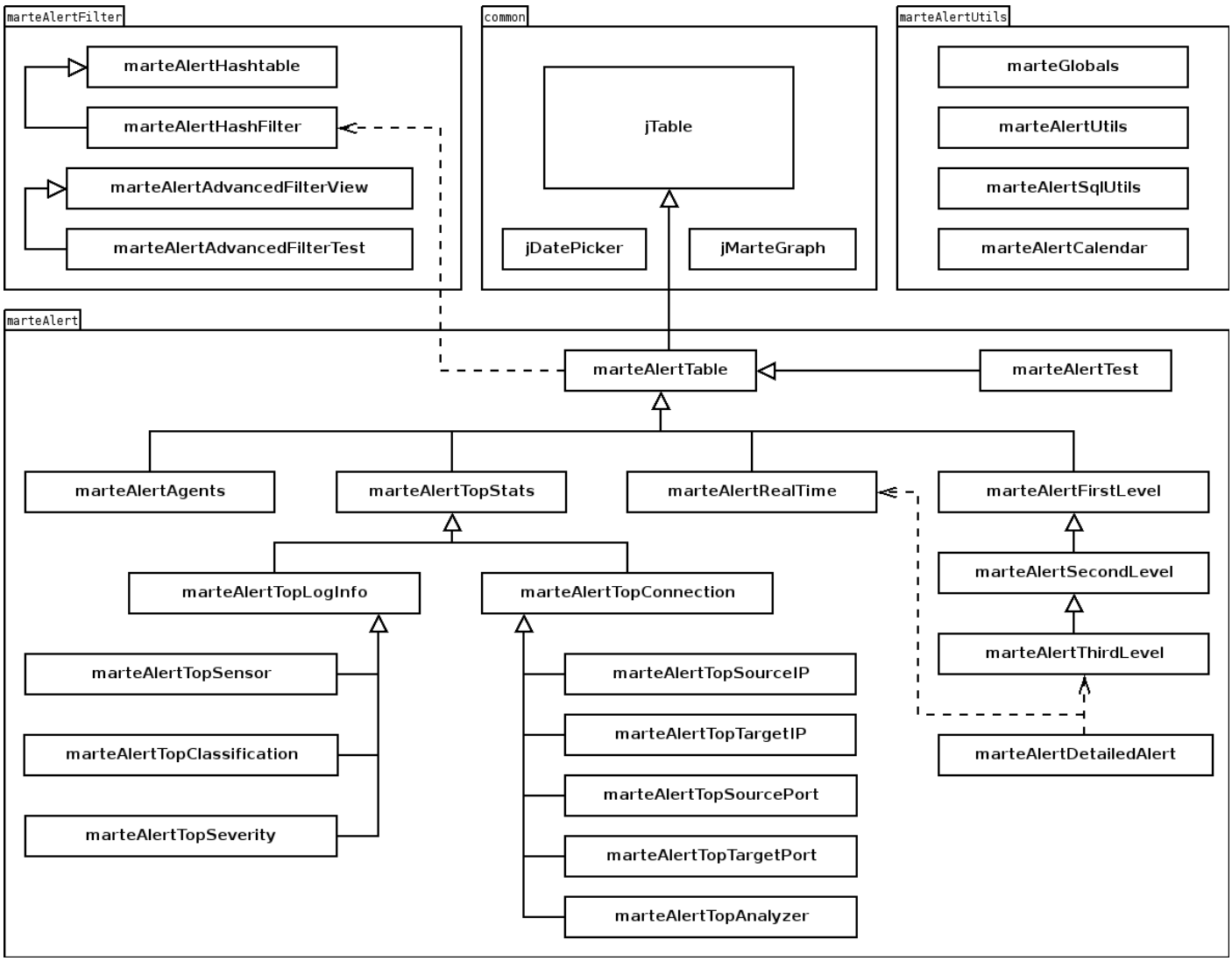


Figura 39.- Diagrama de clases y relaciones de la librería *martealert*.

8.4.Paquete *common*.

Está formado por las clases que pueden ser reutilizables para ampliaciones de este proyecto o módulos que lo complementen, sin necesidad de replicar una misma clase.

8.4.1.Clase *jTable*.

Como ya se comentó con en la introducción, está formada por un vector bidimensional para los datos y uno unidimensional para los nombres de las columnas. El motivo por el que *jTable* tiene esta estructura es por compatibilidad con las clases gráficas existentes, a cuyos métodos deben pasarse objetos *jTable* o derivados para que se generen los gráficos sectoriales y gráficas X-Y (véase el apartado de capturas de MarteAlert).

El paquete *common* podrá ser accesible desde varios proyectos y por lo tanto cualquier paquete que se diseñe en el futuro que opere con bases de datos podrá implementar una clase que herede de jTable la estructura y métodos básicos.

- *public void* ***setDataVector****(Vector data, Vector columnNames):* Almacena en el objeto *this* (desde el que se invoca) los valores pasados como parámetros.
- *public void* ***setData****(Vector data)*: Almacena el vector *data* en el *this.data*.
- *public void* ***setColumnNames****(Vector columnNames*): Almacena en el objeto *this* el vector con los nombres de las columnas que se pasa como parámetro.
- *public void* ***setTableVector****(Connection con, String sql)*: Almacena en el objeto *this* el resultado de la consulta cuya sintaxis se pasa cocmo la cadena *sql*.
- *public synchronized* ***setValue****(int row, int column, Object value)*:
- *public Vector* ***getData****()*: Devuelve el vector bidimensional de datos del objeto *this*.
- *public Vector* ***getColumnNames****()*: Obtiene el vector con los nombres de las columnas de *this.*
- *public Object* ***getValue****(int row, int column):* Devuelve el objeto de coordenadas (row, column) del vector bidimensional *data*.
- *public Object* ***getValue****(int row, int column, Object defaultIfNull)*: Devuelve el objeto de coordenadas (row, column) del objeto actual *this*. Se le puede especificar el objeto que devuelva por defecto si el valor devuelto es NULL.
- *public Object* ***getValue****(int row, int column, Object ob, Object defaultIfNull)*: Devuelve el objeto de coordenadas (row, column) del objeto actual *this*, y devuelve el valor *defaultIfNull* si el valor encontrado es *ob*. Es decir, este método equivale al anterior si *ob* es NULL.
- *public static martealertTable* ***getTableVector****(Connection con, String sql)*: Obtiene el objeto *martealertTable* inicializado con el comando SQL que se le pasa como parámetro. Se lanza la excepción *java.sql.SQLException*.
- *public Vector* ***getRow****(int row)*: Devuelve el vector fila correspondiente al índice especificado.
- *public Vector* ***getColumn****(int column)*: Devuelve la columna de la tabla cuyo índice se pasa como parámetro.
- *public int* ***getRowSize****()*: Devuelve el tamaño de la fila, es decir, el número de columnas de la tabla.

- *public int* ***getColumnSize()***: Devuelve el tamaño de la columna, es decir, el número de filas que tiene la tabla.
- *public int* ***getNamedColumnNumber****(String columnname)*: Obtiene el índice de la columna que tiene como nombre el parámetro que se le pasa o -1 si no encuentra la ocurrencia.
- *public void printTable()*: Imprime la tabla en formato texto. Sólo se utilizará para depuración y testeo, ya que para la impresión de las tablas se hará uso de clases gráficas hechas.

8.4.2.Clase *jDatePicker.*

La clase *jDatePicker* se usa para generar objetos gráficos de tipo calendario. Sus métodos son utilizados por la clase *martealertAdvancedFilterView* para generar los filtros avanzados. En el paquete *martealertFilter* se explicará con más detalle.

8.4.3.Clases gráficas.

Aunque se especifica como una única clase en la figura 39, se trata de un conjunto de clases que generan gráficos sectoriales, gráficos lineales, tablas con colores, ventanas etc. Algunas de ellas se encuentran en el paquete *common* y otras conforman un paquete propio de gráficos. En el esquema anterior se han incluido en la clase common porque se trata de código que será reutilizable y su diseño no es la finalidad de este proyecto. Por tanto no se trata de clases que se hayan rediseñado para este proyecto, sino clases hechas.

El estudio de estas clases gráficas sirve, entre otras cosas, para diseñar objetos que se le puedan pasar como parámetros a los métodos que representan los dichos gráficos. A continuación se muestra un gráfico sectorial de tipos de mensajes.

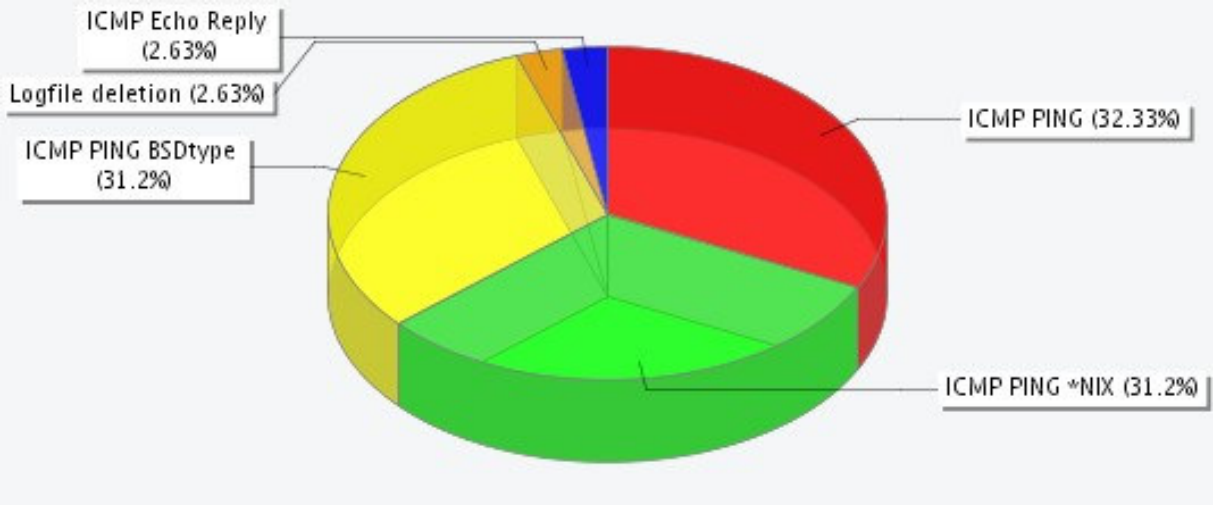


Figura 40.- Gráfico sectorial.

8.5.Paquete *martealert.*

8.5.1.Clase *martealertTable.*

Es la clase de la que heredan todas las subclases del paquete *martealert*. A su vez, hereda de *jTable* la estructura a la que se le suma el objeto *martealertHashFilter* y la definición de los bloques estáticos que inicializan las tablas "Hash"([28]) globales para todo el paquete.

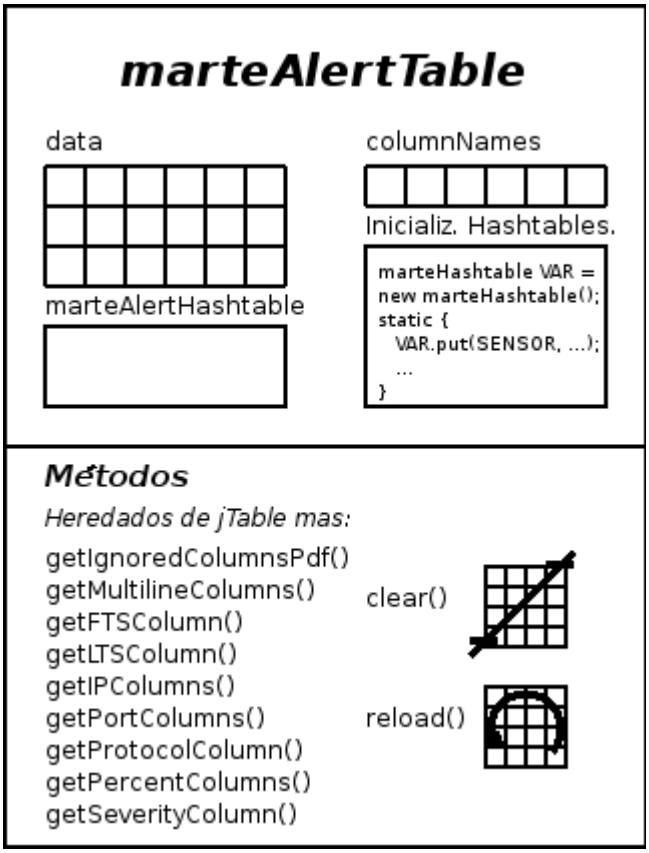


Figura 41.- Esquema de la clase *martealertTable*.

El constructor permite que se inicialice con el resultado del comando SQL que se especifica como cadena de caracteres en el segundo parámetro: *sqlcmd.*

Los métodos nuevos que se incorporan son la mayoría abstractos ya que cada una de las subclases *martealertTopStats, martealertFirstLevel, martealertAgents* y *martealertRealTime* deberán implementar estos métodos. A continuación una breve explicación de los métodos:

- *public void* ***clear()***: Elimina los datos del objeto *martealertTable*.
- *public void* ***reload()***: Dado que el objeto *martealertTable* se calcula con un *martealertHashFilter* concreto, éste método permite a este objeto y todas los que hereden de él recargar los datos conservando los nombres de los campos.
- *public int* ***getFTSColumn()***: Obtiene el índice de la columna de la fecha del FTS.
- *public int* ***getLTSColumn()***: Obtiene el índice de la columna de la fecha del LTS.

[28] Una tabla "hash" es un tipo de vector cuyo índice puede ser cualquier cadena de caracteres. Esto hace que la búsqueda sea muy sencilla a costa de un aumento en el consumo de memoria.

- *public int* ***getIgnoredColumnsPdf()***: Obtiene el índice de la columna que, en caso de generar un informe en PDF, no se mostrará en dicho documento.
- *public int[]* ***getMultilineColumns()***: Obtiene un array con las índices de los campos (columnas) que pueden almacenar datos compuestos (metafilas) para una misma fila del vector *data*. Véase el ejemplo de *martealertFirstLevel* y los campos del tipo de mensaje y el sensor.
- *public int[]* ***getProtocolColumns()***: Obtiene el índice de la columna de los protocolos, los cuales se podrán resolver desde otras tablas, bases de datos etc. Así se obtendrá TCP desde el número 6, UDP desde el 17 etc.
- *public int[]* ***getPortColumns()****:* Obtiene los datos análogos a los del método anterior pero para los puertos orígen y destino.
- *public int[]* ***getIPColumns()***: Obtiene los índices de las columnas de direcciones IP, las cuales se podrán resolver desde una base de datos local o por de resolución inversa.
- *public int[]* ***getPercentColumns()***: Obtiene un array de enteros con los índices de las columnas que mostrarán un valor numérico en tanto por ciento, útil para las librerías gráficas.
- *public int* ***getSeverityColumn()***: Obtiene el índice de la columna del campo "severity".

8.5.2.Clase *martealertTopStats.*

Clase que hereda de *martealertTable* y se particulariza para una tabla de estadísticas formada por el nombre, el número de alertas, el tanto por ciento de este tipo de alerta respecto del total mostrado en la tabla, la fechas de la primera y última vez visto del grupo.

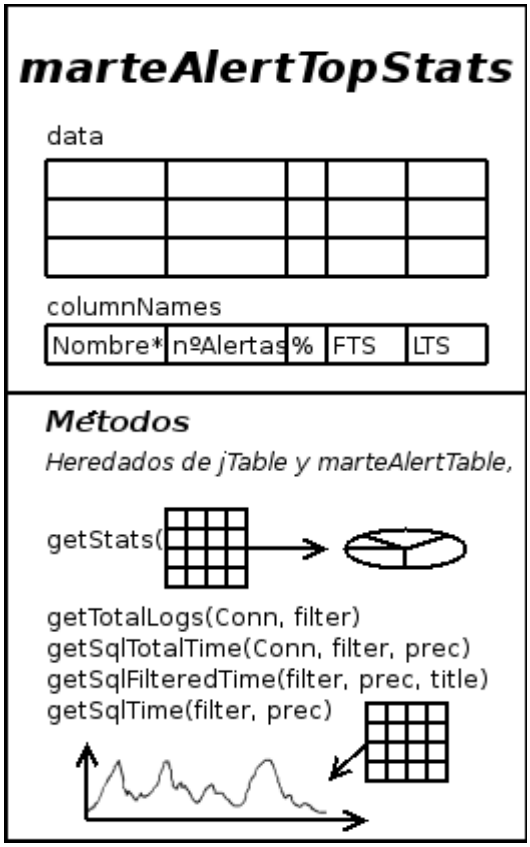


Figura 42.- Esquema de la clase *martealertTopStats*.

En la figura 41 se muestra el tipo de gráfico que se puede generar con los datos obtenidos por los métodos que aparecen. Sólo se han mostrado los públicos, aunque para la implementación con reutilización de código se han definido muchos privados. La columna "nombre" se ha marcado con un asterisco porque en el caso de las estadisticas cuyo valor pueda ser resuelto se pondrán por duplicado, es decir, para el puerto se duplicará la columna número de puerto de modo que aparezca en una resuelto el nombre "ssh" y en otra el número "22".

- *public int* ***getTotalLogs****(Connection con, martealertHashFilter hf)*: Obtiene el número total de alertas recibidas para el rango de tiempo especificado en *hf.time*, filtrado por el resto de campos del filtro que no sean cadenas vacías. Es el valor por el que se divide para calcular los tantos por ciento.

- *public void* ***reload****(Connection con, martealertHashFilter filter)*: Borra los datos del objeto *this* y asigna los datos hallados por *getStats*.

- *public void* ***getStats****(Connection con, martealertHashFilter filter)*: Obtiene la tabla de estadísticas para cada tipo de campo entre los que están cada una de las subclases que heredan de *martealertTopClassification* y *martealertTopLogInfo* y almacena los datos en los vectores *data* y *columnNames*. A continuación, véase un ejemplo de la tabla que representa la librería gráfica con estos datos([29]):

Tipo de mensaje	Alertas	%	Desde	Hasta
Bogus IP address detected	89	58.6%	01-jun-2006	01-jun-2006
SUDO Command Executed	31	20.4%	01-jun-2006	01-jun-2006
New ARP address detected	10	6.6%	01-jun-2006	01-jun-2006
User login successful	5	3.3%	01-jun-2006	01-jun-2006
ICMP Echo Reply	3	2.0%	01-jun-2006	01-jun-2006
ICMP PING	3	2.0%	01-jun-2006	01-jun-2006
ICMP PING BSDtype	3	2.0%	01-jun-2006	01-jun-2006
ICMP PING *NIX	3	2.0%	01-jun-2006	01-jun-2006
Admin login failed	2	1.3%	01-jun-2006	01-jun-2006
Admin login successful	2	1.3%	01-jun-2006	01-jun-2006
Logfile deletion	1	0.7%	01-jun-200	-jun-2006

15:18:48

Figura 43.- Tabla representada con los datos calculados por *getStats*.

- *abstract protected String* ***getSqlVars****(int tot)*: Método al que se le pasa el entero recibido por *getTotalLogs* y devuelve los nombres de los campos que se desea que aparezcan. Al ser abstracta, cada clase que herede de ésta, devolverá sus variables correspondientes. Por ejemplo, para el puerto, se devolverá "prelude_service.port, max(prelude_createtime.time)...", ya que la implementación de dicho método se hace en la clase *martealertTopTargetPort*.

- *abstract protected String* ***getSqlGroupedBy****()*: Método que devuelve las variables por las que se agrupa la consulta. Al igual que la anterior es "protected" ya que sólo se invoca desde esta clase.

- *abstract public String* ***getField****()*: Método que devuelve el campo cuyas estadísticas se calculan con *getStats* (sensor, ip origen, ip destino, sonda, tipo de mensaje, gravedad, aplicación y protocolo)

- *abstract protected String* ***getSqlOrderBy****()*: Método que devuelve las variables por las que se ordena. Para el caso actual se ordenará por el número de alertas o lo que es equivalente, por el tanto por ciento.

[29] Los campos "Desde" y "Hasta" corresponden a los valores de FTS y LTS, respectivamente. Se muestra en principio la fecha pero al detener el ratón sale en recuadro azul la hora.

- *protected String* ***calcPercent****(Connection con, int total)*: Método que calcula el tanto por ciento dada una conexión y el total de alertas calculado.

- *public String* ***getCommand****(martealertHashFilter filtro)*: Compone y devuelve un comando SQL para obtener un conjunto de datos determinados a partir de un filtro determinado. Este conjunto de datos lo condicionan los métodos *getSqlVars, getSqlGroupedBy, getSqlCommand* y *getSqlOrderBy*. Cada uno de los mencionados componen las variables a mostrar (campos) acordes a la clase en la que estén, las variables por las que se agrupa, las tablas por las que se hace producto o "join", y las variables por las que se ordena la tabla devuelta. La composición del comando completo se devuelve en formato cadena de caracteres.

- *protected static String* ***getPrecisionVar****(int precision, String title)*: Método que devuelve la variable en SQL para obtener la lista de las componentes de abscisas (tiempo) discretizada según el entero *precision* que será una de las constantes de la interfaz *martealertTopInterface*. La cadena *title* será el título/subtítulo que aparecerá al pie de la gráfica (ver figuras 43 y 44).

- *public String* ***getSqlTotalTime****(Connection con, martealertHashFilter filter, int precision)*: Método que devuelve el comando SQL necesario para generar un gráfico temporal en el intervalo especificado por el filtro, especificando la precisión del gráfico discreto. A continuación un ejemplo del gráfico que se representa con el comando devuelto por este método:

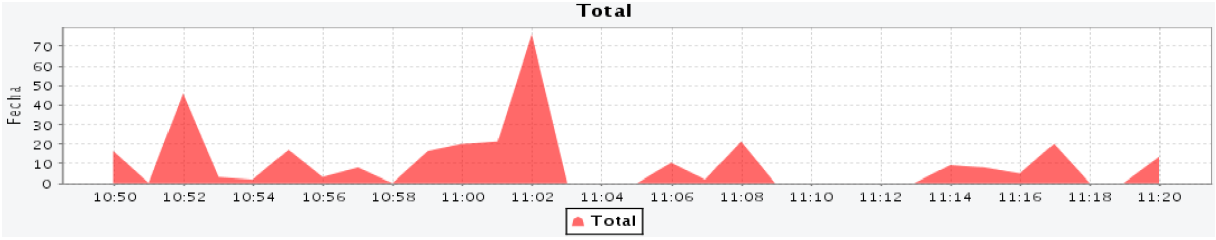


Figura 44.- Representación temporal del total de alertas sin filtrar.

- *public String* ***getSqlFilteredTime****(martealertHashFilter filter, int precision)*: Método que devuelve el comando necesario para que se genere un gráfico como el anterior para filtros no nulos. Por tanto, para un filtro que tenga todas las componentes vacías (exceptuando la de tiempo) se obtendrá el mismo gráfico que el anterior. Se trata de un par de columnas en la que la primera es la componente de abscisas (tiempo discretizado por la precisión) y la segunda es la ordenada (número de alertas).

- *public String[]* ***getSqlTime****(martealertHashFilter, int prec, int strtam)*: Método que devuelve un "array" de comandos SQL para la obtención de los gráficos temporales filtrados por cada componente. Por ejemplo, para el "top" IP origen, cada comando estará filtrado por una IP origen, tantas como se especifique en el entero *strtam*, siendo el valor por defecto 5. Este conjunto de comandos SQL lo interpreta la librería gráfica para generar un gráfico apilado temporal como el de la figura 44:

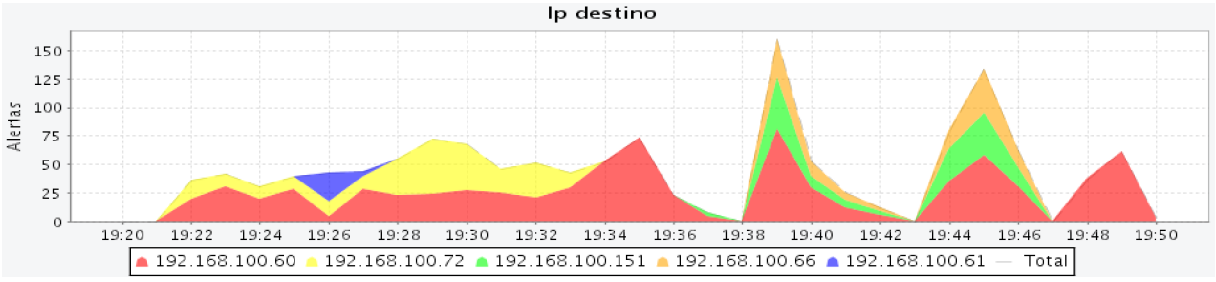


Figura 45.- Representación temporal del gráfico apilado de las cinco IP destino con mayor número de alertas generadas.

8.5.2.1. Clase *martealertTopLogInfo.*

Es una subclase abstracta de *martealertTopStats* y a la vez superclase de TopSensor, TopClassification y TopSeverity.

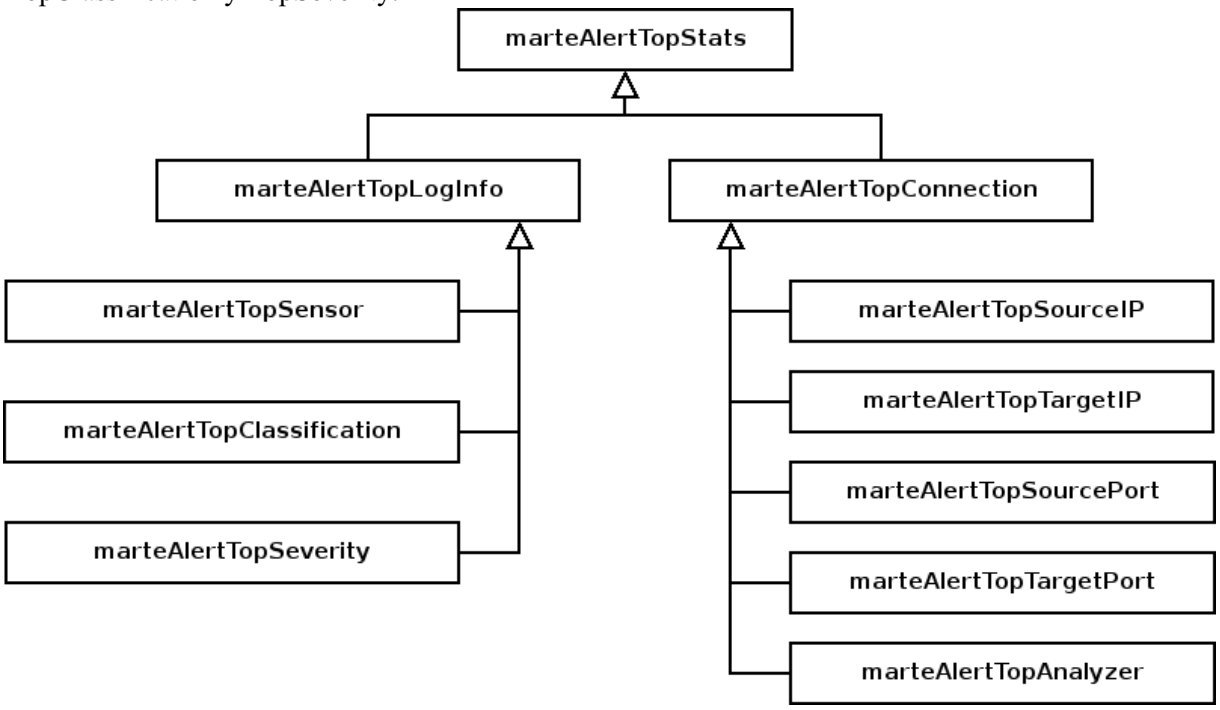


Figura 46.- Esquema de clases de *martealertTopStats*.

La diferenciación entre *martealertTopLogInfo* y *martealertTopConnection* estructuralmente se debe a una duplicación de la primera columna en el caso del *topConnection*. *MarteAlertTopLogInfo* posee las subclases que aportan información sobre los mensajes, la cual no es de carácter numérico o dirección que se pueda resolver. Por tanto, posee cinco columnas como el ejemplo siguiente:

Tipo de mensaje	Alertas	%	Desde	Hasta
(spp_stream4) possibl...	79	36.6%	26-may-2006	08-jun-2006
SCAN Proxy Port 8080 ...	40	18.5%	07-jun-2006	07-jun-2006
SCAN UPnP service dis...	33	15.3%	26-may-2006	08-jun-2006
ICMP Echo Reply	21	9.7%	26-may-2006	03-jun-2006
ICMP PING	14	6.5%	26-may-2006	03-jun-2006
ICMP PING BSDtype	13	6.0%	26-may-2006	29-may-2006
ICMP PING *NIX	13	6.0%	26-may-2006	29-may-2006
(portscan) TCP Portscan	3	1.4%	26-may-2006	07-jun-2006

Figura 47.- Estructura de tabla con 5 columnas para las subclases de *martealertTopLogInfo*.

Esta clase implementa los métodos abstractos de *martealertTopStats* referentes a obtener la posición de una columna como *getAlertColumn, getPercentColumns, getFTSColumn, getLTSColumn*.

Además de los mencionados sigue declarando como abstracto el método *getField* para que se implemente en sus subclases.

8.5.2.1.1 Clase *martealertTopSensor.*

Son objetos instanciables que almacenan la tabla de los sensores, y permite a la librería gráfica representar el diagrama de sectores. Los cinco valores con más alertas son representados en el gráfico temporal, junto con el total en línea discontinua y el total filtrado.

A continuación se detalla en la figura 48 una captura de toda la información referente al sensor, aunque la aplicación permite el cruce de información seleccionando por ejemplo *topSensor* para los gráficos temporales y *topSourceIP* para la tabla y diagrama de sectores.

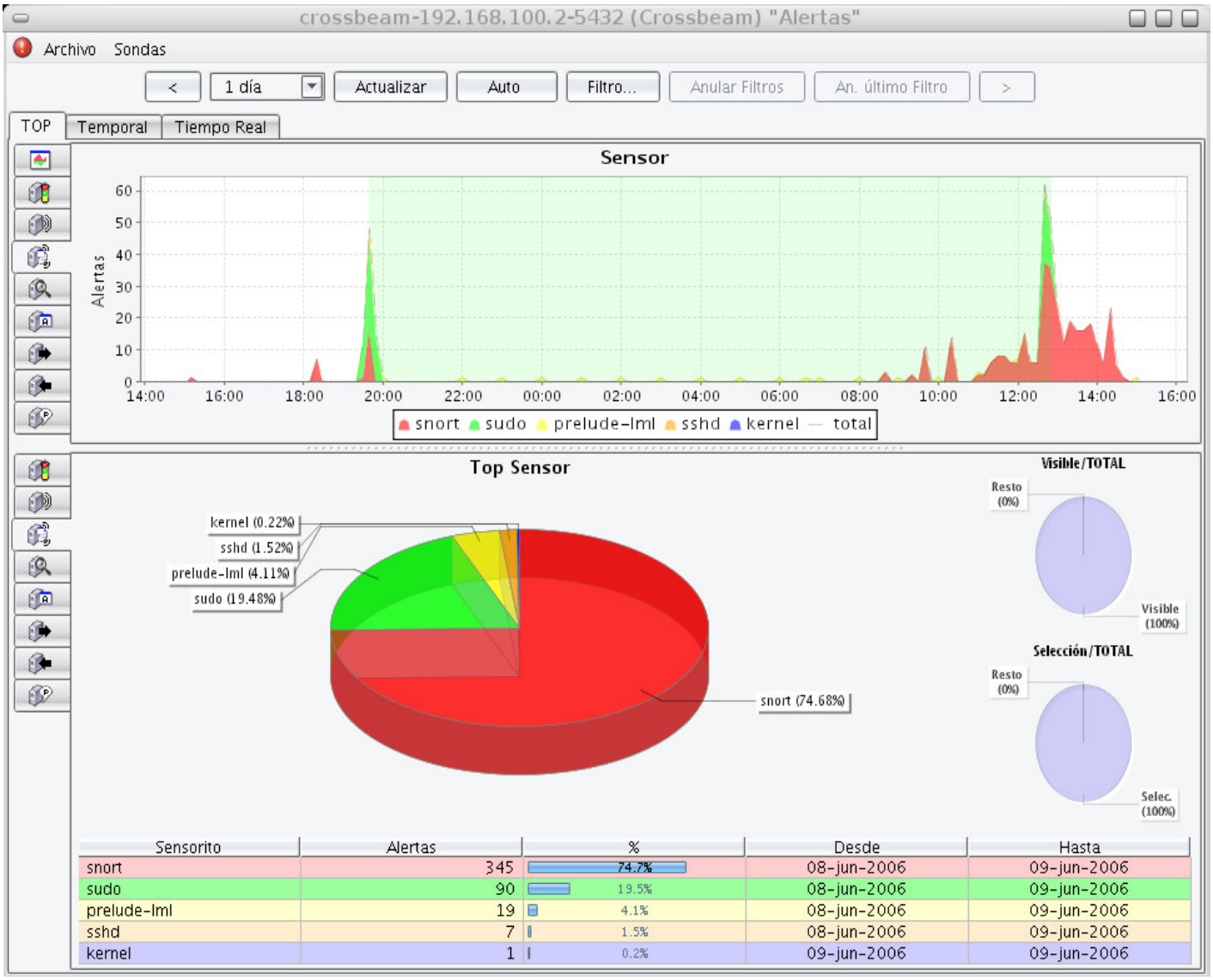
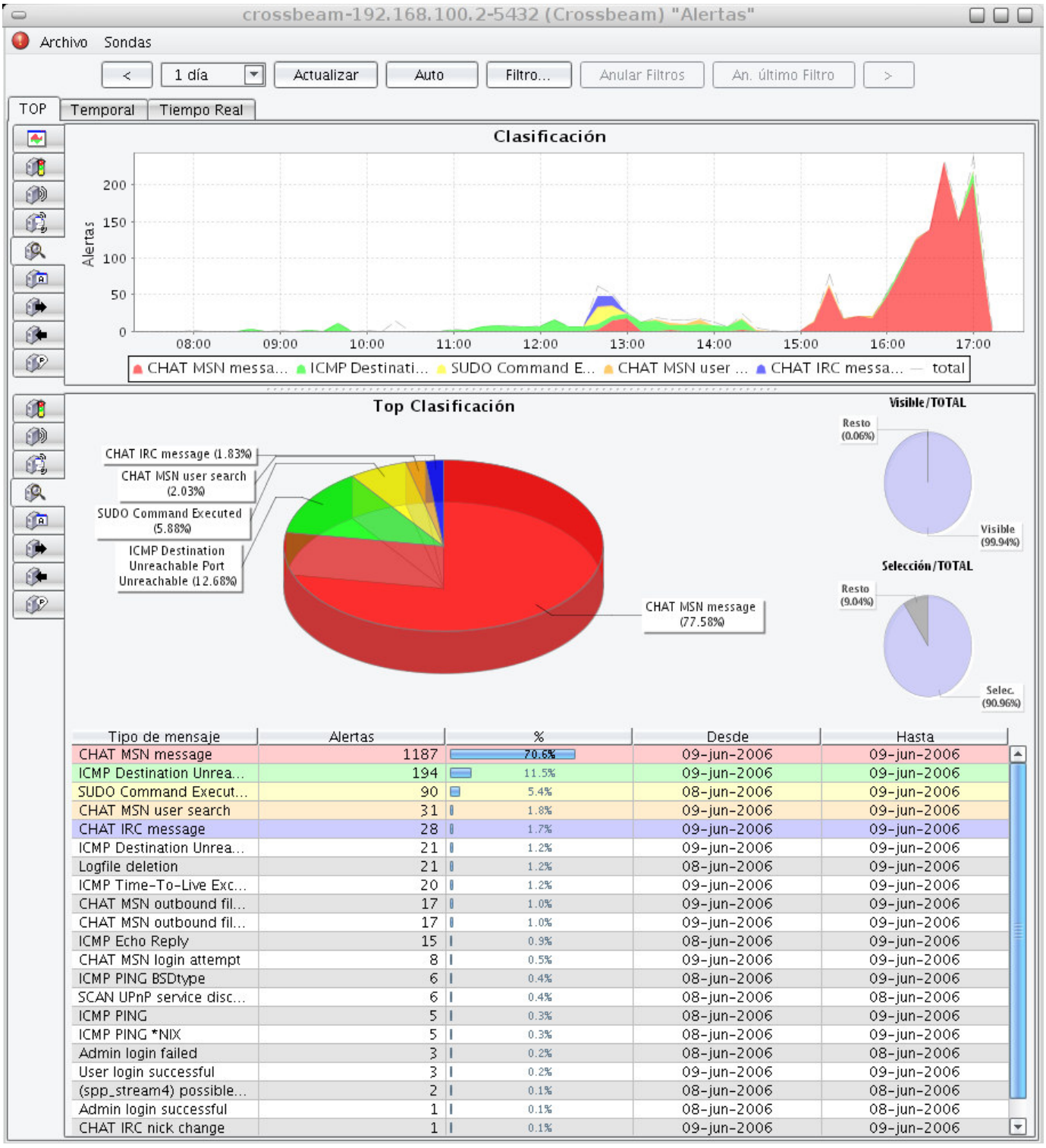


Figura 48.- Top Sensor.

Otra utilidad que permite la librería gráfica es mostrar en la gráfica temporal el intervalo que comprende [FTS, LTS] en el color de la línea seleccionada en la tabla. En el ejemplo se ha seleccionado "sudo", por lo que se muestra una franja verde en la gráfica temporal.

8.5.2.1.2 Clase *martealertTopClassification.*

Es la clase instanciable subclase de *martealertTopLogInfo* que contiene la información referente al tipo de mensaje o la clasificación. En la figura 49 se muestra cómo a partir de las 16:00 h. aproximadamente crece el número de alertas por mensajes de chat messenger sin seguridad.



Tipo de mensaje	Alertas	%	Desde	Hasta
CHAT MSN message	1187	70.6%	09-jun-2006	09-jun-2006
ICMP Destination Unrea...	194	11.5%	09-jun-2006	09-jun-2006
SUDO Command Execut...	90	5.4%	08-jun-2006	09-jun-2006
CHAT MSN user search	31	1.8%	09-jun-2006	09-jun-2006
CHAT IRC message	28	1.7%	09-jun-2006	09-jun-2006
ICMP Destination Unrea...	21	1.2%	09-jun-2006	09-jun-2006
Logfile deletion	21	1.2%	08-jun-2006	09-jun-2006
ICMP Time-To-Live Exc...	20	1.2%	09-jun-2006	09-jun-2006
CHAT MSN outbound fil...	17	1.0%	09-jun-2006	09-jun-2006
CHAT MSN outbound fil...	17	1.0%	09-jun-2006	09-jun-2006
ICMP Echo Reply	15	0.9%	08-jun-2006	09-jun-2006
CHAT MSN login attempt	8	0.5%	09-jun-2006	09-jun-2006
ICMP PING BSDtype	6	0.4%	08-jun-2006	09-jun-2006
SCAN UPnP service disc...	6	0.4%	08-jun-2006	08-jun-2006
ICMP PING	5	0.3%	08-jun-2006	09-jun-2006
ICMP PING *NIX	5	0.3%	08-jun-2006	09-jun-2006
Admin login failed	3	0.2%	08-jun-2006	08-jun-2006
User login successful	3	0.2%	09-jun-2006	09-jun-2006
(spp_stream4) possible...	2	0.1%	08-jun-2006	08-jun-2006
Admin login successful	1	0.1%	08-jun-2006	08-jun-2006
CHAT IRC nick change	1	0.1%	09-jun-2006	09-jun-2006

Figura 49.- Top clasificación.

8.5.2.1.3 Clase *martealertTopSeverity.*

Es la clase instanciable subclase de *martealertTopLogInfo* que contiene la información referente a la gravedad de los mensajes. El formato IDMEF define cuatro posibles valores para el campo *severity* que se almacena en la tabla *prelude_impact*. En la tabla 5 se resume una breve descripción de estos cuatro valores: *info, low, medium* y *high*. Estos cuatro valores se pueden observar en la figura 50.

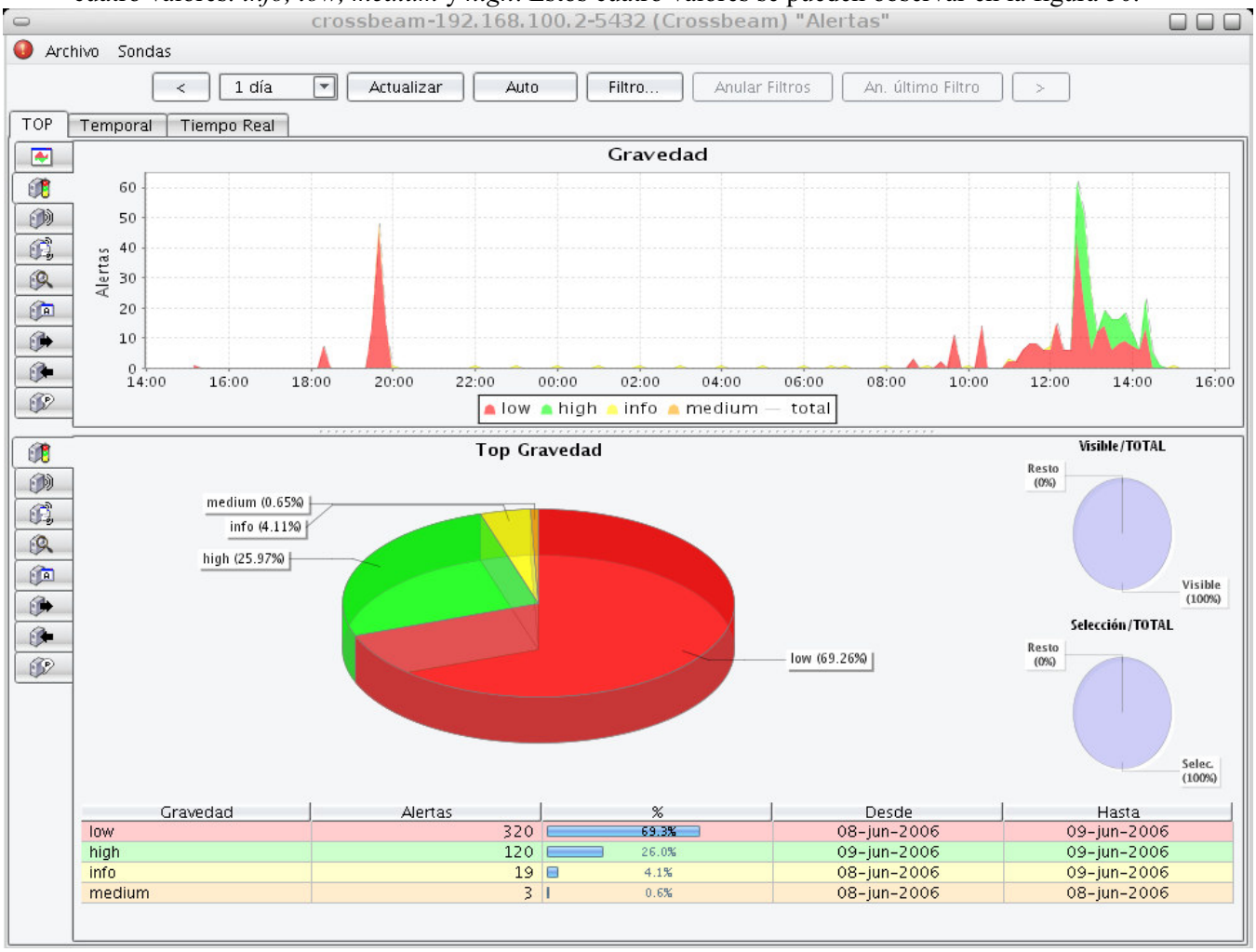


Figura 50.- Top gravedad.

De tipo high tendremos por ejemplo mensajes de chat, intentos de autenticación fallidos como administrador por ssh. De tipo medium serán intentos de autenticación pero como un usuario no privilegiado. Los low serán los mayoritarios como escaneos de puertos, mensajes de eco etc. Los de información muestran borrado de ficheros en rotación de logs, por ejemplo.

8.5.2.2. Clase *martealertTopConnection.*

Es una clase abstracta, subclase de *martealertTopStats*, que se encuentra al mismo nivel que la anteriormente desglosada *martealertTopLogInfo*, y es muy similar en cuanto a estructura, diferenciándose principalmente en que las subclases de *topConnection* tendrán 6 columnas. La columna equivalente al nombre se duplica para poder resolverse bien consultándose una base de datos, como en el caso de los puertos y protocolos, o por resolución inversa como para las direcciones IP origen, IP destino y sonda.

8.5.2.2.1 Clase *martealertTopSourceIP.*

Es la clase instanciable, subclase de *martealertTopConnection* que almacena las estadísticas por número de alertas y su distribución temporal según la dirección IP origen.

En la figura 51, a diferencia de las anteriores, se representa la gráfica temporal con una precisión mayor ya que el rango es menor: en este caso desde el instante actual hasta cinco horas atrás. La línea punteada muestra el total de alertas, el cual no se alcanza porque como se ha comentado anteriormente, se representan temporalmente las cinco entradas con más alertas de la tabla bajo el diagrama sectorial.

Los otros dos diagramas muestran el total visible en la pantalla respecto del total, ya que puede haber tal cantidad de líneas que no quepan, por lo que aparece el scroll azul. El otro pequeño gráfico sectorial muestra la relación respecto del total que tiene una selección que hagamos con el ratón sobre las filas de la tabla.

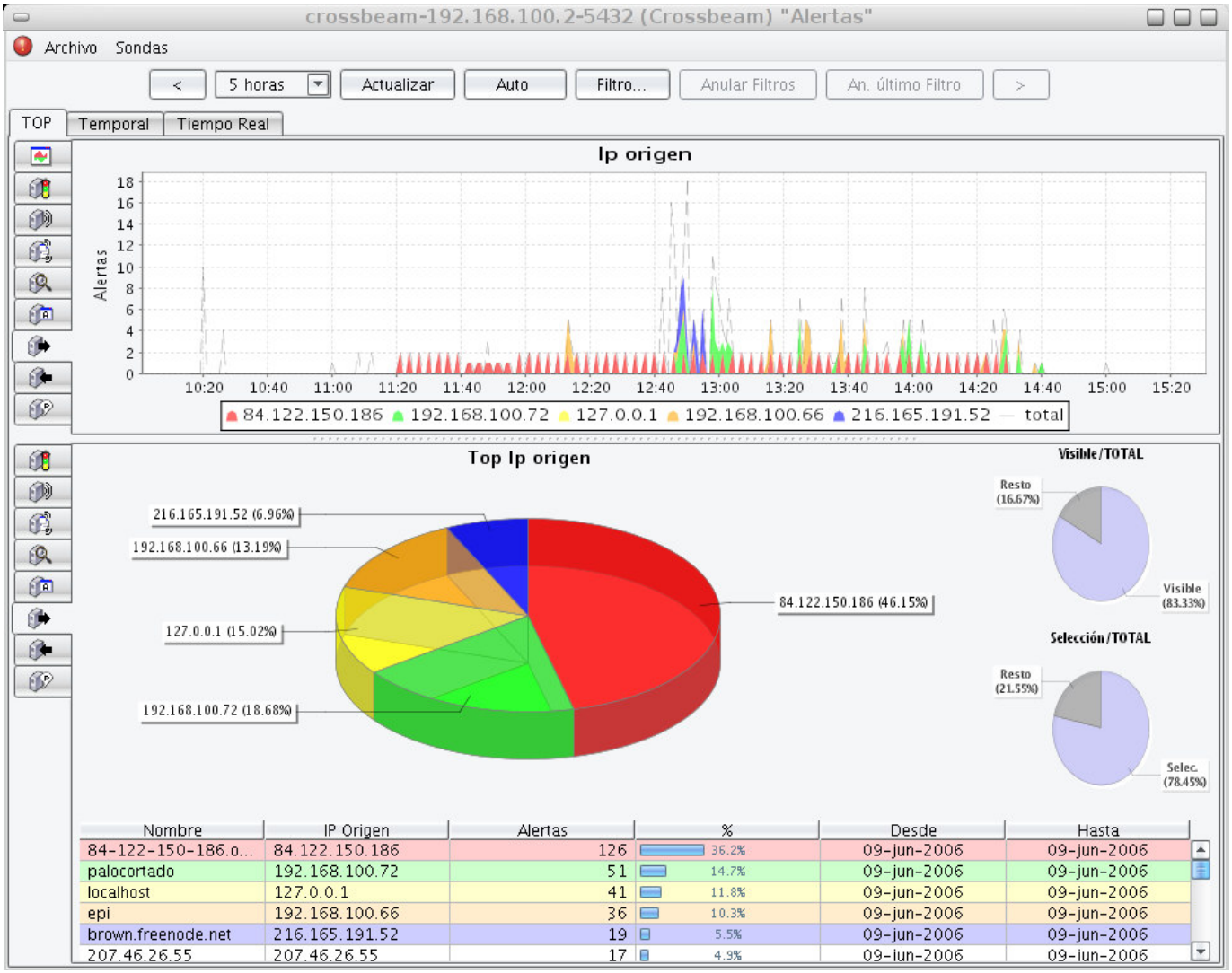


Figura 51.- Top dirección IP origen.

8.5.2.2.2 Clase *martealertTopTargetIP.*

Es la clase instanciable, subclase de *martealertTopConnection* que almacena las estadísticas por número de alertas y su distribución temporal según la dirección IP origen.

Es una clase de especial importancia ya que muestra los equipos (nombre resuelto más dirección IP) atacados. Deteniendo el ratón sobre las fechas FTS o LTS sobresalta un diálogo (tooltip) mostrando la hora.

El rango considerado es de un día; no obstante, se representan siete horas en el gráfico temporal como fruto de la ampliación de la imagen (zoom) que se consigue gracias a una utilidad de la librería gráfica como es seleccionar una porción de la ventana en la que se aloja dicho gráfico temporal.

Como excepción puede darse el caso que no exista dirección IP destino almacenada en la base de datos por ejemplo si se trata de una alerta que opera sobre un fichero de destino y no un mensaje que se envíe por un servicio de red. En este caso, en lugar de mostrarse un valor en blanco o nulo se representa este caso excepcional con la dirección de la máquina local (localhost) 127.0.0.1.

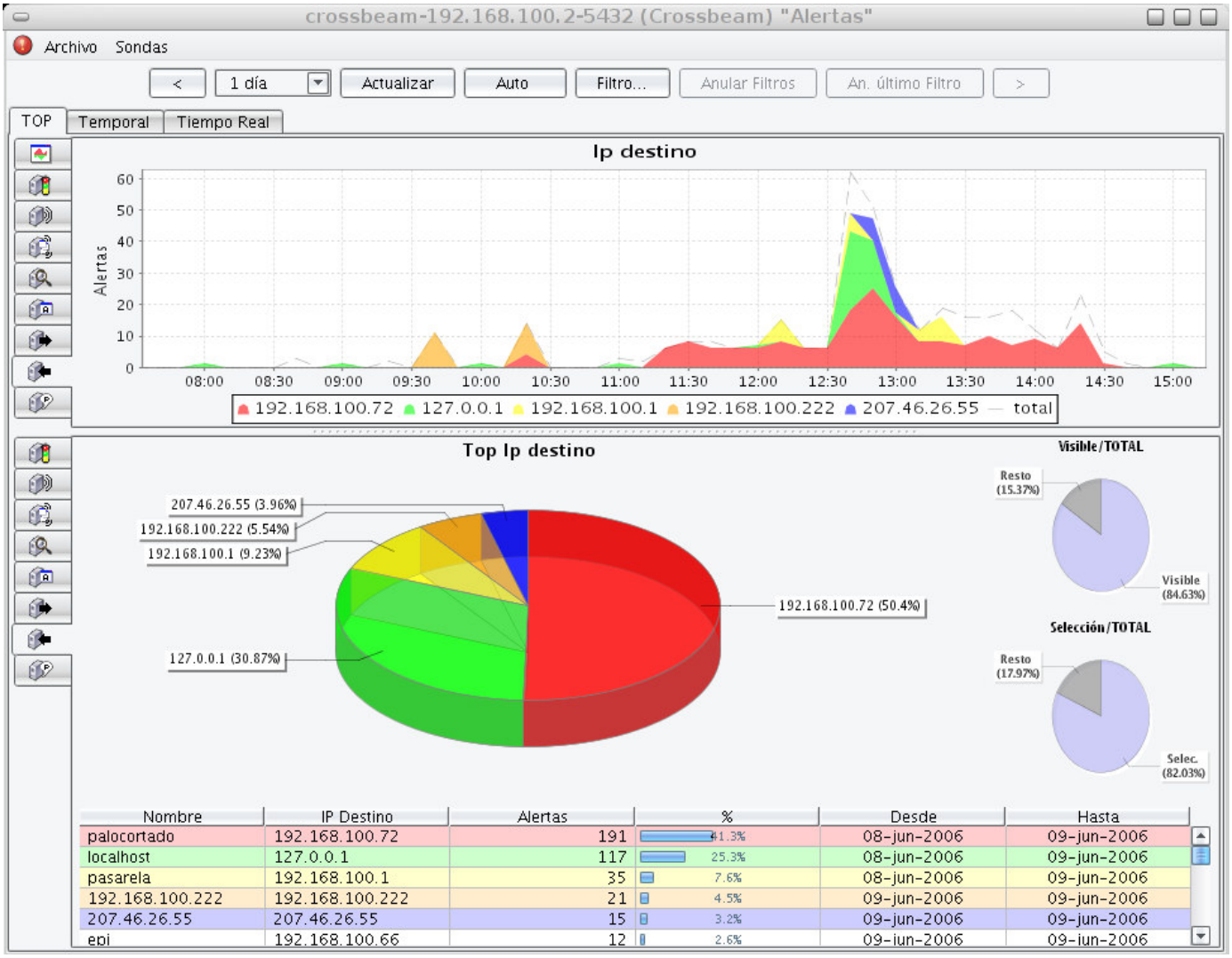


Figura 52.- Top dirección IP destino.

8.5.2.2.3 Clase *martealertTopTargetPort.*

Es la clase instanciable, subclase de *martealertTopConnection* que almacena las estadísticas por número de alertas y su distribución temporal según el puerto destino o la aplicación.

El puerto es resuelto a partir de una tabla llamada *services* externa al sistema prelude pero alojada en la misma base de datos, de modo que se obtiene de ella el nombre del puerto y una descripción que aparece al detener el ratón sobre cualquiera de los nombres resueltos de la tabla de Top aplicación.

A pesar de disponer de tablas para la resolución de puertos, hay determinados protocolos que no usan puertos, como el ICMP. En estos casos se utilizará la palabra "sin_registro" para resolver el puerto no existente o no definido "0".

Otro caso excepcional será cuando se trate de múltiples puertos, como en un escaneo múltiple con la herramienta *nmap*. En este caso, se representará el puerto 255 y el detalle de aplicaciones destino afectadas saldrá del informe detallado al que se accede desde el tercer nivel temporal o desde la pestaña de "Tiempo Real".

Al igual que en la figura 52, en la 53 también se muestra un gráfico ampliado para mejor observación de los intervalos en los que se han producido las alertas.

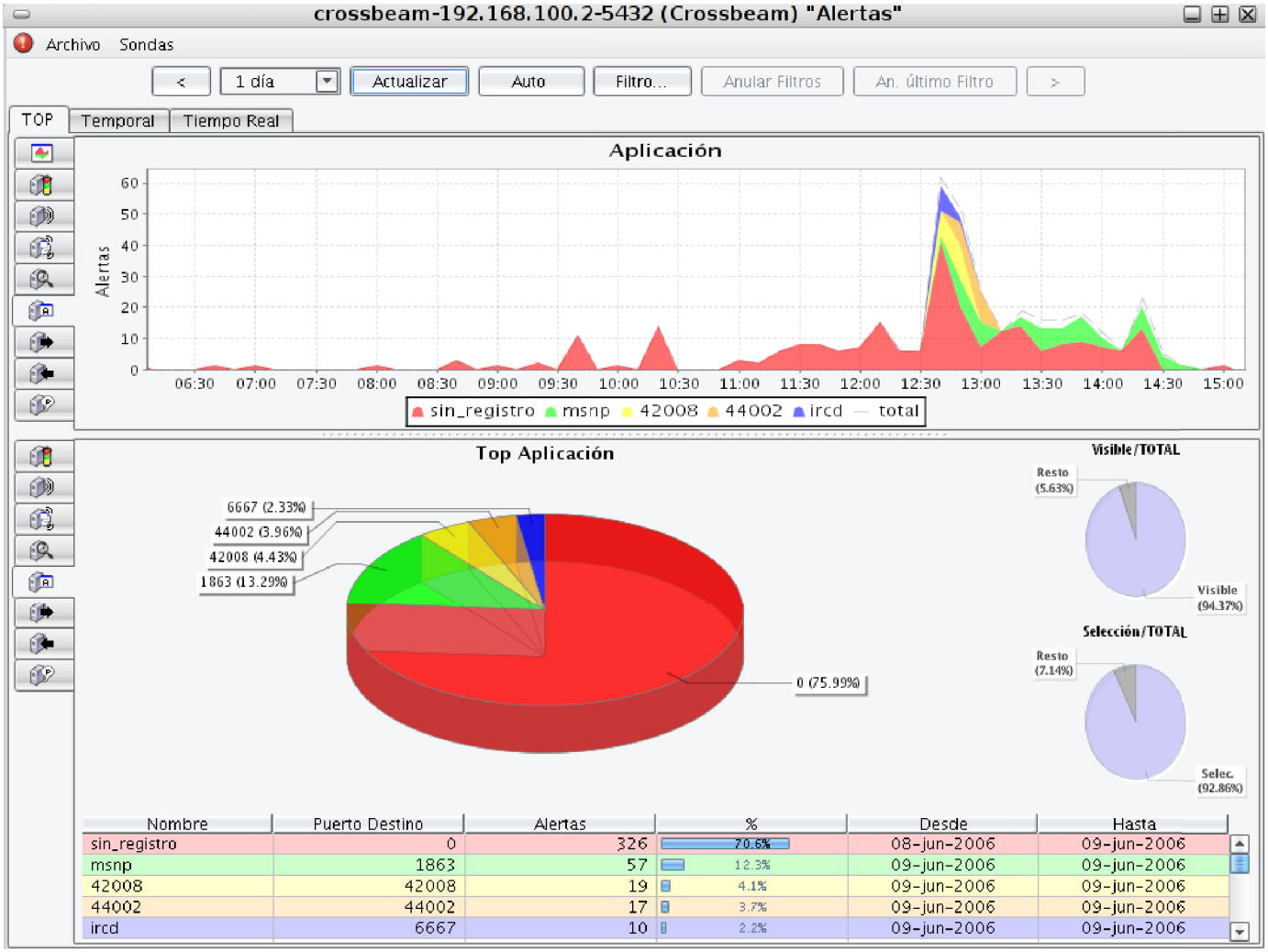


Figura 53.- Top dirección IP destino.

8.5.2.2.4 Clase *martealertTopAnalyzer.*

Es la clase instanciable, subclase de *martealertTopConnection* que almacena las estadísticas por número de alertas y su distribución temporal según la sonda.

En el caso actual únicamente se ha instalado una sonda de Snort y otra de Prelude-LML en la misma máquina en la que se encuentra el colector Prelude-Manager. En este caso, aunque ambas son la 127.0.0.1, el sistema *Prelude* resuelve la de Snort y almacena directamente en la base de datos "localhost" por lo que salen dos líneas. Así queda mucho más claro las alertas que son de mensajes del sistema (syslog) y las que detecta el IDS.

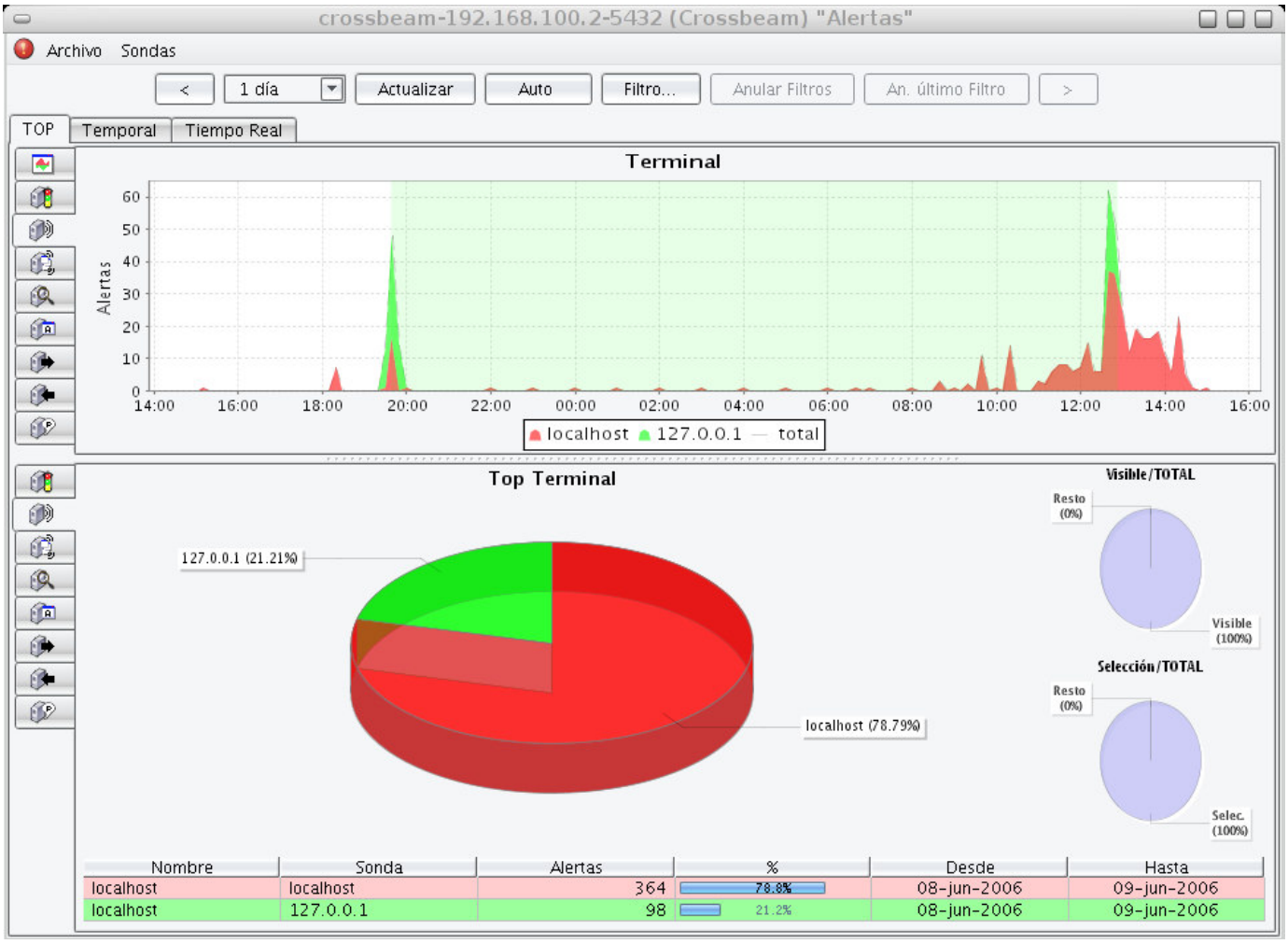


Figura 54.- Top sonda analizadora.

8.5.3.Clase *martealertFirstLevel.*

Es la clase que implementa un objeto que hereda la estructura básica de *jTable* y *martealertTable*. Dicha tabla almacena una agrupación de mensajes por parejas de direcciones IP origen y destino, a la vez que ordenados por fecha en orden descendente, es decir, que se muestran los grupos de arriba a abajo de más reciente a más antiguo.

Esto da una idea de las conexiones más recientes generadoras de alertas, a la vez que se muestra en la segunda columna la gravedad máxima del grupo. Esta columna condiciona el color de la fila, siendo roja para gravedad "high", amarilla para "medium", verde para "low" y azul para "info".

En la parte superior se muestra el gráfico temporal para el total de alertas, en el que se marcan los intervalos [FTS, LTS] cuando se selecciona alguna fila o conjunto de las mismas. Se observan escaneos de puertos, mensajes de chat en horario laboral, mensajes de "ping" etc.

Para el cálculo del primer nivel (en el que no se muestra información de puertos y protocolo), se calculan en primer lugar los pares de direcciones IP origen y destino ordenadas por fecha desde la más actual a la más antigua. Para cada par de direcciones se calcula todos los tipos de mensaje diferentes para ese par de direcciones, y en un último bucle se calcula el sensor, gravedad y sonda, además de representar los parámetros ya calculados: IP origen, IP destino, N x Tipo de mensaje (siendo N el número de ocurrencias) y las fechas de inicio y fin del intervalo.

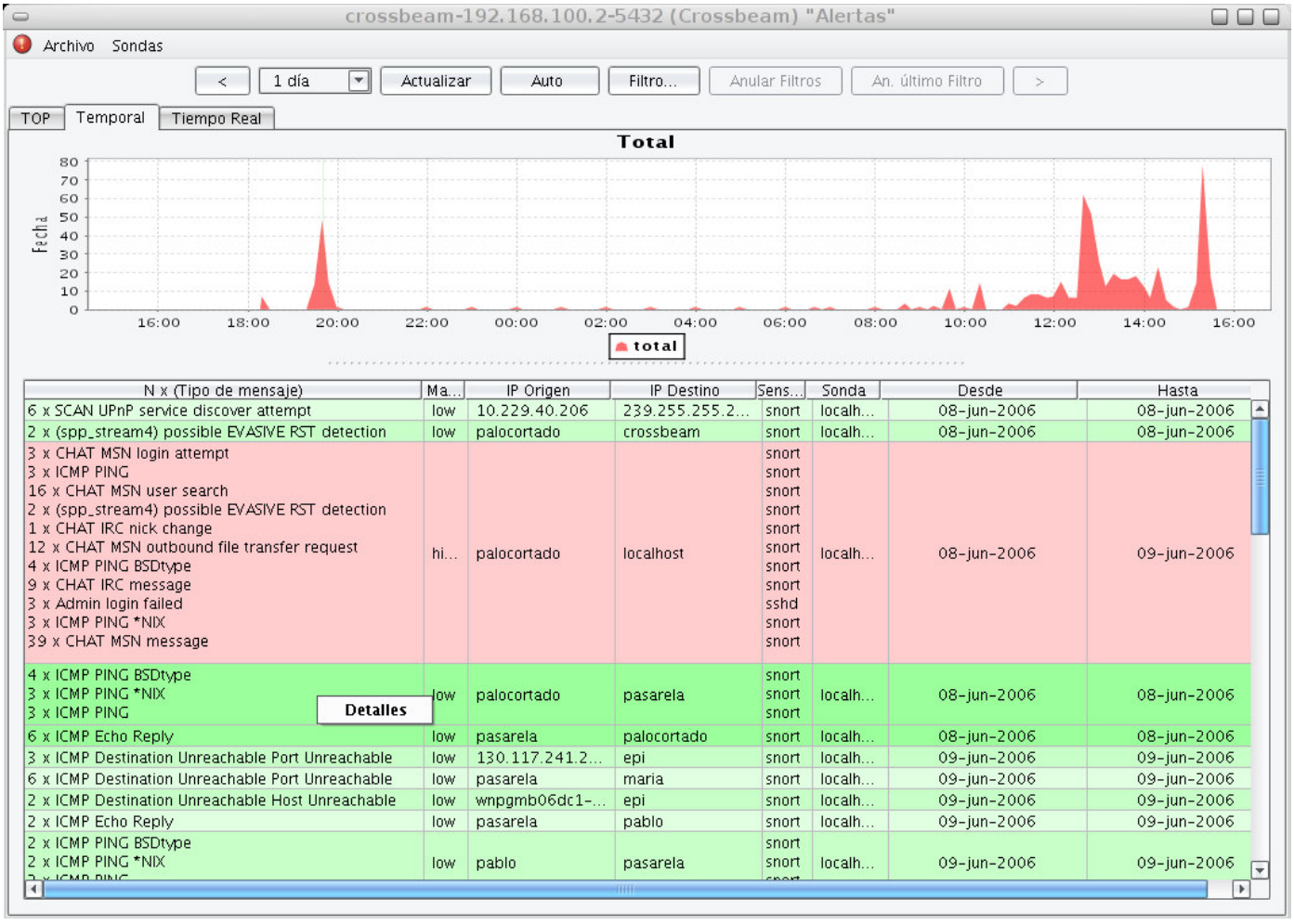


Figura 55.- Representación gráfica del objeto *martealertFirstLevel*.

Pinchando en un conjunto de líneas con el botón derecho aparece el rótulo "Detalles", con el que se accede al siguiente nivel de detalle: el segundo nivel. También se puede acceder haciendo doble click en una fila.

8.5.3.1. Clase *martealertSecondLevel*.

Es la subclase inmediatamente heredera de *martealertFirstLevel*, la cual desglosa las metafilas y filas del primer nivel en filas agrupadas por IP origen, IP destino y tipo de mensaje, ordenadas por fecha última vez vista (LTS o lo que se representa en el gráfico como columna "Hasta"). Al igual que el primer nivel, se trata de una clase instanciable.

Además de los campos del primer nivel, se añaden como novedad:

- *Completion*: Campo booleano que dice si la alerta se completó satisfactoriamente o no.
- *Protocolo*: Protocolo usado para el tipo de alertas mostrado.
- *Puerto Origen*: Identifica el puerto del que parte la conexión. Para protocolos que no usen puertos TCP ó UDP se mostrará "sin_registro". Si hay varios puertos origen diferentes para la misma conexión, se mostrará el menor.([30])
- *Puerto Destino*: Identifica la aplicación o puerto destino que recibe la acción mostrada por la alerta.

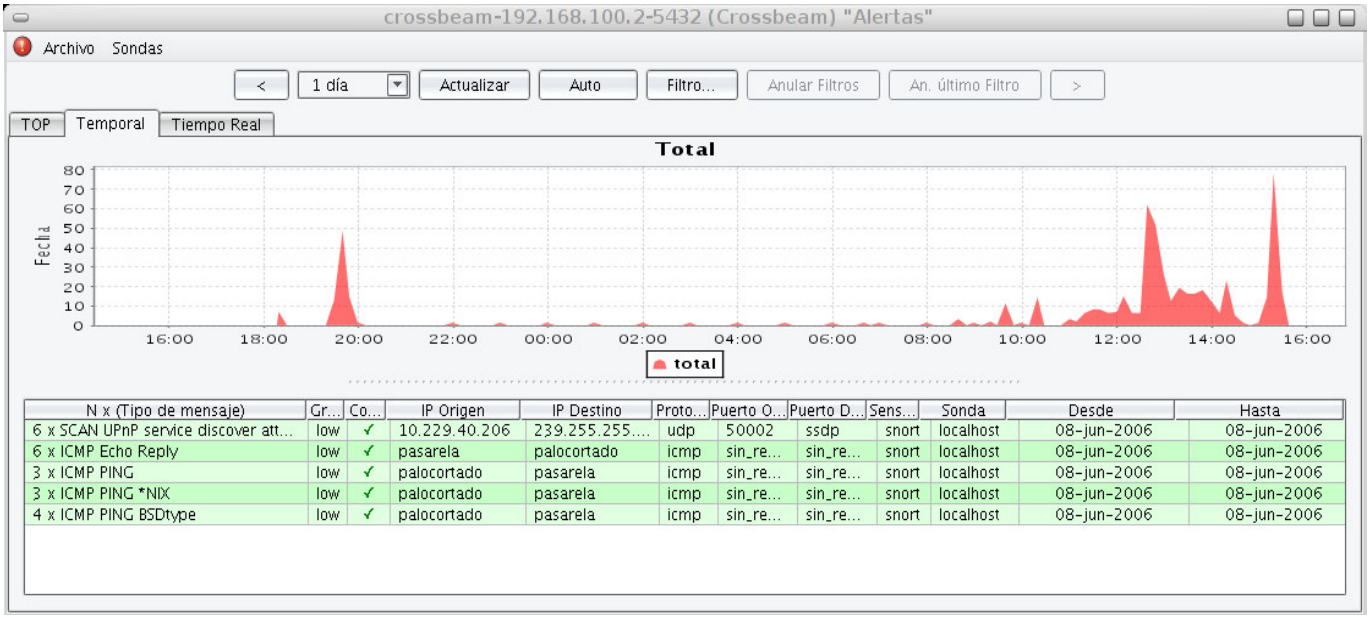


Figura 56.- Representación gráfica del objeto *martealertSecondLevel*.

Si volvemos a hacer doble click en una alerta o seleccionamos un conjunto y pinchamos en "Detalles" con el botón derecho, accederemos al desglose de alertas en el tercer nivel. Para volver al primer nivel hay que pulsar la tecla **Esc.**

[30] Para el caso de 6 escaneos de puertos "6 x SCAN UPnP..." podríamos tener como puertos origen 50002, 50004, 50005, 50010, 50018, 50022, con el mismo puerto destino 22 (sshd). Aparecería el 50002 para todo el grupo. Si se quisiera saber cada grupo habría que hacer doble click y desplegar los detalles (tercer nivel).

8.5.3.2. Clase *martealertThirdLevel.*

Es una subclase de *martealertSecondLevel*, también instanciable. Hereda todos los métodos del primer y segundo nivel a excepción de los que especifican las variables a ser seleccionadas en la base de datos. De hecho se obtiene con un único comando SQL cuya estructura es idéntica al que obtiene el segundo nivel a excepción de los campos que se especifican inmediatamente después del *"SELECT"* de PostgreSQL.

Los valores mostrados pueden verse cribados por filtros realizados en la pestaña TOP. Para la figura 57 se ha hecho doble click en una única fila del segundo nivel para el que se muestran tantas filas como repeticiones aparezcan en el mencionado *martealertSecondLevel*. Es decir, que una fila de segundo nivel en la que aparezca "***N*** *x SCAN UPnP service discover attempt*" se transforma en ***N*** filas del tercer nivel.

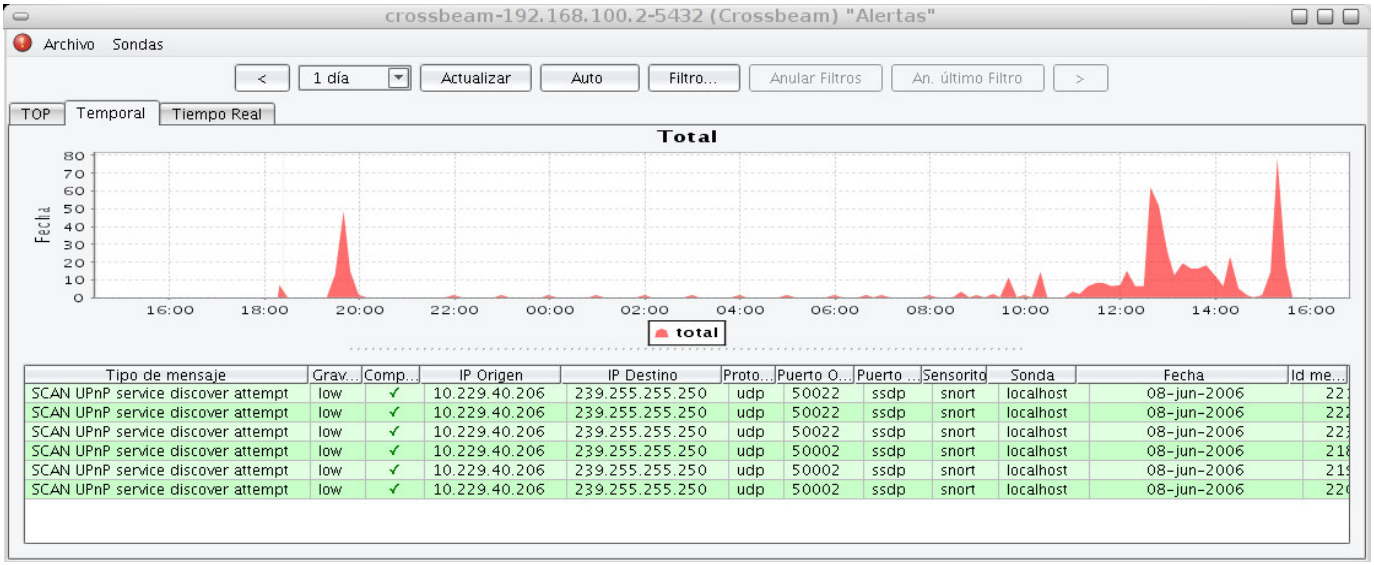


Figura 57.- Representación gráfica del objeto *martealertThirdLevel*.

Con este nivel se tiene un objeto muy similar al siguiente, de la clase *martealertRealTime*. No obstante, este tercer nivel no muestra las alertas en tiempo real ya que se le puede especificar un filtro temporal de otra fecha, a la vez que se muestra más nivel de detalle.

8.5.4.Clase *martealertRealTime.*

Es la clase que implementa un objeto que hereda la estructura básica de *jTable* y *martealertTable*. Dicha tabla almacena las últimas alertas recibidas sin filtrar, para todas las sondas, sensores y direcciones. El contenido es similar al del tercer nivel en el sentido que aparecen las alertas de forma individual, sin agrupar por origen, destino ni tipo de mensaje. No obstante, a diferencia del tercer nivel, se ha sacrificado detalle como los puertos o el protocolo a costa de obtener una mayor rapidez.

Si se pincha en el botón derecho se permiten las opciones de congelación o de mostrar más alertas de las que hayan desaparecido de la pantalla por actualización de alertas nuevas.

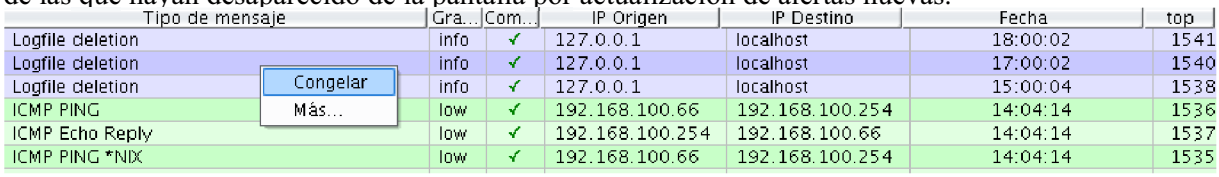
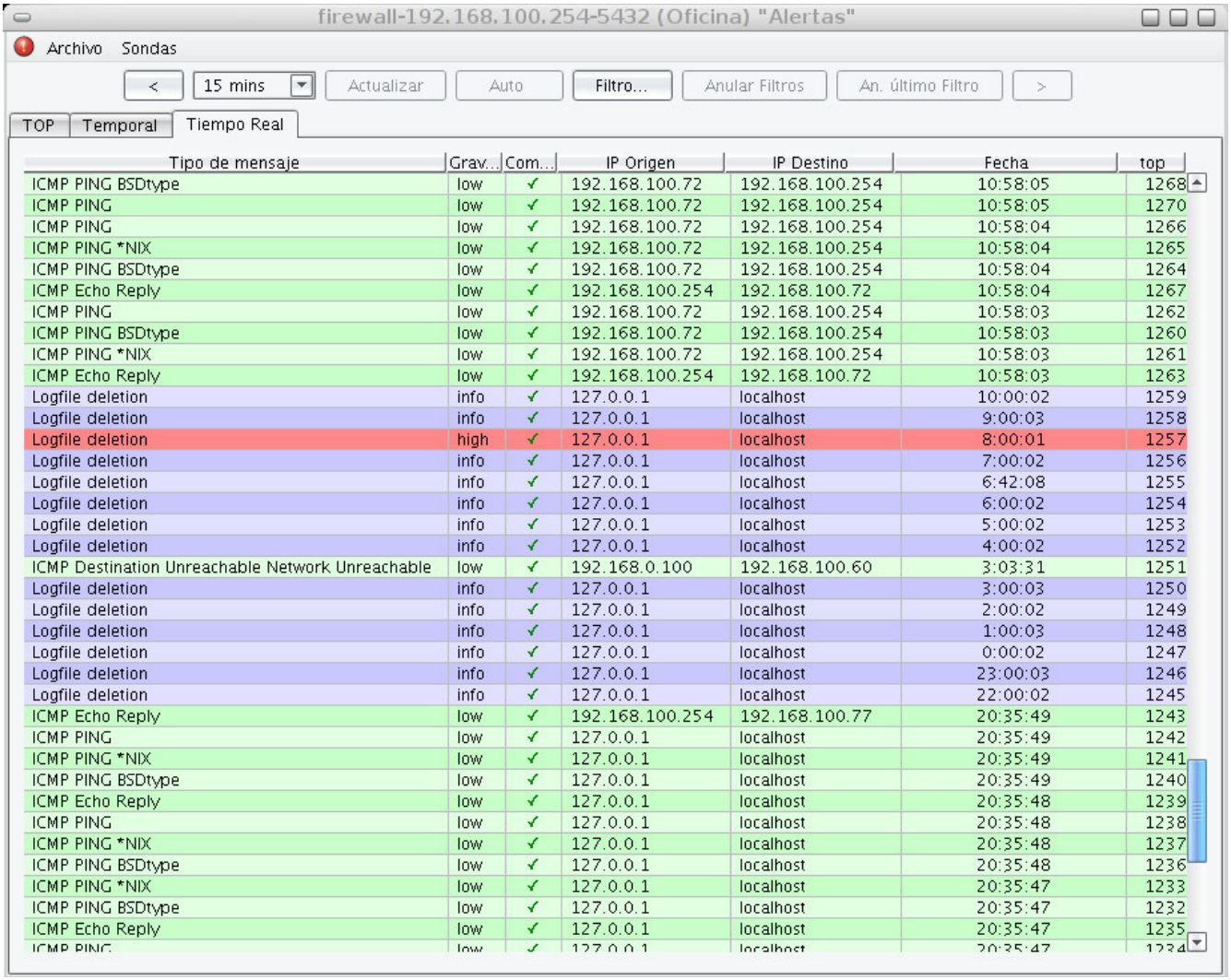


Figura 58.- Opciones permitidas con el botón derecho en el objeto *martealertRealTime*.

Conforme van llegando las alertas se van añadiendo a la parte superior y se desplazan hacia abajo el resto de alertas. Para evitar desbordamiento está controlado con un límite de ráfaga de llegada.

En la figura 59 se puede observar cómo en la fecha se muestra la hora sin especificar el día o mes. Para ello, saldrá en una pequeña ventana al detener el ratón sobre dicha hora. También se puede observar cómo el sistema es capaz de discernir dos tipos diferentes de prioridad para un mismo tipo de mensaje entre la misma dirección IP origen y destino. Para este caso es dentro de la misma máquina que tiene instalado el colector de alertas (localhost). Para saber la diferencia entre las dos alertas de borrado de fichero hay que acceder a un nivel mayor de detalle.

Haciendo doble click sobre una y sólo una fila accedemos al siguiente nivel de detalle: *martealertDetailedReport* o informe detallado de alertas.



Tipo de mensaje	Grav...	Com...	IP Origen	IP Destino	Fecha	top
ICMP PING BSDtype	low	✓	192.168.100.72	192.168.100.254	10:58:05	1268
ICMP PING	low	✓	192.168.100.72	192.168.100.254	10:58:05	1270
ICMP PING	low	✓	192.168.100.72	192.168.100.254	10:58:04	1266
ICMP PING *NIX	low	✓	192.168.100.72	192.168.100.254	10:58:04	1265
ICMP PING BSDtype	low	✓	192.168.100.72	192.168.100.254	10:58:04	1264
ICMP Echo Reply	low	✓	192.168.100.254	192.168.100.72	10:58:04	1267
ICMP PING	low	✓	192.168.100.72	192.168.100.254	10:58:03	1262
ICMP PING BSDtype	low	✓	192.168.100.72	192.168.100.254	10:58:03	1260
ICMP PING *NIX	low	✓	192.168.100.72	192.168.100.254	10:58:03	1261
ICMP Echo Reply	low	✓	192.168.100.254	192.168.100.72	10:58:03	1263
Logfile deletion	info	✓	127.0.0.1	localhost	10:00:02	1259
Logfile deletion	info	✓	127.0.0.1	localhost	9:00:03	1258
Logfile deletion	high	✓	127.0.0.1	localhost	8:00:01	1257
Logfile deletion	info	✓	127.0.0.1	localhost	7:00:02	1256
Logfile deletion	info	✓	127.0.0.1	localhost	6:42:08	1255
Logfile deletion	info	✓	127.0.0.1	localhost	6:00:02	1254
Logfile deletion	info	✓	127.0.0.1	localhost	5:00:02	1253
Logfile deletion	info	✓	127.0.0.1	localhost	4:00:02	1252
ICMP Destination Unreachable Network Unreachable	low	✓	192.168.0.100	192.168.100.60	3:03:31	1251
Logfile deletion	info	✓	127.0.0.1	localhost	3:00:03	1250
Logfile deletion	info	✓	127.0.0.1	localhost	2:00:02	1249
Logfile deletion	info	✓	127.0.0.1	localhost	1:00:03	1248
Logfile deletion	info	✓	127.0.0.1	localhost	0:00:02	1247
Logfile deletion	info	✓	127.0.0.1	localhost	23:00:03	1246
Logfile deletion	info	✓	127.0.0.1	localhost	22:00:02	1245
ICMP Echo Reply	low	✓	192.168.100.254	192.168.100.77	20:35:49	1243
ICMP PING	low	✓	127.0.0.1	localhost	20:35:49	1242
ICMP PING *NIX	low	✓	127.0.0.1	localhost	20:35:49	1241
ICMP PING BSDtype	low	✓	127.0.0.1	localhost	20:35:49	1240
ICMP Echo Reply	low	✓	127.0.0.1	localhost	20:35:48	1239
ICMP PING	low	✓	127.0.0.1	localhost	20:35:48	1238
ICMP PING *NIX	low	✓	127.0.0.1	localhost	20:35:48	1237
ICMP PING BSDtype	low	✓	127.0.0.1	localhost	20:35:48	1236
ICMP PING *NIX	low	✓	127.0.0.1	localhost	20:35:47	1233
ICMP PING BSDtype	low	✓	127.0.0.1	localhost	20:35:47	1232
ICMP Echo Reply	low	✓	127.0.0.1	localhost	20:35:47	1235
ICMP PING	low	✓	127.0.0.1	localhost	20:35:47	1234

Figura 59.- Representación gráfica del objeto *martealertRealTime*.

8.5.5.Clase *martealertDetailedReport.*

Es una clase instanciable que genera un informe detallado para una alerta concreta, a partir de un identificador de alerta. El informe consta de la estructura que se muestra en la figura 60: un vector tridimensional que se forma con los vectores bidimensionales de cada grupo.

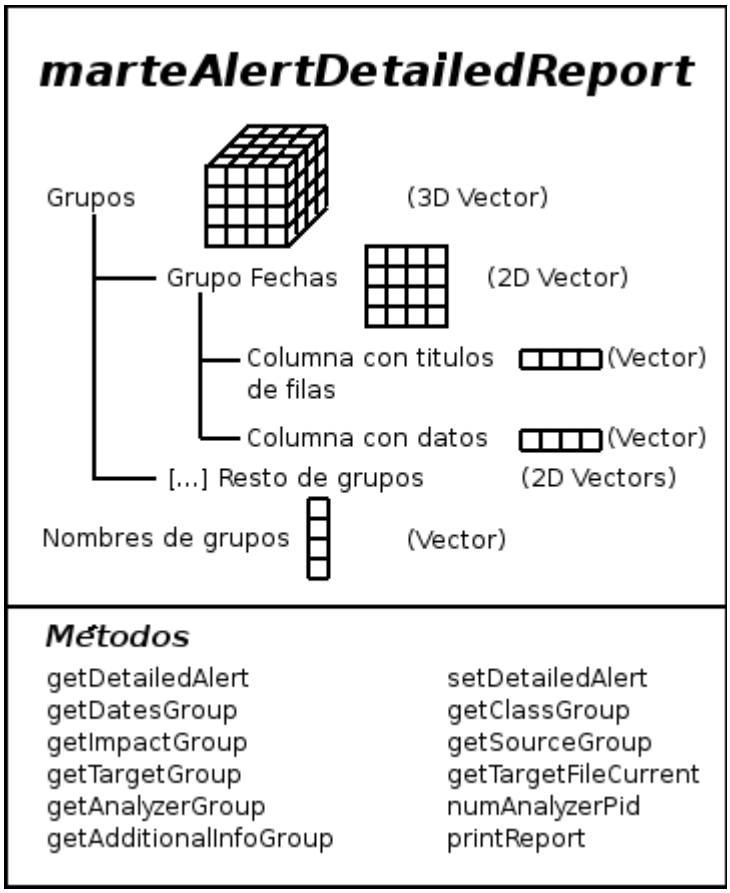


Figura 60.- Estructura del objeto *martealertDetailedReport*.

En la figura se muestra **"[...] resto de grupos"** para exponer implícitamente que el resto de grupos son vectores bidimensionales al igual que el grupo de fechas. A continuación se especifican los posibles grupos, aunque no por ello tendrán que aparecer siempre. Cuando los valores que contengan sean nulos no aparecerán. Por ejemplo, si se efectúa una operación de borrado de fichero, aparecerá el grupo del fichero objetivo actual, devuelto por *getTargetFileCurrent*. En cualquier otro caso este grupo no aparecería, como en el caso de autenticación por PAM, mensajes de chat...

- *Grupo de fecha*: Información de las fechas de detección, creación y análisis de la alerta.

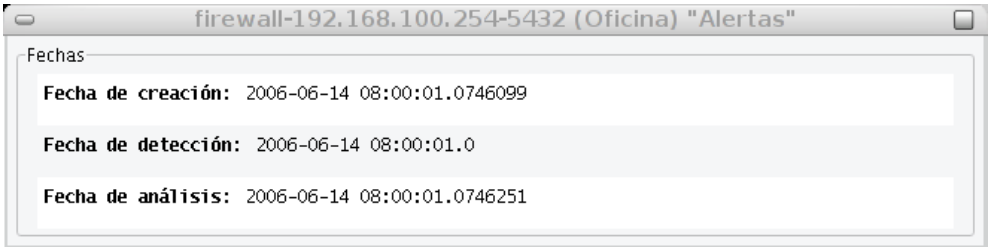


Figura 61.- Grupo Fechas en el informe detallado.

- *Grupo de tipo de mensaje*: Aparece tipo de mensaje ("classification").

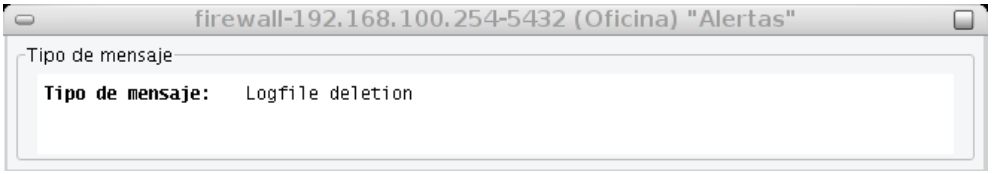


Figura 62.- Grupo de tipo de mensaje en el informe detallado.

- *Grupo de Impacto*: Es el grupo en el que aparece el alcance, gravedad y si fue satisfactorio o no. Da una breve descripción explicando y justificando la gravedad atribuida a la alerta.

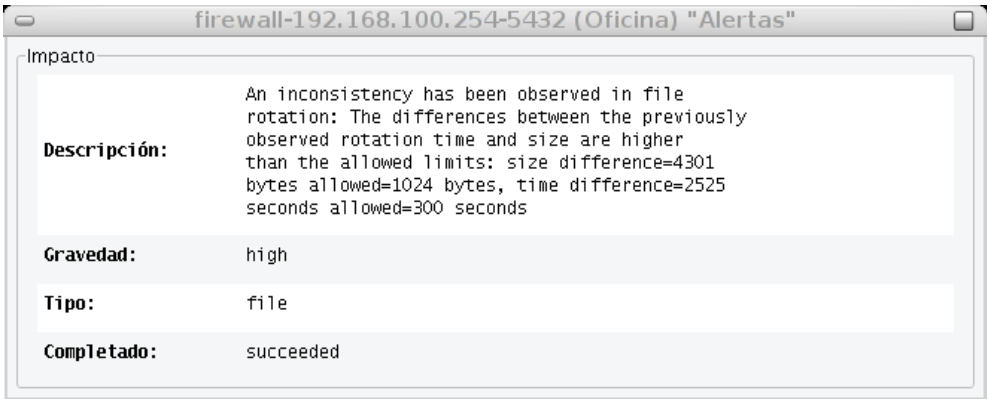


Figura 63.- Grupo de Impacto en el informe detallado.

- *Grupo de dirección origen*: Es el grupo que muestra la información referente al origen de la alerta. Se suele mostrar la dirección IP, puerto y protocolo, ya que sin dirección IP no aparecerá el grupo. Ocasionalmente se podrán mostrar los usuarios (root, postgres,...) que originaron la alerta y la dirección MAC origen para alertas generadas por ejemplo por el sensor de *arpwatch*.

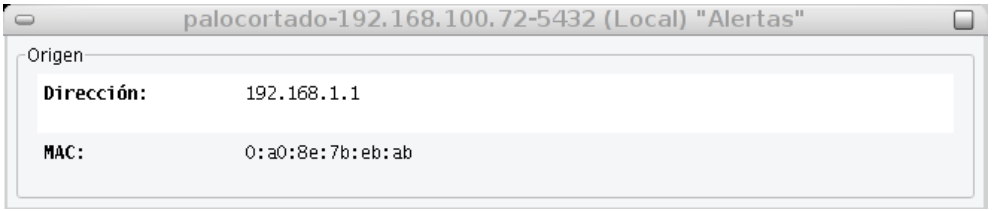


Figura 64.- Grupo de dirección origen en el informe detallado.

- *Grupo de dirección destino*: Es el grupo que muestra la información referente al destino de la alerta de manera análoga al grupo de origen de la alerta. Además puede mostrar el nombre y *pid* del proceso receptor de la alerta.

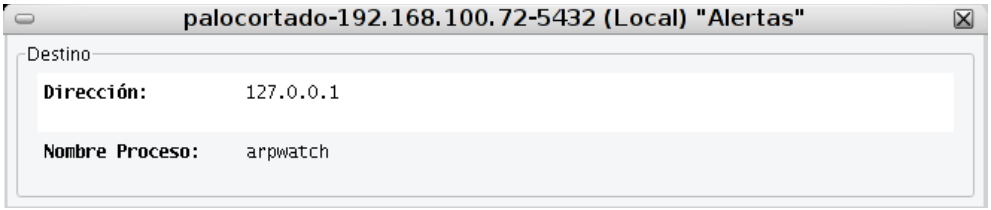


Figura 65.- Grupo de dirección destino en el informe detallado.

- *Grupo de fichero de destino actual*: En caso de que la alerta se produzca por una acción sobre un fichero, se muestra la información referente al nombre, ruta, fecha de acceso, tamaño y número de Nodo-I (En caso de particionamiento en Linux).

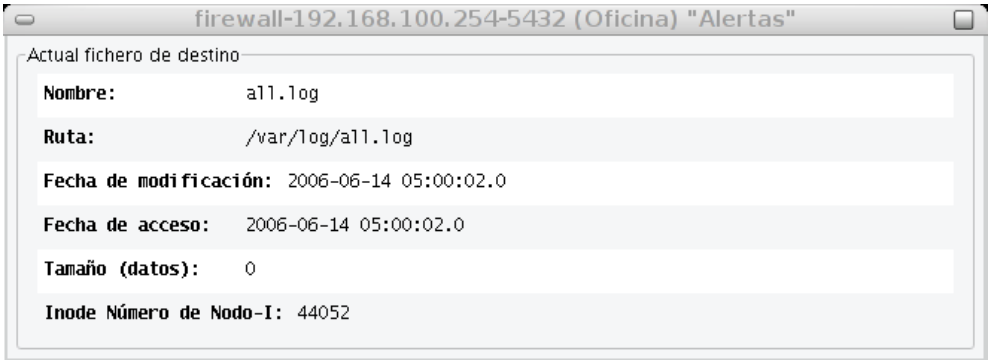
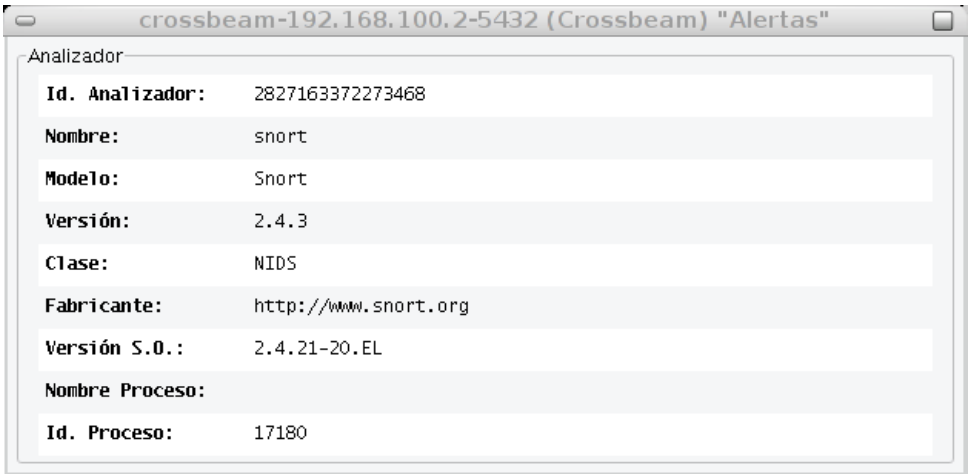


Figura 66.- Grupo de información de fichero objetivo en el informe detallado.

- *Grupo de Analizador*: Puede estar formado por varios analizadores. Como regla general, podremos disponer de los analizadores *prelude-LML, prelude-Manager* y algún otro adicional como por ejemplo *Snort* o *Arpwatch,* según el sensor. A continuación se detalla el de Snort y Arpwatch.



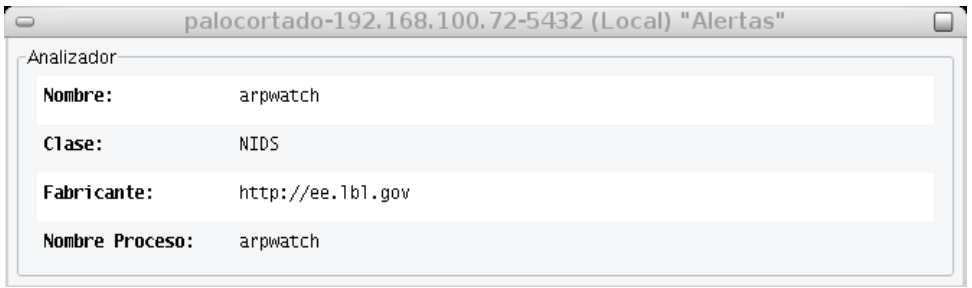


Figura 67.- Grupos de información de analizadores en el informe detallado.

- *Grupo de datos adicionales*: Es un grupo cuyo número de filas no está determinado a priori, ya que la relación *Prelude_AdditionalData* y Prelude en general permite que se definan campos arbitrarios en esta tabla. Por ejemplo, podrá aparecer el origen y el mensaje completo. Otra opción muy común es que se muestren los campos del datagrama IP (checksum, offset,

length...). Incluso para alertas tipo "Chat MSN" se puede mostrar la carga (*payload*) o mensaje. A continuación, los detalles de dos tipos de mensaje distintos:

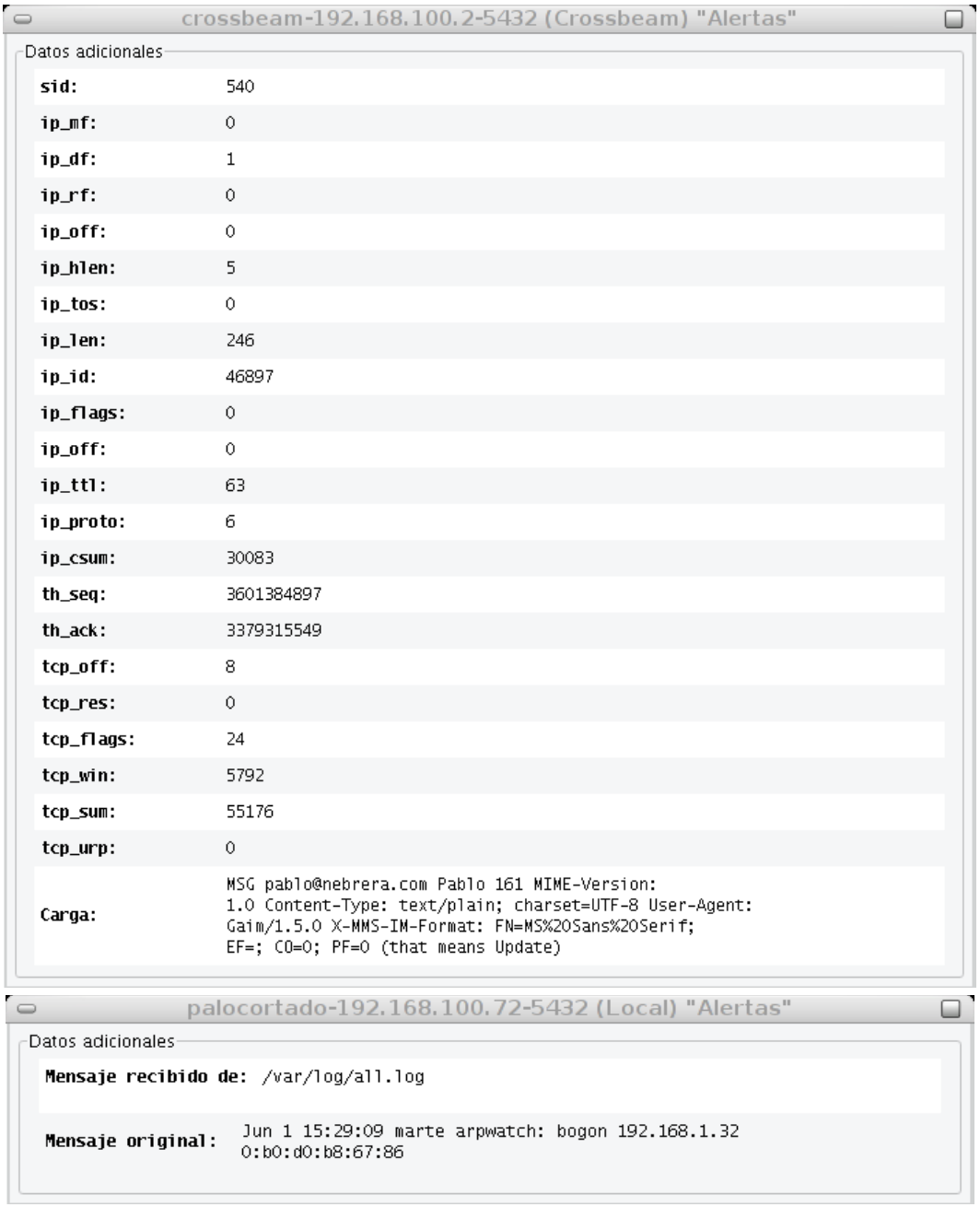


Figura 68.- Distintas opciones de datos adicionales en el informe detallado.

8.5.6.Clase *martealertAlertAgents*.

Es la clase que implementa un objeto que hereda la estructura básica de *jTable* y *martealertTable*. Dicha tabla almacena las sondas activas e inactivas. Además, se muestra el sistema operativo, la fecha del último pulso (heartbeat) recibido, y la dirección IP, siempre que hubiera sido especificada en los ficheros de configuración de Prelude de cada sonda. Además aparecen los gestores Prelude-Manager e incluso si hacen de sonda de otros Prelude-Manager, es decir, si hacen *relay*. El intervalo viene especificado en segundos. Si no se recibe un pulso en el tiempo especificado en el intervalo, la sonda está inactiva (offline).

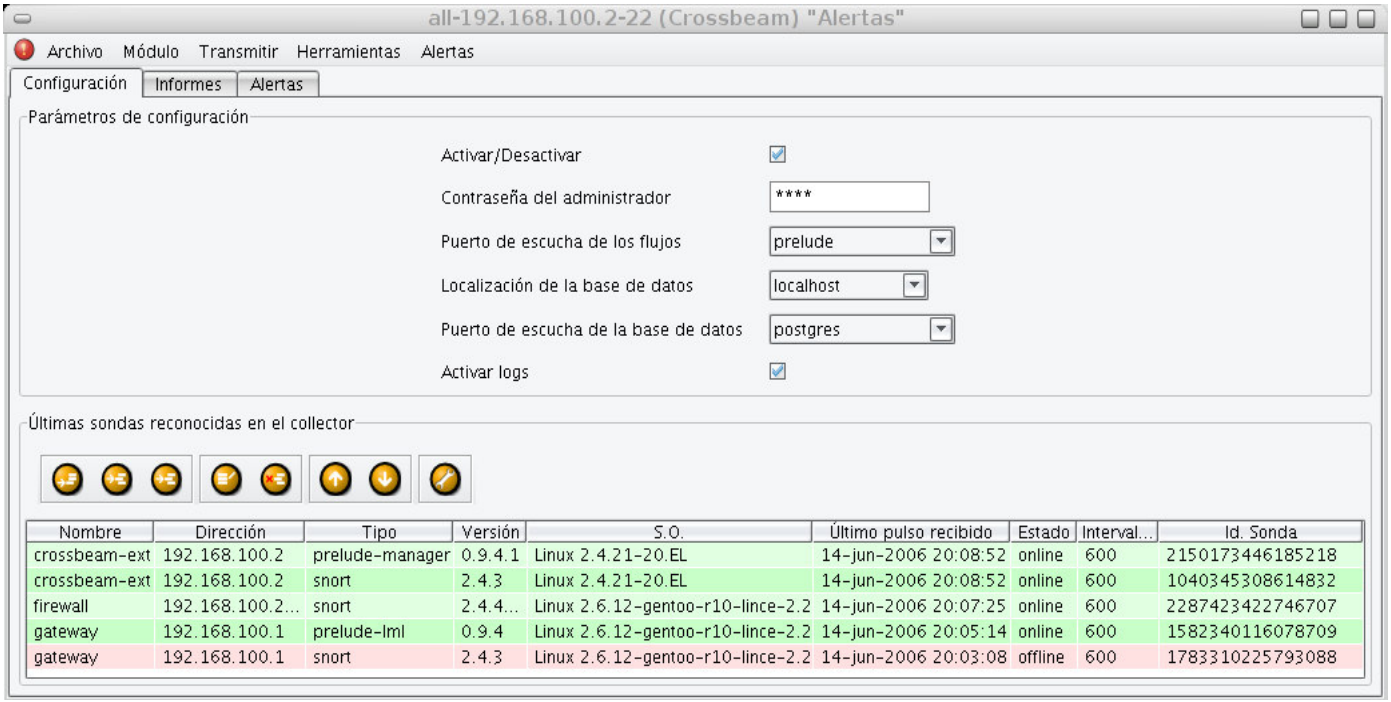


Figura 69.- Agentes en línea e inactivos (sondas y colector).

8.6.Paquete *martealertFilter*.

Es el paquete que contiene todas las clases referentes a la creación, modificación y uso de los filtros para la librería *martealert*. Para una mayor modularidad, está dividido en dos partes. La primera se encarga de la implementación del objeto *martealertHashFilter* y sus métodos, el cual hereda de *martealertHashtable* que a su vez es subclase de *hashtable*. La segunda parte se encarga de generar la ventana para filtros avanzados: *martealertAdvancedFilterView* y *martealertAdvancedFilterTest*.

8.6.1.Clase *martealertHashtable*.

Una tabla *hash* es un vector cuyo índice puede ser cualquier cadena de caracteres. Simplifica enormente el indexado y el acceso es rápido ya que se conforma a partir de un árbol binario. No obstante, ocupa más memoria que un objeto *Vector* o un *array*.

Los métodos que la conforman son:

- *public Object get(Object key)*: Obtiene el valor almacenado en el índice *key*, devolviéndose una cadena vacía en lugar de *null*.
- *public void setFilter(String field, String filter)*: Asigna al elemento *field* de la tabla *hash* el valor especificado por *filter*, que será del tipo: “t5.port = '0'”, incluyendo los valores por defecto, como el puerto '0' que se especifica cuando no hay entrada en la tabla de servicios.
- *public void printHashtable()*: Imprime una tabla *hash* genérica por pantalla en modo texto para depuración.
- *public String getTemporalFilter(String field)*: Obtiene un filtro válido para las representaciones de gráficas temporales, especificando el campo.

8.6.2.Clase *martealertHashFilter*.

Es la clase que implementa el objeto que almacenará los filtros que se apliquen a las alertas. Podrá filtrarse por cada uno de los campos, lo cual se almacena en una tabla *hash*, además de contar con valores enteros y cadena de caracteres (*properties*) para un acceso más rápido a datos frecuentes como el FTS o su id.

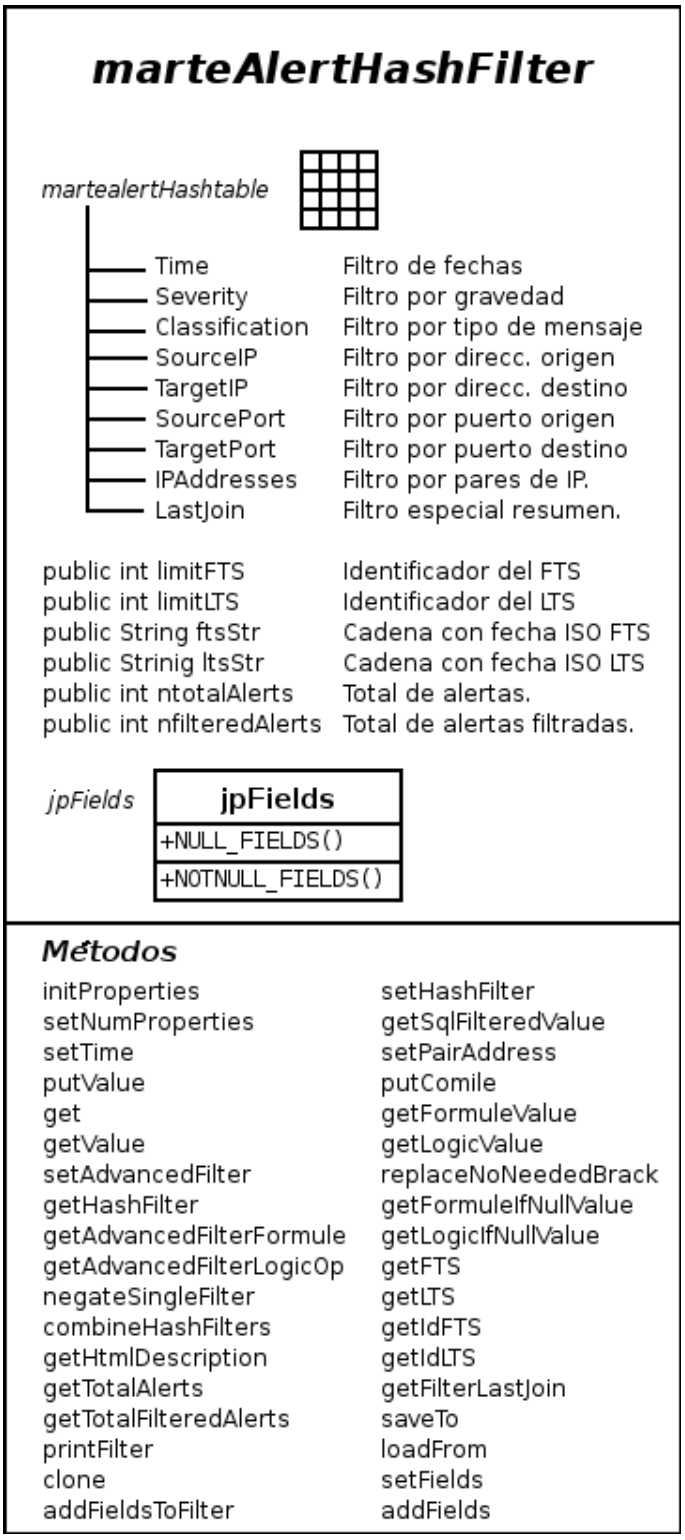


Figura 70.- Estructura del objeto *martealertHashFilter*.

Además, dado que hay campos que pueden contener valores nulos, éstos se clasifican en dos grupos y por tanto se filtrará de forma diferente. Esto último se recoge en el objeto *jpFields*, el cual tendrá para cada grupo de campos sendos vectores.

Dado que se dispone de una gran cantidad de métodos y muchos de ellos son métodos privados cuya única transcendencia es reutilizar código en los métodos públicos y protegidos. Por tanto, a continuación se detallan los métodos principales para el acceso y modificación del filtro. Para el resto, véase el anexo de la documentación en *javadoc*.

- *public void initProperties(Connection, FTS, LTS)*: Inicializa los valores enteros del número total de alertas, total de alertas filtradas, valor del FTS, LTS y sus identificadores.
- *public void setTime(FTS, LTS)*: Conforma la cadena del filtro de tiempo en la tabla *hash* para el índice TIME y la almacena.
- *public synchronized putValue(key, value)*: Almacena en el índice especificado por *key* el valor "(*value*) AND ", teniendo *value* el formato siguiente: t5.port = '22'. En caso de que se encuentre NULL o un valor por defecto, se añadiría "t5.port IS NULL" en el índice LASTJOIN.
- *public Object getValue*
- *public static martealertHashFilter getHashFilter(int rows[], topTable)*: Obtiene el filtro combinado resultado de la selección de las filas especificadas en *rows* para la tabla *topTable*. Por ejemplo, si se seleccionan las filas que especifican los puertos 80 y 22 de la tabla de *topTargetPort*, el resultado sería un filtro en cuyo índice TARGETPORT contiene para este caso: "`(t5.port = '22' OR t5.port = '80') AND `"
- *public void setHashFilter(int rows[], topTable)*: De forma similar al método anterior, obtiene el objeto *martealertHashFilter* pero lo almacena en *this* en lugar de devolverlo de forma estática.
- *public martealertHashFilter setAdvancedFilter(String field, String value, String operator*): Es el método que asigna el filtro avanzado con el valor, campo y operador lógico especificado. Se invoca desde la clase *martealertAdvancedFilterView* al rellenar el diálogo de filtro avanzado.
- *public void setPairAddress(martealertTable)*: Asigna un par de direcciones al índice IPADDRS para la generación del primer nivel que clasifica por parejas IP origen – IP destino antes de ordenar por tiempo.
- *protected Vector getAdvancedFilterFormule()*: Obtiene un vector de cadenas del tipo "(192.168.100.72 OR 210.160.100.146)" cuando va a cargar el filtro "(t3.address = '192.168.100.72' OR t3.address = '210.160.100.146')". Se usa para la carga de filtros avanzados desde fichero.
- *protected Vector getAdvancedFilterLogicOp()*: Idem de la anterior pero para obtener el operador lógico del filtro avanzado almacenado en fichero.
- *public static martealertHashFilter combineHashFilters(Connection, Vector filters, FTS, LTS)*: Combina de forma inclusiva los filtros que haya en el vector de filtros *filters* y en caso de que sea un filtro vacío, se devuelve un filtro con el índice TIME inicializado gracias a los valores FTS y LTS pasados como parámetros.
- *protected String getFilterLastJoin()*: Obtiene el filtro adecuado para la última operación *Join* de las consultas SQL.
- *public void saveTo(FileOutputStream out)*: Guarda el filtro avanzado en un archivo.

- *public void loadFrom(FileInputStream in)*: Carga un fichero desde un flujo de entrada.
- *public int getTotalAlerts()*: Obtiene el número total de alertas en el rango temporal especificado.
- *public int getTotalFilteredAlerts()*: Obtiene el número total de alertas considerando todas las componentes del filtro.
- *public void printFilter()*: Método que hereda de *martealertHashtable* el método *printHashtable()* y añade la impresión de los *properties* y los vectores del objeto *jpFields*.
- *public Object clone()*: Clona el filtro this y lo devuelve como Object (es necesario el cast).
- *protected void setFields(String []nn, String []nllj)*: Asigna los valores de las tablas *nn* (Campos que **no** pueden ser nulos por los que se está filtrando), y *nllj* (Campos que pueden ser nulos por los que se está filtrando). Los valores anteriores se pierden.
- *protected void setFields(String []nn, String []nllj, int fd)*: Añade los valores de las tablas *nn* (Campos que **no** pueden ser nulos por los que se está filtrando), y *nllj* (Campos que pueden ser nulos por los que se está filtrando). Los valores anteriores **no** se pierden, y se permite la opción de especificar una bandera para indicar si se han añadido valores compuestos o no. Por ejemplo: se podría añadir dos campos simples en *nn{SEVERITY, COMPLETION},* en cuyo caso *fd = 0* (valor por defecto) u optimizar el acceso a la base de datos con un campo complejo *nllj{SEVERITY_COMPLETION},* en cuyo caso *fd = 1*.

8.6.3.Clase *martealertAdvancedFilterView.*

Es la clase que implementa la ventana del diálogo para la generación de filtros avanzados. Este diálogo tendrá objetos *martealertDatePicker* o calendario, *jComboBox* para las celdas desplegables, botones etc.

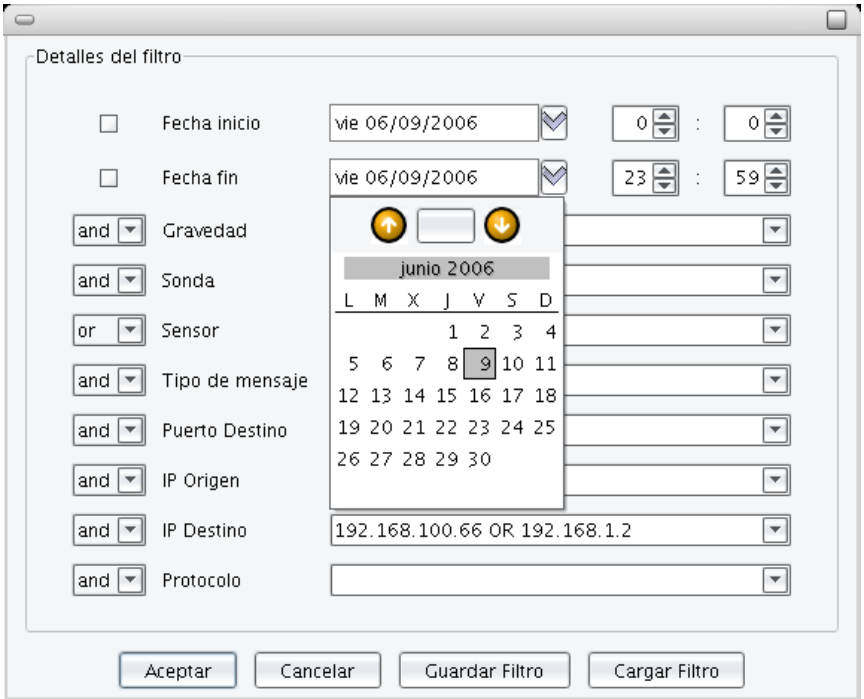


Figura 71.- Calendario de la ventana de filtros avanzados.

El diálogo está formado principalmente por objetos de las clases gráficas *javax.swing,* y *java.awt*.

Al pulsar en el botón Aceptar se aplica el filtro. Si se pulsa en Cancelar se cierra la ventana sin aplicar filtro. Si se pulsa Guardar Filtro ó Cargar Filtro aparece un objeto tipo *javax.swing.JFileChooser* gracias al cual se abre una ventana de selección de fichero como la de la figura 72:

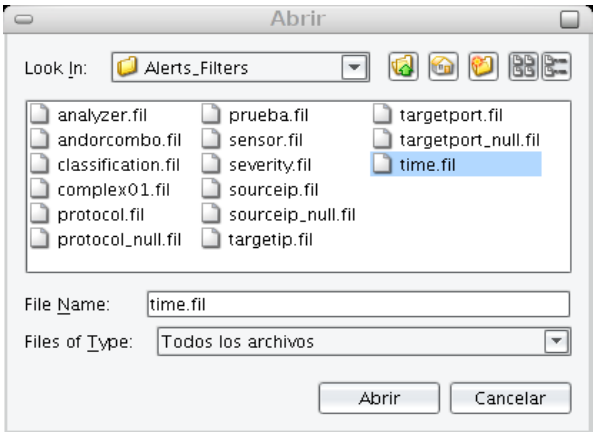


Figura 72.- Carga de filtros (objeto *JFileChooser*).

8.6.3.1. Clase *martealertAdvancedFilterTest.*

Clase que testea la ventana del diálogo con un método *main* que calcula los *top* para facilitar la selección de los valores por los que queremos filtrar.

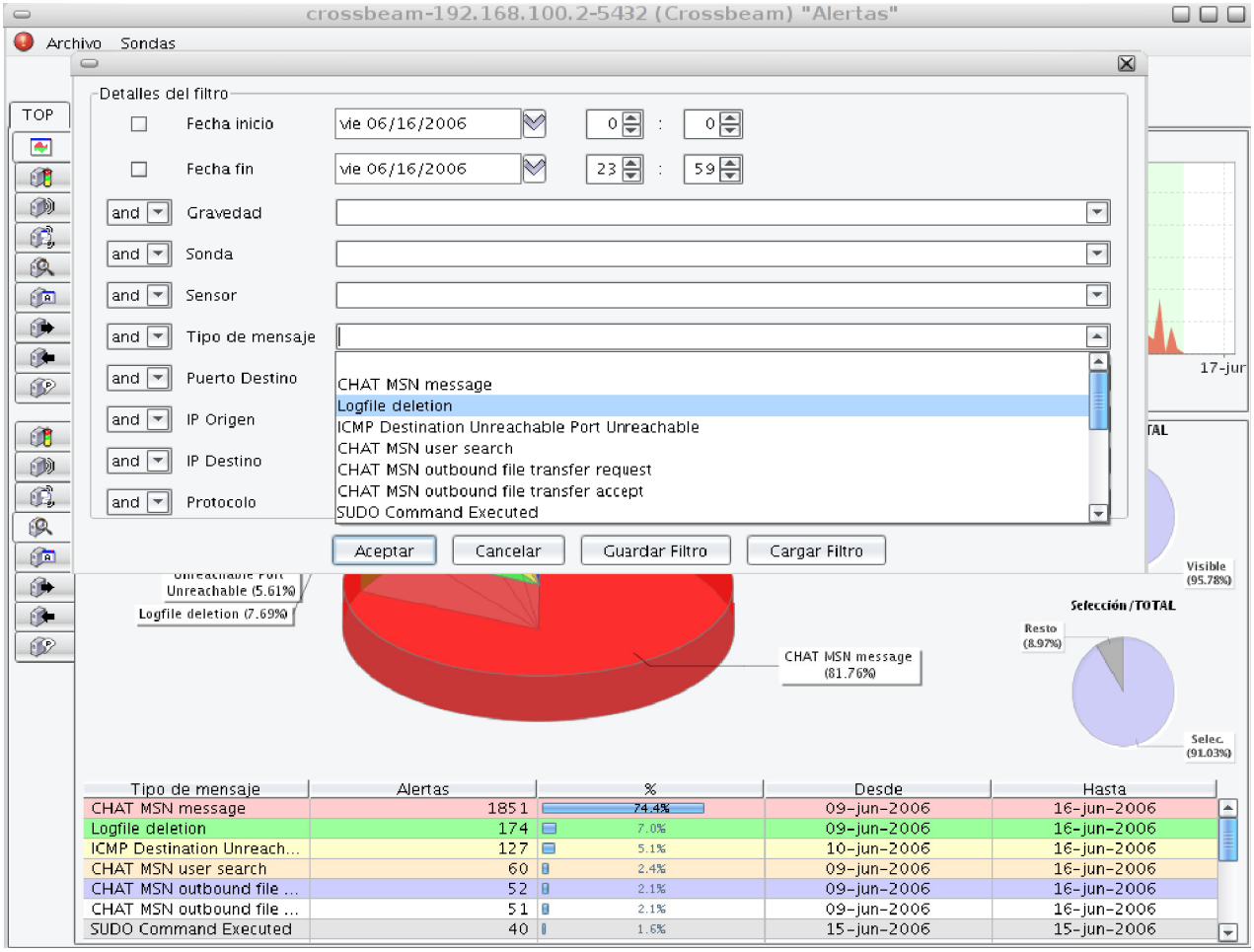


Figura 73.- Opciones facilitadas en los objetos desplegables según los datos de los TopStats.

En el filtro avanzado se puede escribir cualquier fórmula compuesta por operadores lógicos, valores y paréntesis. Aunque la clase *martealertAdvancedFilterView* no calcula todos los *martealertTop*, facilita el filtrado por los campos cuyos *top* ya se hayan calculado, de modo que se despliega una lista con todos los valores cuyas alertas se han producido en el intervalo temporal considerado. Con el fin de simularlo de forma independiente a las clases gráficas, la clase *martealertAdvancedFilterTest* calcula todos los *martealertTop* (la primera columna) para pasar los datos a la ventana del diálogo.

java.awt.BorderLayout, java.awt.GridBagConstraints, java.awt.GridBagLayout, java.awt.Insets, java.awt.event.ActionEvent, java.awt.event.ActionListener, java.awt.event.KeyEvent, javax.swing.AbstractAction, javax.swing.Action, javax.swing.JButton, javax.swing.JCheckBox, javax.swing.JComboBox, javax.swing.JComponent, javax.swing.JDialog, javax.swing.JFrame, javax.swing.JLabel, javax.swing.JOptionPane, javax.swing.JPanel, javax.swing.JSpinner, javax.swing.KeyStroke, javax.swing.SpinnerNumberModel, javax.swing.border.CompoundBorder, javax.swing.border.EmptyBorder, javax.swing.border.TitledBorder, javax.swing.event.ChangeEvent, javax.swing.event.ChangeListener;

8.7.Paquete *martealertUtils*.

Está formado por las utilidades de *martealert*, cuyas clases también podrían alojarse dentro del paquete principal de *martealert*. No obstante, principalmente son clases que aportan utilidad a objetos de otras clases y no al propio objeto que definan, en muchas ocasiones son públicos y estáticos; por tanto no operan sobre el objeto *this* actual.

8.7.1.Clase *martealertCalendar*.

Es una clase que hereda de *java.util.GregorianCalendar* y por tanto también de *Calendar*. Posee muchos métodos para devolver el calendar en formato *Timestamp* y en cadena de caracteres expresando una fecha en notación ISO, por ejemplo "2006-08-01 05:30:12.8" (AAAA-MM-DD hh:mm:ss.d). A continuación se explica brevemente la funcionalidad de los métodos.

- *public Timestamp getTimestamp()*: Obtiene la fecha del objeto *this* en formato Timestamp.
- *public String toDateString()*: Obtiene en formato cadena de caracteres la fecha en el siguiente formato: AAAA-MM-DD, sin especificar la hora. El código de este método es el siguiente:

```
public String toDateString() {
        int auxdate;
        final int month;
        String ret;
        try {
            month = this.get(this.MONTH)+1; //To represent January as month #1.
            ret = ""+ this.get(this.YEAR)+"-"+
                ((auxdate = month)<10?"0":"").toString()+auxdate+"-"+
                ((auxdate = this.get(this.DATE))<10?"0":"").toString()+auxdate;
        } catch (java.lang.NullPointerException ex) {
            ret = "";
        }
        return ret;
    }
```

- *public String toString()*: De manera análoga al anterior método, obtiene la fecha en formato ISO devolviendo una cadena de caracteres a partir del *martealertCalendar this*.

- *public static martealertCalendar add(GregorianCalendar sum1, GregorianCalendar sum2)*: Suma los *GregorianCalendar sum1* y *sum2*, devolviendo el resultado.
- *public static martealertCalendar substract(GregorianCalendar op1, GregorianCalendar op2)*: Resta las fechas *op1* y *op2*, devolviendo el resultado.
- *public static Timestamp getLocalTimeFromGMT(martealertTable tab)*: Obtiene la hora local a partir de la hora en GMT del objeto (0, columna LTS) de la tabla *martealertTable tab*.
- *public static String getCurrentTime()*: Obtiene la hora local del sistema en cadena de caracteres con formato ISO.
- *public static String getPostgreTime(GregorianCalendar time)*: Convierte la fecha pasada con el *Calendar time* a cadena de caracteres en formato ISO, válida para concatenarlas a los comandos de PostgreSQL.
- *public static String compressTime(String timestamp)*: Comprime la notación de la fecha pasada como parámetro suprimiendo los segundos y décimas de segundo. Por ejemplo, haría la siguiente operación: A partir de "2006-11-03 10:03:45.8" se obtiene "2006-11-03 10:03".
- *public static String expandTime(String shortTime)*: Efectúa la operación inversa a la del método anterior.
- *public static String getLTS(Connection con)*: Obtiene la fecha del evento más recientemente recibido.
- *public static String getFTS(Connection con, String LTS, String interval)*: Obtiene la fecha FTS dado un LTS y la anchura del intervalo [FTS, LTS].

8.7.2.Clase *martealertUtils*.

Es una clase formada por los métodos que dotan de funcionalidad general al resto de clases de *martealert*. En su mayoría se trata de métodos estáticos por lo que no es una clase diseñada para instanciar objetos *martealertUtils*. Los métodos que implementa son los siguientes:

- *public static Timestamp minDate(Vector row)*: Calcula la fecha mínima del vector de fechas *row* y lo devuelve en formato *Timestamp*.
- *public static Timestamp maxDate(Vector row)*: Calcula la fecha máxima del vector de fechas *row* y lo devuelve en formato *Timestamp*.
- *protected static String unComile(String s)*: Suprime las comillas en primera y última posición, en caso de que exista y devuelve la cadena resultado.
- *public static int min(int[] tab)*: Calcula el valor mínimo entero del array de enteros *tab* y lo devuelve. Se puede usar para el cálculo de índices y añade funcionalidad a java.lang.Math.min.
- *public static int max(int[] tab)*: Calcula el valor máximo entero del array de enteros *tab* y lo devuelve. Se puede usar para el cálculo de índices y añade funcionalidad a java.lang.Math.max.
- *public static String unconcatNx(String NxClassif)*: Devuelve el tipo de mensaje en cadena de caracteres a partir de la cadena "N x tipo de mensaje" del primer nivel. Por ejemplo: "N x Log_type" pasa a "Log_type".

- *public static int nOcurrences(String str, String c)*: Cuenta y devuelve el número de ocurrencias de la cadena *c* en la cadena *str*.
- *public static String maxSeverity(Vector row)*: Calcula la gravedad máxima de un vector que almacena valores en cadena de caracteres "low", "medium", "high" o "info", siguiendo éste mismo orden.
- *public static boolean isValidIpAddr(String value)*: Devuelve si la cadena pasada *value* tiene el formato de una dirección IPv4 válida o no.

Los siguientes cinco métodos añaden funcionalidad a las clases gráficas para la representación y gestión de mensajes de los filtros avanzados:

- *public static void error(Component parent, String s)*: Muestra por pantalla una ventana con el mensaje de error pasado como cadena *s*.
- *public static void information(Component parent, String s)*: Muestra por pantalla una ventana con un mensaje de información (y su correspondiente icono) pasado como parámetro (*s*).
- *public static int confirmDialogYesNoCancel(Component parent, String text)*: Muestra una ventana de confirmación con los botones: "Sí", "No", "Cancelar". Existe una sobreescritura de este método para que se muestre únicamente "Sí" o "No".
- *public static void dialog(Component parent, String s)*: Muestra por pantalla un diálogo pidiendo una entrada proviniente del componente padre. Para nuestro caso, sería un diálogo abierto desde la ventana "padre" que para nuestro filtro avanzado es el diálogo de la figura 71. Se trata de un diálogo genérico al que también se el puede especificar un valor usado para inicializar la entrada.
- *public static void beep()*: Genera un sonido "beep".

Éstos últimos tres métodos estáticos añaden funcionalidad a las cadenas de caracteres y Vectores.

- *public static String uniSpaced(String s)*: Devuelve la cadena pasada como parámetro con un único espacio entre palabras, sin introducir espacios entre paréntesis y variables. Ejemplo: Si se pasa la cadena `s = " ( palabra1 palabra2 palabra3 palabra4) "` se devuelve la siguiente: `"(palabra1 palabra2 palabra3 palabra4)"`
- *public static String unSplit(Vector v, String separator, boolean includesep)*: Realiza la operación de concatenación de los elementos del vector v, separados por la cadena *separator* la cual se incluirá o no según sea "true" o "false" la variable *includesep*.
- *public static Vector toVector(String[] st)*: Convierte una tabla de cadena de caracteres en un Vector.

8.7.3.Clase *martealertGlobals*.

Es la clase cuyos métodos permiten la utilización de un diccionario. En la tabla estática bidimensional *languages* se definen los idiomas de la siguiente manera:

```
public static String[][] languages = {
        {"spanish" ,"Espa\u00F1ol" ,"es","ES"},
        {"english" ,"English" ,"en","UK"}
}
```

Por cada idioma definido deberá haber un fichero en la carpeta martealert/language/martealertLanguage_es_ES.java, martealertLanguage_en_UK.java... etc.

Los métodos fundamentales de esta clase son:

- *public String getValue(String key)*: Obtiene el valor del diccionario para la clave *key*, es decir, obtiene el valor traducido el índice *key*.
- *public static String i18n(String key)*: Método análogo al anterior pero permite el uso de forma estática, sin instanciar objetos *martealertGlobals*.
- *public static void setLanguage(String language)*: Asigna un lenguaje de modo que se toma el diccionario definido en la clase *martealertGlobals*.

Se ha visto anteriormente que las peticiones SQL que generan las *martealertTable*, se conforman a partir de varios métodos. El método *getSqlVars* (tanto para los *tops* como para los *FirstLevel, RealTime...*) devuelve una cadena en la que cada variable sigue el siguiente formato:

```
"tabla1.variable1 AS \""+martealertGlobals.i18n("Nombre variable 1")+"\";
```

Para la selección del diccionario es necesario que en la clase ejecutable principal, se asigne un diccionario existente como primera línea del método *main*:

```
public static void main(String args[]) throws java.lang.OutOfMemoryError {
        martealertGlobals.setLanguage("spanish");
        ...
```

8.7.4.Clase *martealertSqlUtils*.

Es una clase pensada para proveer de funcionalidad a las clases del paquete *martealert* y no para instanciar un objeto del tipo *martealertSqlUtils*. Por tanto, al igual que con la clase *martealertUtils* está conformada por métodos estáticos que generan los comandos SQL o interaccionan con la base de datos creando y cerrando la conexión. Estos métodos se explican brevemente a continuación:

- *public static Connection openSQLConnection(String usname, String passwd, String ipaddr, int port)*: Clase que crea una conexión usando *jdbc* con la base de datos de PostgreSQL definida en la constante DATABASE. Además se especifica el nombre de usuario, la dirección IP de la máquina en la que se encuentra la base de datos y el puerto en el que escucha el servidor *postmaster*.

```
public static Connection openSQLConnection(String usname, String passwd,
                                           String ipaddr, int port) {
        Connection con = null;
        String dbname=DATABASE;
        final String url = "jdbc:postgresql://"+ipaddr+":"+
                           java.lang.String.valueOf(port)+"/"+dbname;
        final String driver = "org.postgresql.Driver";

        try {
            Class.forName(driver).newInstance();
        } catch(Exception e) {
            System.out.println("Failed to load postgreSQL driver");
            return null;
        }
```

```
        try {
            con = DriverManager.getConnection(url,usname,passwd);
        } catch(Exception e) {
            e.printStackTrace();
        }
        return con;
    }
```

Todos los métodos que se detallan a continuación deben de lanzar la excepción **java.sql.SQLException**, y que se capture en las clases que usen esta librería.

- *public static void closeSQLConnection(Connection con)*: Cierra la conexión con la base de datos.
- *public static void addColumn(Connection con, String tableName, String newColumnName)*: Añade la columna de nombre *newColumnName* a la tabla *tableName*.
- *public static int getIdFromTime(Connection con, String time, String timetype)*: Obtiene el identificador de la alerta que tenga por fecha *time*. La cadena *timetype* puede ser "FTS" ó "LTS".
- *protected static int getLastId(Connection con)*: Obtiene el identificador de la alerta más reciente.
- *public static String getLastJoinTable(martealertHashFilter hf)*: Obtiene la tabla para componer el comando SQL que haga la última operación de *join*.
- *public static String getTopJoinTAble(martealertHashFilter hf)*: Obtiene la tabla con las alertas cribadas por el filtro *hf*.
- *protected static String getSqlCommand(martealertHashFilter hf)*: Obtiene la parte del comando SQL a partir de la cláusula "FROM" incluyendo tablas, agrupado, ordenado, limitado y filtrado a excepción únicamente de las variables del "SELECT".
- *protected static String getSqlJoinTable(String field, martealertHashFilter hf)*: Obtiene una cadena para añadir al comando SQL una operación de *join* con la tabla que contiene las variables del campo *field*, filtrada por el *martealertHashFilter hf*.