CAPITULO 5

Implementación y Pruebas

Este capítulo muestra como se implementaron al sistema las diferentes herramientas para la realización de GeoMVisio. Se muestra primero el modelado del sistema donde pueden observarse las clases que se crearon y como se integraron en módulos. Una vez conociendo los módulos se procede a mostrar la implementación de algunas funciones importantes para dar una mayor idea de como se resolvieron los problemas. Se muestra como se integran las tecnologías Grappa y la aplicación Dot al sistema para cumplir con los objetivos de la tesis.

5.1 Modelado

Esta tesis cuenta con un conjunto de clases que en conjunto cumplen los requerimientos de GeoMVisio. En la Figura 5.1 se muestra el diagrama de clases donde se muestra la integración de todas las clases involucradas. GeoMVisio cuenta con dos módulos principales los cuales son responsables de realizar las funciones para cumplir la misión de esta tesis y están organizados según su uso. El primer módulo se encarga de la lectura de los archivos de entrada así como la creación de los grafos con que se trabajará. El segundo módulo se encarga de la visualización de las subestructuras y de la interacción con las imágenes mostradas procedentes de información que ya fue cotejada por el primer módulo.

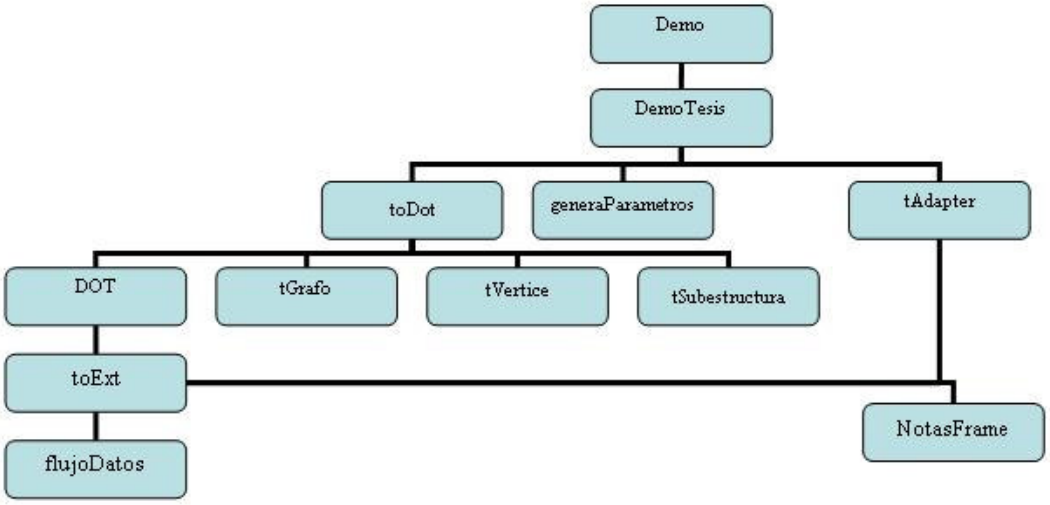

Figura 5.1
Diagrama de Clases

Las clases Demo y DemoTesis son las más importantes, mas en concreto la clase DemoTesis pues se encarga de orquestar todos los procesos y funciones de las demás clases y esta relacionada con ambos módulos de GeoMVisio. En esta clase interviene todas las acciones y se crea toda la interfaz visual es decir todos los paneles, áreas de texto entre otros. La clase Demo es una clase que fue creada debido a una limitante de Grappa ya que se tiene mandar un grafo de entrada de lo contrario provoca errores. El resto de las clases se muestran durante la explicación de los módulos de GeoMVisio en las siguientes subsecciones.

5.1.1 Módulo de Creación y Almacenado

Este módulo se encarga de crear los grafos que representan las subestructuras, así como obtener la información procedente del archivo de entrada de GeoMVisio. Este módulo esta basado en uno creado en la tesis doctoral de Manuel Pech Palacio con algunas modificaciones. Básicamente los

cambios son que el módulo en la tesis doctoral de Manuel Pech Palacio esta integrado de clases que incluso crean la interfaz visual que interactúa con el usuario y en GeoMVisio son clases de apoyo. Los demás cambios que tuvieron que hacerse están relacionados directamente con el comportamiento de GeoMVisio y de Grappa.

El módulo creado para GeoMVisio se muestra en la Figura 5.2 que consta de seis clases las cuales se explican a continuación.

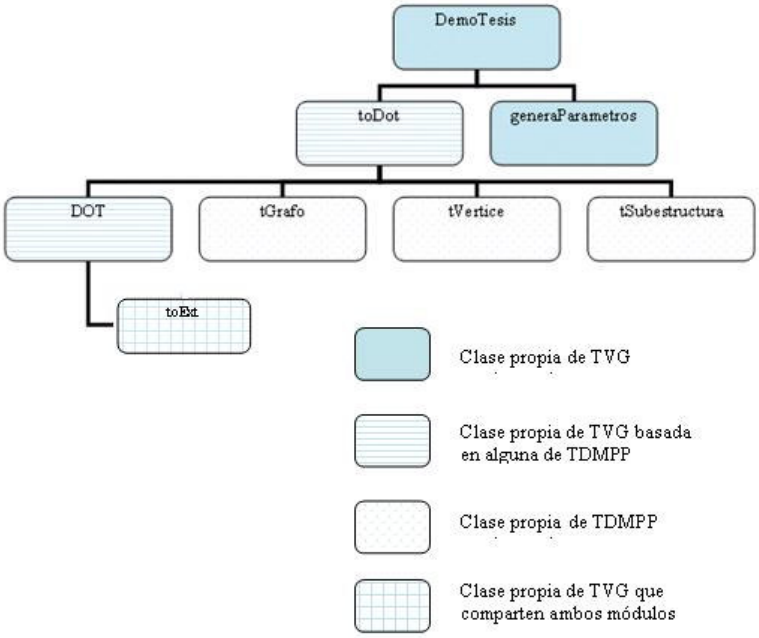

Figura 5.2

Módulo de Creación de Archivos

Clase *ToDot*, es la más importante de este módulo pues organiza y manda llamar al resto de las clases del módulo. Se encarga de leer el archivo de entrada para identificar cuantas instancias, subestructuras, subestructuras

predefinidas, vértices y arcos conforman la información del archivo de entrada. Esta clase está basada en la clase SUBDUE de la tesis doctoral de Manuel Pech Palacio que básicamente hace lo mismo con respecto al manejo de subestructuras.

Los cambios realizados son principalmente que se eliminaron funciones de la clase SUBDUE que no servían para GeoMVisio como una conexión a una base de datos, los elementos visuales que interactúan con el usuario, entre otros. El cambio más importante para adaptarlo a GeoMVisio fue cuando se eliminaron las opciones que crean los subgrafos de las subestructuras encontradas que en este caso hacen el papel de hipergrafo pero éstos no tienen interacción alguna ya que se crea una simple imagen jpeg.

La clase Dot se encarga de traducir las subestructuras en sintaxis de Subdue a lenguaje Dot para crear los subgrafos con que se van a trabajar. Esta clase está basada en la clase Dot de la tesis doctoral de Manuel Pech Palacio. Dot presenta cambios más significativos con respecto a Dot, que los que hubo de SUBDUE a toDot, principalmente al momento de trabajar con las subestructuras predefinidas y en relación a la creación de subgrafos.

La clase generaParametros fue creada para GeoMVisio y lee el archivo de entrada para buscar los parámetros con que trabajó Subdue para la búsqueda de patrones. Los parámetros se almacenan en un archivo de texto para su posterior extracción y presentación.

La clase *toExt* se encarga de asignar las posiciones de los elementos en los grafos a visualizar, para esto se apoya de la aplicación *Dot*. Esta clase también se encarga del la creación de los archivos en formato de imagen, una de las funciones del módulo de visualización.

Las clases *tGrafo*, *tVertice* y *tSubEstructura* son tres estructura de datos que representan cada elemento de los grafos. Estas clases están contenidas en la utilería que usa MPP para la tesis doctoral de Manuel Pech Palacio. No tiene ninguna modificación y son necesarias debido a que Dot y toDot hacen uso de estas.

5.1.2 Módulo Visualización e Interacción.

Este módulo se encarga de la parte central de la tesis que es la visualización de las subestructuras encontradas por Subdue, también comprende las funcionalidades de GeoMVisio que se pueden realizar los grafos. Para este objetivo se cuenta con tres clases en este módulo y una más que comparte con el módulo de creación y almacenamiento que integradas se muestran en la Figura 5.3 donde se puede ver la estructura de este módulo. Las clases se explicaran a continución.

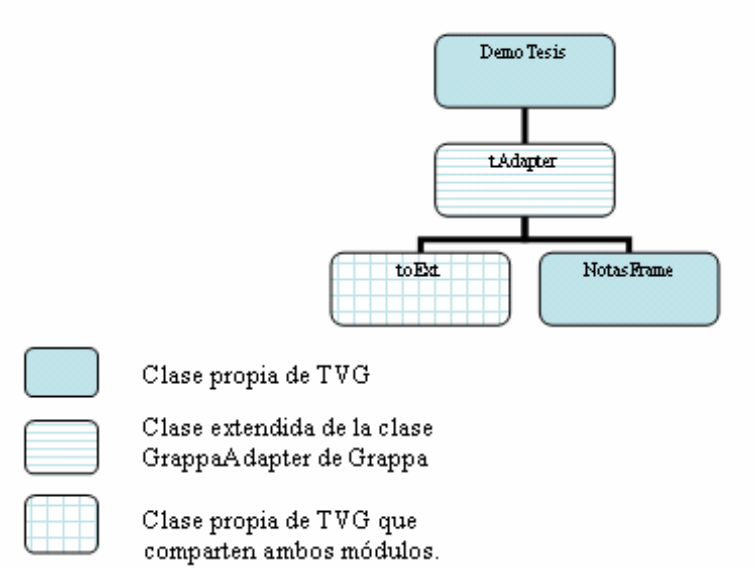

Figura 5.3

Módulo Visualización

La clase *tAdapter* esta se encarga de la interacción con los hipergrafos para su visualización, también realiza las opciones a las imágenes por ejemplo imprimir, agregar notas, la conexión con InterGeoM y conversión a jpg. Esta clase es una extensión de la clase GrappaAdapter de Grappa. Esta clase reescribe en su totalidad el método GrappaClicked de GrappaAdapter que se encarga de las acciones a realizar en caso de que se interactúe con el Mouse.

NotasFrame ésta clase se encarga de mostrar un marco para agregar un comentario si el usuario así lo desea. Esta clase hace uso de Grappa para modificar la información del grafo y agregar la nota por medio del lenguaje Dot creando un atributo nuevo referenciado para cada nota en el grafo. Una vez modificado el grafo se reescribe el archivo Dot referenciado con la información adicional de las notas. Esta opción originalmente pertenece a InterGeoM, pero

GeoMVisio hace uso de este para poder implementar esta función de tal manera que si se agrega una nota ya sea por parte de GeoMVisio InterGeoM puedan verse en ambos sistemas.

La clase toExt crea las imágenes en formato jpg. Para ello usa la información del grafo visualizado en ese momento y nuevamente se usa la aplicación *Dot* que crea automáticamente la imagen en formato jpg por medio del comando asignado para este caso.

5.2 Implementación

Como se puede observar en el modelado de nuestro sistema, este se compone de dos módulos principales. Sin embargo, estos dos módulos comparten clases para cumplir sus objetivos, para la implementación tales clases se van a separar para ser integrados en un paquete llamado GeoMVisio y se van a crear otros dos paquetes referentes a los dos módulos mencionados. En la Figura 5.4 se muestra el diagrama de paquetes y como se integran.

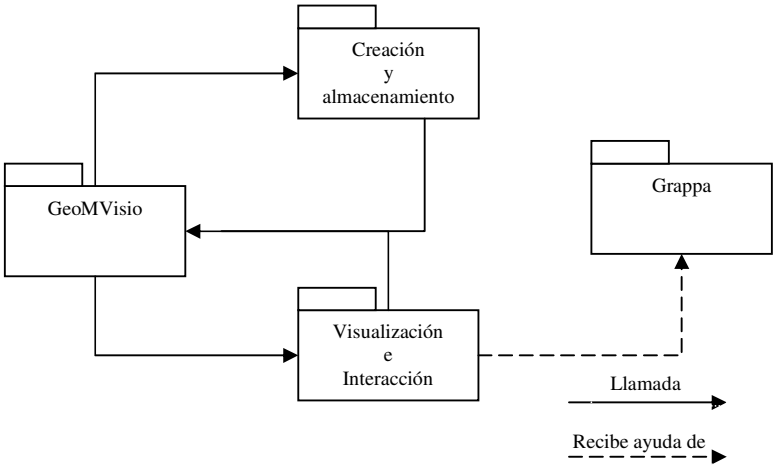

Figura 5.4
Diagrama de Paquetes

En el diagrama de paquetes se puede observar la interacción que existe entre éstos, donde tanto el paquete de Creación y Almacenamiento y el de Visualización e Interpretación mandan llamar al paquete GeoMVisio y viceversa. En tanto que el primer paquete de Visualización manda llamar al paquete Grappa para realizar algunas funciones referentes para la visualización.

A continuación se explica como se implementaron en Java dichos paquetes y se muestra como realizan algunas funciones de una manera orientada a la programación.

5.2.1 Paquete Creación y Almacenamiento

Esta subsección muestra como se implementó en el sistema la abstracción de la información de las subestructuras desde un archivo de salida de Subdue, que es el archivo de entrada del sistema de GeoMVisio. Este paquete esta formado por tres clases toDot, Dot y generaParametros.

La lectura de un nuevo archivo de entrada la lleva a cabo la clase toDot por medio de los métodos *cargaArchivo* y *analizaLinea*. Éstos son mandados llamar por la clase DemoTesis y son activados solamente cuando se ingresa un archivo de entrada nuevo, en la Figura 5.5 se puede observar la estructura de la clase.

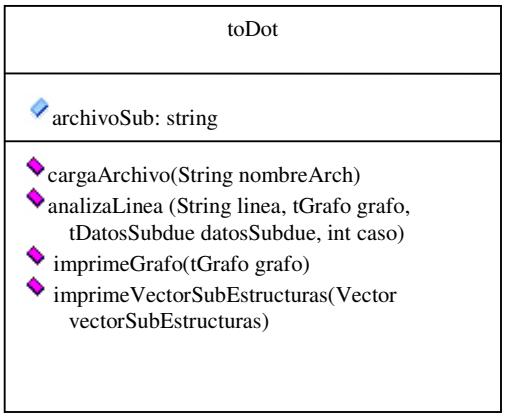

Figura 5.5

Clase toDot

La clase *toDot* se encarga de leer el archivo de entrada del sistema para abstraer las subestructuras y almacenarlas. Cabe recordar que esta clase es una modificación de la clase SUBDUE de la tesis doctoral de Manuel Pech Palacio.

toDot por medio del método *cargaArchivo* lee el archivo de entrada en formato Subdue. Primero revisa si en el archivo se encuentra escrita la línea que indica si existe solo una iteración, para después buscar la información que indique si contiene información y crea una instancia para cada elemento del grafo es decir subestructuras, vértices y arcos.

En caso de que existan más iteraciones se busca la línea que representa a cada iteración y crea una instancia subestructura para cada iteración. Después sigue leyendo el archivo y busca la información que le indique si es una subestructura dentro de una iteración y crea una instancia de la clase que representa los grafos. Una vez identificados los grafos empieza a buscar los vértices y arcos que lo conforman. De igual manera busca la información referente a las subestructuras predefinidas identificándola primero como tal.

La clase Dot, como ya se ha mencionado, se encarga entre otras funciones de la traducción de la sintaxis de Subdue al lenguaje Dot, así como la asignación de los nombres de los archivos que contienen los grafos a visualizar. Hay que recordar que esta clase es una modificación de la clase Dot de la tesis doctoral de Manuel Pech Palacio. La estructura de esta clase se puede observar la Figura 5.6.

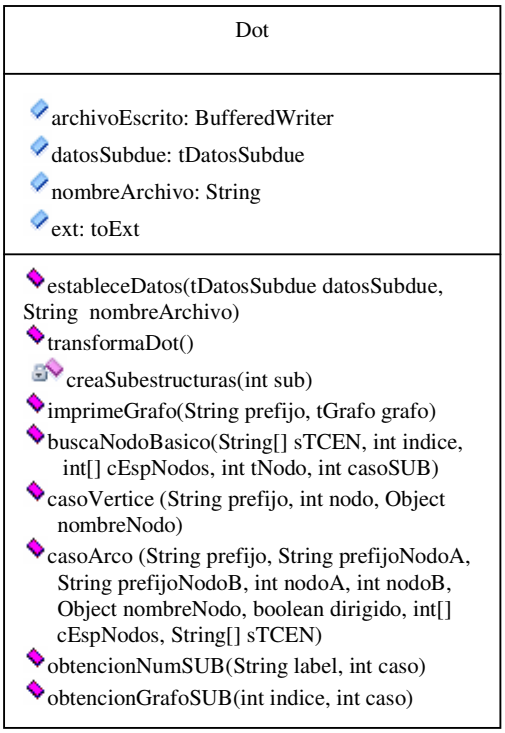

Figura 5.6

Clase Dot.

Una de las funciones importantes de Dot es diferenciar si una subestructura es predefinida (nos ayudan a guiar la minería) o normal (aquellas que descubre la minería), el método que se encarga de esta acción es el de *creaSubestructura*. Separa las subestructuras por medio de un número procedente de la clase toDot, en caso de que sea un cero la subestructura es normal, de ser uno entonces es una subestructura predefinida.

Después de haber definido el tipo de subestructura este mismo método crea el nombre con el cual se identificara cada grafo. El nombre es estructurado de manera que indique el nombre del archivo del cual precede, la iteración en que

fue encontrado la subestructura y finalmente el número de subestructura reportada en la iteración.

Finalmente la función más importante de esta clase también realizada por el método *creaSubestructura* es la de la traducción al lenguaje Dot. Esto se hace por medio de un tipo de plantilla la cual está en formato del lenguaje Dot y representa a cada una de las subestructuras. Este método se apoya en otro método llamado casoCuerda que recibe la cadena de caracteres procedente de la plantilla para almacenar los grafos.

En este caso podemos observar como se crea un grafo en lenguaje Dot con la información de la subestructura y se almacena en un grafo temporal el cual solo tendrá la información de los elementos sin posiciones. Aquí Dot hace uso de la clase toExt para mandar llamar a la aplicación Dot mandándole como parámetros un vector con todos los grafos temporales para asignarles posiciones a los diferentes elementos de los grafos. Los grafos temporales después serán eliminados.

La clase generaParametros también lee el archivo de entrada pero en este caso para buscar los parámetros con que fue implementada la fase de minado de datos en Subdue. Esto lo hace almacenando la información en la sección de “paramter” de los resultados presentados por Subdue. Una vez identificada la información la almacena en un archivo de texto cuyo nombre esta formado con el nombre del archivo del cual procede más el sufijo par que nos indica que

contiene los parámetros, por ejemplo "*house3it3par.txt*". La Figura 5.9 muestra como esta integrada esta clase.

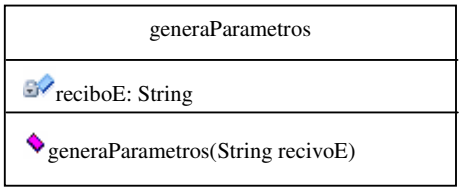

Figura 5.9

Clase generaParametros.

5.2.2 Paquete de Visualización e Interacción.

Este paquete esta formado por dos clases NotasFrame y tAdapter. Este paquete realiza parte de las funciones más importantes de GeoMVisio, entre ellas se encuentra el hipergrafo.

La clase tAdapter realiza la acción del hipergrafo que permite básicamente expandir el grafo raíz o contraerlo según sea el caso. Los hipergrafos son parte primordial de esta tesis y para realizar esta función esta clase funciona de la siguiente manera. Primero, cuando es seleccionado un elemento de la imagen con el Mouse por medio del método GrappaClicked, la clase verifica si es un vértice, una vez identificado como vértice comprueba si su etiqueta esta estructurada con el prefijo *SUB_* o PS_SUB y si el sufijo tiene un número entonces es considerado un hipergrafo.

Al saber que es un hipergrafo y para poder expandirlo, ese vértice se elimina para suplirlo por un subgrafo que contenga la información de la subestructura relacionada, pero agregar el subgrafo no es una acción simple pues puede

generar en el grafo expandido información fantasma si no se hace correctamente. En esta acción los métodos involucrados son *nuevoGraph*, *formatearSubgraph*, *nodoLabel*, *remplazar, getFirstNode*, *getLastNode*. La estructura de esta clase se puede observar en la Figura 5.8.

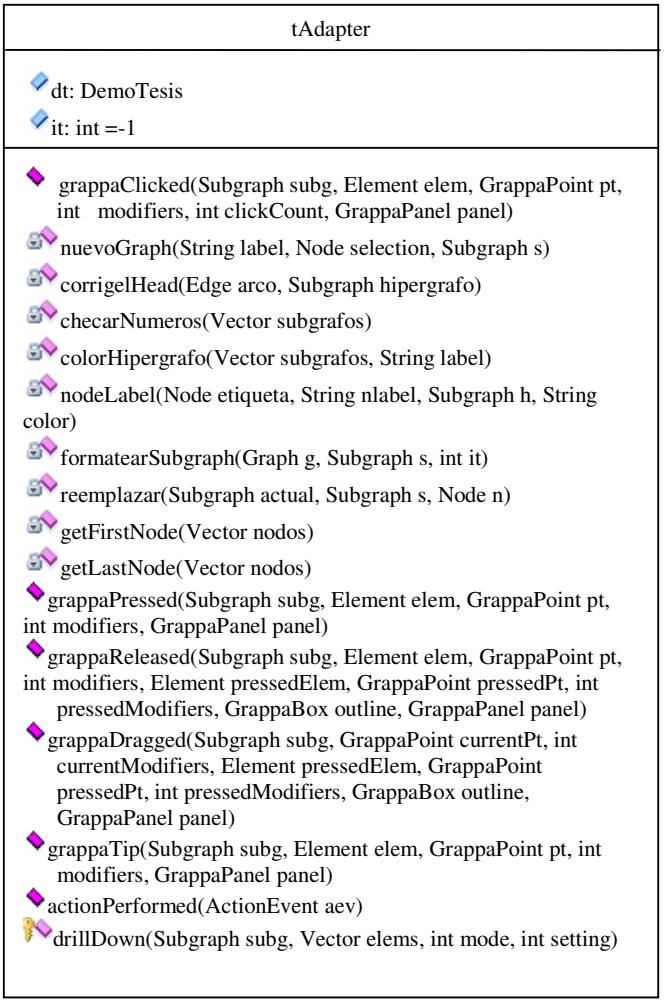

Figura 5.8

Clase tAdapter.

Para realizar esta tarea elimina el vértice hipergrafo y todos los arcos que contenga ya sea que estén dirigidos al vértice o salgan de él. Ya eliminados el vértice y los arcos, se añade el subgrafo que contenga la información de la

subestructura referenciada al grafo raíz con un color para su distinción. Para añadir el subgrafo la información de los arcos del vértice desaparecido se usa para crear nuevos arcos pero ahora relacionando al nuevo subgrafo. El método que se encarga de agregar el hipergrafo es *remplazar*.

De manera contraria, se realiza la compresión del grafo anidado para que quede en su estado original, eliminando primero al subgrafo y agregando un nuevo vértice hipergrafo. Esta acción no es tan complicada como la expansión del hipergrafo ya que no hay necesidad de manipular el vértice nuevo solamente se agrega y se elimina el subgrafo. Todo esto se hace en un grafo temporal que se usa para mostrar los grafos extendidos sin tener que modificar el grafo original.

La acción compresión del grafo la realiza el método GrappaClicked, sin embargo elimina los arcos relacionados con el subgrafo que se tienen que quitar sigue el mismo procedimiento que se muestra para la expansión pero en este caso la información de los arcos a eliminar esta relacionada con el subgrafo a remover, además no es necesaria integrar la información al nuevo vértice ya que este se agrega de manera directa a la subestructura.

Al hipergrafo como se mencionó se le asigna un color para poder ser distinguido o conjuntado para mayor claridad de la imagen observada. La asignación del color es asignado por el método *colorHipergrafos*. Para la asignación de colores hace uso de ocho colores para grafo normales y cuatro colores para grafos de subestructuras predefinidas.

La manera en que *colorHipergrafos* asigna el color depende de la iteración o la importancia del grafo predefinido a la cual pertenezca el hipergrafo representado por el ultimo numero de la etiqueta del vértice, de ahí ese numero es evaluado por la operación matemática módulo del paquete java.Math para obtener el color que se asigna al hipergrafo. Por lo tanto, hipergrafo diferentes siempre tendrán colores diferentes e hipergrafos relacionados tendrán el mismo.

Entre el resto de las funciones que presenta la clase tAdapter es la de la impresión de la imagen actual, tal como se muestra el grafo en el panel de visualización, para obtener el grafo que imprime se apoya de la clase de GrappaPanel que cuenta con un método llamado *getGraph* que obtiene este del panel de visualización. Ya obtenido el grafo se manda al formato de impresión ya sea por medio de una impresora o en la creación de una imagen en formato jpg. Para la creación de las imágenes tAdapter le manda el grafo obtenido a toExt para que este realice tal acción.

El método getGraph de GrappaPanel fue de gran ayuda pues permitió hacer la creación de la imagen así como la manipulación de los grafos. Sin embrago, se tiene que crear una instancia de la clase Graph de Grappa para poder almacenar el grafo que recupera este método y después se trabaja con esa instancia.

Por último la clase NotasFrame, que es parte de InterGeoM, permite agregar un comentario en un grafo. Para esto la clase nuevamente hace uso de las funcionalidades de Grappa. Sin embargo aquí solamente se muestra un textArea donde se ingresa la nota, la funcionalidad de las notas lo realiza DemoTesis que se mostrará en seguida.

5.2.3 Paquete GeoMVisio

Este paquete está formado por las clases DemoTesis y toExt ya explicadas en la sección de modelado. Lo interesante de estas clases es el funcionamiento que éstas realizan, primero se explicara los puntos mas trascendentales de la función de DemoTesis.

Entre las funciones más importantes que hace DemoTesis esta la entrada de los archivos de entrada para trabajar en el sistema, una vez seleccionado tal archivo se manda llamar al método *proc_uno* que se encarga de mandar llamar a todas las clases y métodos involucrados para la manipulación de la información contenida en el nuevo archivo. La Figura 5.9 muestra a esta clase que es la que se encarga de orquestar todas las funciones del sistema.

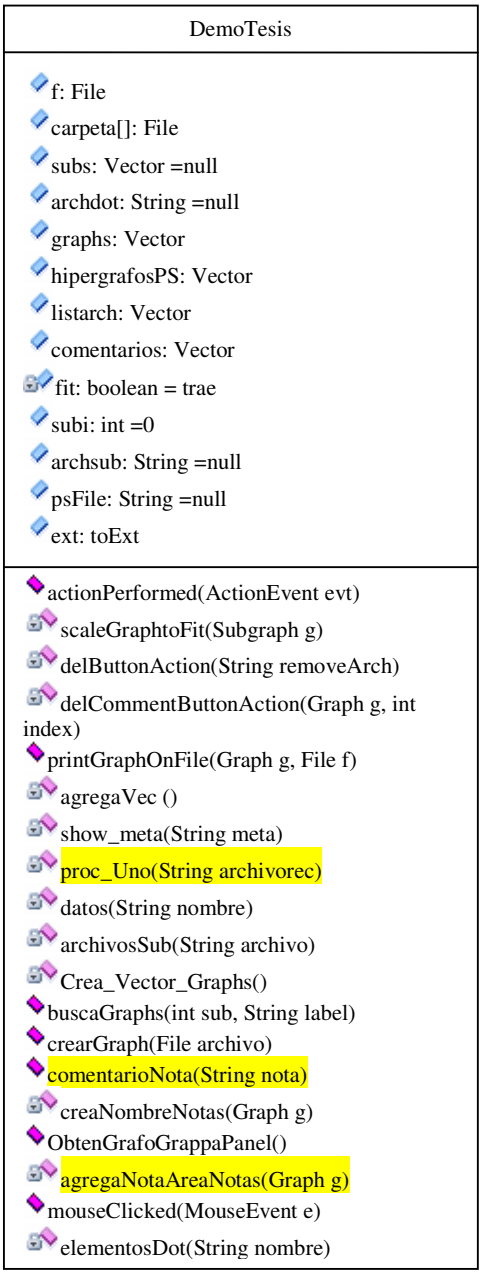

Figura 5.9

Clase DemoTesis

Otra función de esta clase es la relacionada con las notas que contenga el grafo de la subestructura que se visualiza en un momento dado. En la figura superior se muestra el método de DemoTesis que se encarga del la integración de las notas que es *comentarioNotas*. Esta función permite al usuario

interactuar con las notas ya sea eliminándolas o modificándolas según su conveniencia, también se encuentra relacionado con este trabajo el método *agregaNotaAreaNotas* que como su nombre lo indica se encarga de mostrar las notas al usuario por medio del panel indicado para esto.

La clase toExt es la que se encarga de mandar llamar la aplicación Dot, ya sea para asignar posiciones a los elementos de los grafos creados en toDot o crear los archivos imágenes en formato jpg que son solicitadas desde tAdapter. Esta clase cuenta con dos constructores debido a que se puede mandar un solo archivo Dot o un vector de estos archivos como parámetros. Una vez diferenciando que tipo de salida es la que se desea (*Dot* o *jpg*) se crea una instancia de la clase Runtime para crear un proceso que haga uso de la aplicación Dot y poder crear las salidas. Este trabajo lo realiza el método *run* de esta clase. La organización de esta clase se observa en la Figura 5.10

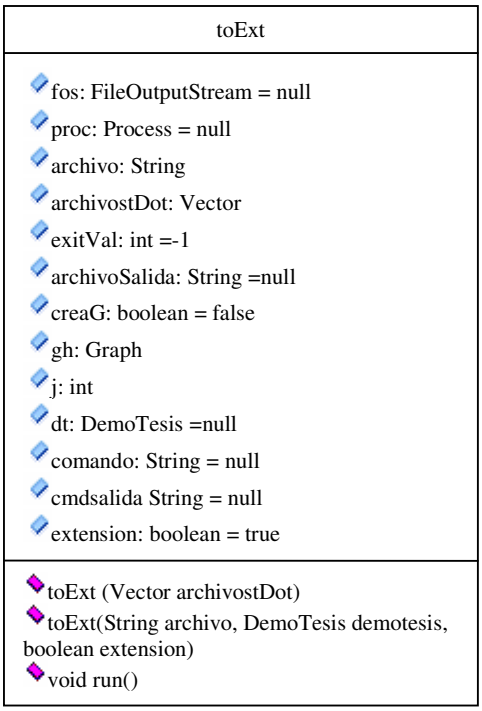

Figura 5.10. Clase toExt.

En este capítulo se mostraron las funciones de GeoMVisio más a fondo, explicando su implementación. También se mostró el modelado de GeoMVisio y la estructura de los procesos que realiza esta tesis. Esto da una idea más clara el trabajo que se realizó para crear el software de GeoMVisio.