Universidad de las Ciencias Informáticas

Facultad 6

IMPLEMENTACIÓN DEL MÓDULO DE VISUALIZACIÓN DE DATOS GEOLÓGICOS PARA EL SISTEMA SYAM.

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE
INGENIERO EN CIENCIAS INFORMÁTICAS

AUTOR: YUNIER LÓPEZ MARTÍNEZ

TUTOR: DAGOBERTO ANTONIO SUÁREZ MORALES

La Habana, junio de 2013
"Año 55 de la Revolución"

"Los que no tienen el valor de sacrificarse han de tener a lo menos el pudor de callar ante los que se sacrifican, o de elevarse en la inercia inevitable o en la flojedad, por la admiración sincera de la virtud que no alcanza. Debe ser penoso inspirar desprecio a los hombres desinteresados y viriles."

José Martí

A mis padres

DECLARACIÓN DE AUTORÍA

Declaro por este medio que yo, Yunier López Martínez con carné de identidad 89052543740, soy el autor de este trabajo y que autorizo a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio, así como los derechos patrimoniales con carácter exclusivo.

Para que así conste, firma la presente declaración jurada de autoría en La Habana a los ____ días del mes _____ del año _____.

_____	_____
Yunier López Martínez	Dagoberto Antonio Suárez Morales
Autor	**Tutor**

DATOS DE CONTACTO

Nombre completo: Yunier López Martínez

Correo electrónico: yunier0525@gmail.com

Teléfono: 01 – 52260506

Dirección particular: C/ San Luis #226 La Caridad, Palma Soriano, Santiago de Cuba, Cuba.

Código postal: 92610

RESUMEN:

En Cuba, la informatización de todos los sectores de la sociedad es de vital importancia para así lograr productividad, eficiencia y crecimiento económico. El sector geólogo-minero en el país tiene un peso significativo en la economía, por lo que se hacen esfuerzos por informatizar todos sus procesos. Hasta la fecha no se ha logrado que ninguna solución informática cubana remplace por completo la dependencia al *software* propietario usado en la actualidad. No obstante se trabaja actualmente en el desarrollo de un sistema para el análisis y modelado de yacimientos minerales, con el cual se pretende sustituir por completo el uso de las aplicaciones extrajeras contratadas.

El siguiente trabajo titulado "Implementación del módulo de visualización de datos geológicos para el sistema Syam" tiene entre sus objetivos proveer al sistema una forma de visualizar en forma de imágenes los datos geológicos con los que trabaja el mismo. La presente investigación consistirá en la realización de las actividades para el rol de implementador, teniendo como entrada el análisis y diseño realizado con anterioridad. Se muestra además como resultado una aplicación de escritorio funcional en la que estará integrado un módulo para la visualización tridimensional.

PALABRAS CLAVE:

Objetos geológicos, proyección, renderizar, visualización tridimensional.

ÍNDICE DE CONTENIDOS

ÍNDICE DE TABLAS

INTRODUCCIÓN

En la actualidad la minería es una de las actividades económicas fundamentales que impulsan el desarrollo de los países con recursos minerales. Para explotar un recurso mineral los especialistas se apoyan en el empleo de diferentes técnicas como la recopilación de información, teledetección, geología, geoquímica, sondeos mecánicos y geofísica. Estas técnicas han evolucionado con el tiempo y en la actualidad generan gran cantidad de información, la cual debe ser procesada por los especialistas mineros.

Con el advenimiento de las Tecnologías de la Información y las Comunicaciones (TICs) se dio paso a la utilización de un conjunto de sistemas especializados en la planificación de la explotación minera. Esto revolucionó la industria minera agilizando los procesos y aumentando la calidad y precisión de las planificaciones, al permitir procesar grandes volúmenes de datos en un corto lapso de tiempo.

En la actualidad hay productos líderes en el campo de la planificación de la explotación minera tales como: DATAMINE, DATASTREAM 7I, GEMCOM GEMS, HONEYWELL, MAPTEK, MINCOM y WENCO, desarrollados y comercializados por Datamine Latin America S.A, Datastream, Gemcom Software Internacional, Honeywell, Maptek, Mincom y Wenco International Mining Systems respectivamente. Su aplicación abarca todos los aspectos geológicos y de planificación minera, tales como: la estimación de reservas, modelado de relieves y levantamiento topográfico. Estas herramientas, además, pueden traducir un gran volumen de datos a imágenes en tres dimensiones (3D), facilitando así una mejor comprensión y entendimiento por parte de los especialistas.

En Cuba la industria minera en la década del 90 introduce el uso de soluciones informáticas para la planificación minera. Actualmente su utiliza software privativo de empresas extranjeras tales como: GEMCOM GEMS, GEMCOM SURPAC, DATAMINE y MICROLIN, teniendo que invertir en el pago de licencias que autoricen su uso legal. Dado el gasto adicional por el uso de estas aplicaciones surge la necesidad del desarrollo de soluciones informáticas para la planificación minera en el país.

Desde entonces en el país se han desarrollado algunas herramientas informáticas tales como: TIERRA, SIM, CORTE y MICRONIQ. El desarrollo de las mismas ha estado a cargo de instituciones

como Universidad de Pinar del Río, Instituto Superior Metalúrgico de Moa (ISMM), Industrias del Níquel, Empresas Geomineras y el Instituto Superior Politécnico José Antonio Echeverría (ISPJAE). De las mencionadas soluciones hay un factor que ha incidido en la aceptación de estas por los especialistas y es que brindan pocas funcionalidades para el manejo y representación gráfica de los datos geológicos con los que se trabajan.

Con el objetivo de solventar las deficiencias de las soluciones cubanas y de sustituir importaciones, desde hace dos años en la Universidad de la Ciencias Informáticas (UCI), en el proyecto Sistema minero cubano (SMC) del centro GEySED, se trabaja en el desarrollo de la herramienta informática ***Sistema de análisis y modelado de yacimientos minerales*** (SYAM) para la planificación minera, que cumpla con las especificidades y funcionalidades que buscan los especialistas cubanos en los software extranjeros.

Dado que la herramienta está en desarrollo, actualmente la forma en que se muestran los datos es mediante tablas recuperadas de una base de datos, por tanto el análisis e interpretación de la información se hace muy engorrosa, ya que no se tiene una visión clara de la forma real del yacimiento mineral con solo analizar las tablas de la base de datos; influyendo así en la toma de decisiones de la empresa que pudiera hacer uso del sistema en su estado actual.

En el proyecto SMC, utilizando la metodología de *software* Proceso Unificado de Rational (RUP), se realizó el análisis y diseño para el desarrollo de un módulo que permita la traducción de la información contenida en la base de datos a imágenes tridimensionales.

Actualmente el proyecto SMC cuenta con los modelos de análisis, diseño y prototipos para el desarrollo de un módulo para la visualización tridimensional de datos geológicos, pero el mismo aún no es funcional. Con el fin de dar solución a la situación descrita se plantea el siguiente **problema a resolver**: ¿Cómo lograr las funcionalidades del diseño propuesto por el analista del sistema para el módulo de visualización del sistema Syam?

La investigación tiene como **objeto de estudio** el proceso de implementación de sistemas para la visualización tridimensional de objetos geológicos enmarcado en el **campo de acción** las técnicas de representación y modelado tridimensional de objetos geológicos en el sistema Syam.

Para dar solución a los problemas antes expuestos se define como **objetivo general**: Implementar el módulo de visualización del sistema Syam que permita visualizar espacialmente los datos almacenados en la base de datos del sistema.

Para guiar el desarrollo de la investigación se define la siguiente **idea a defender**: Con la implementación del módulo se podrá visualizar espacialmente los datos contenidos en la base de datos del sistema, logrando así que los especialistas puedan hacer un análisis satisfactorio de los recursos minerales.

Tareas de la investigación:

1. Elaborar el diseño teórico – metodológico de la investigación.
2. Caracterizar los componentes de visualización de datos geológicos espaciales en los principales sistemas informáticos dedicados a los procesos geólogo–mineros.
3. Caracterizar las herramientas y tecnologías asociadas a la solución del problema.
4. Valorar el diseño propuesto para la implementación de la solución.
5. Valorar el diseño y la arquitectura de las tecnologías utilizadas para la solución del problema.
6. Generar los artefactos y la documentación relacionada con el rol de implementador de acuerdo a la metodología de desarrollo a utilizar.
7. Implementar, basado en el diseño de la solución y haciendo uso de la patrones de diseño, la solución de la investigación.
8. Integrar el módulo de visualización con el sistema Syam.
9. Validar los requisitos del módulo.

Como parte de la investigación que se debe llevar a cabo se utilizan para su desarrollo los siguientes métodos científicos:

Métodos teóricos:

- Analítico-Sintético: mediante el cual se extrae la información conceptual más relevante, necesaria para realizar un completo proceso de representación y visualización de objetos geológicos en 3D.

- Histórico-Lógico: para fundamentar la evolución y desarrollo de los sistemas de planificación minera en el mundo y en Cuba.

- Modelación: permite realizar una abstracción del objeto para organizar el flujo de información de la investigación.

En la investigación se esperan como posibles resultados:

- El módulo de visualización del sistema Syam.

- La documentación generada por el rol de implementador para el módulo de visualización del sistema Syam.

El documento tendrá la siguiente estructura:

Capítulo 1: Fundamentación teórica: Este capítulo contiene una explicación de los conceptos necesarios en el proceso de modelado y visualización en tres dimensiones de objetos geológicos. Una caracterización de las tecnologías que serán utilizadas en el desarrollo de la solución que se propone, tales como: *framework* de desarrollo, lenguaje de programación y librerías utilizadas. Un estudio de las principales soluciones mineras que se utilizan en Cuba.

Capítulo 2: Descripción y análisis de la solución propuesta: En este capítulo se abordan las fases de planificación y diseño propias de la metodología de desarrollo utilizada para la implementación del componente de visualización de objetos geológicos en 3D y se exponen los artefactos generados durante el transcurso de las mismas para el rol de implementador.

Capítulo 3: Validación de la solución propuesta: Este capítulo constituye la última fase del proceso de desarrollo del módulo de visualización. Se aborda el diseño de casos de prueba y los resultados obtenidos una vez realizadas las pruebas al módulo desarrollado.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

En este capítulo se realiza un análisis crítico de los paradigmas de programación más utilizados en la actualidad, así como de las herramientas, tecnologías y lenguajes de programación seleccionados para el desarrollo de la solución que dará respuesta al problema. Además se realiza un estudio de las principales soluciones para la actividad minera existentes y en uso en Cuba, enfocado principalmente a las funcionalidades encargadas de la visualización tridimensional de los datos geológicos.

1.1 Conceptos y características de la visualización tridimensional

La visualización es la formación de imágenes visuales. Es el mapeo de datos en representaciones que pueden ser percibidas visualmente (Ribarsky and D. Foley 1994).

La visualización es el proceso a través del cual se construyen representaciones visuales interactivas de datos, con el objetivo de que su manipulación facilite la extracción de información y el entendimiento de los procesos que los generan. El proceso de visualización puede entenderse como una serie de transformaciones que comienzan con un conjunto de datos y finalizan con una representación visual que los codifica (Soto, Sánchez et al. 2007).

Con el paso de los años, la computadora se ha integrado a la visualización y actualmente constituye una herramienta fundamental en ella. Básicamente, la visualización permite interpretar datos que se obtienen de investigaciones matemáticas o científicas, como es el caso de la prospección geológica. En este caso se utilizan los sistemas computacionales para representar los datos recopilados y no para simularlos.

La tecnología de visualización es una integración de las áreas de graficación, procesamiento de imágenes, visión computacional, modelado geométrico, diseño asistido por computadora, entre otros. Un sistema de visualización tiene al menos tres partes fundamentales:

1. Construcción de un modelo empírico de los datos: Este modelo es generado a partir de los datos científicos recopilados.

2. Selección de esquemas: Se toma como modelo un objeto de visualización abstracta (una nube de puntos por ejemplo).

3. La representación de la imagen en un ambiente gráfico.

La visualización es un área de gran importancia en la computación. Al igual que en otras áreas, gracias al avance del software y a la disminución del costo del hardware, se han hecho grandes avances en la visualización.

La visualización exige una capacidad enorme de procesamiento. Por ejemplo: una matriz de 1000 x 1000 x 1000 que contenga datos tridimensionales, tiene 1000 millones de datos de los cuales deberán ser interpretados sus tres componentes X, Y, Z.

Mediante la visualización de los datos no se espera un resultado gráfico cuantitativo, es decir, la representación fiel de valores, sino cualitativo en donde la imagen generada permita tomar decisiones sobre los datos.

Una vez procesados los datos y obtenido un modelo geométrico se procede a renderizarlo.

1.1.1 Conceptos tridimensionales

El campo de la visualización tridimensional es amplio y rico en contenido, por tal razón es recomendado familiarizarse con los conceptos básicos asociados a esta materia.

Rendering

Renderizar es un término usado para referirse al proceso de generar una imagen desde un modelo. Este término es utilizado por los animadores o productores audiovisuales y en programas de diseño en 3D (Eralte 2010).

La palabra renderización proviene del inglés *render*, no existe un verbo con el mismo significado en español, por lo que es frecuente usar las expresiones renderizar o renderear. El término *rendering* también es usado para describir el proceso del cálculo de los efectos en la edición de archivos de videos para producir una salida final (Eralte 2010).

Modelado tridimensional

Es el proceso que se lleva a cabo en la construcción de objetos tridimensionales. El modelado se realiza creando una o varias partes de un objeto geométrico, que juntas darán como resultado la estructura final. Estas partes pueden ser creadas mediante el uso de primitivas tales como: el punto, la línea, el polígono, y superficies curvas.

Existen al menos dos tipos básicos de modelado:

- El modelado conceptual, donde lo importante es la representación tridimensional de una idea o concepto sin entrar en aspectos más técnicos de la estructura, este puede ser muy rápido y útil para las primeras etapas del desarrollo de un producto, componente o estructura y para fines de visualización, mercadotecnia y venta (Iturriaga Morales 2004).
- El modelado en detalle, que incluye todos los detalles con miras al desarrollo de ensambles, animación y planos para la fabricación o construcción (Iturriaga Morales 2004).

En el proceso de modelado tridimensional intervienen además otros factores como el espacio de trabajo o escenario que define los límites del espacio donde se modelará la figura u objeto.

Sistema de coordenadas tridimensionales

Es un sistema de referencia formado por tres rectas o ejes coordenados que se cortan en un punto llamado origen y una unidad de medida, a estas tres rectas también se les llama coordenadas cartesianas respecto al sistema XYZ(Charro Arévalo and Valencia Armijos 2007).

Proyección

Es el proceso de encajar un espacio N-dimensional en un espacio N-1 dimensional. Aplicado a la visualización tridimensional, es el proceso de reducir de tres dimensiones a dos dimensiones una escena en la pantalla.

Existen varios tipos de proyecciones que se pueden aplicar, tales como: proyección en perspectiva, proyección en paralelo, proyección oblicua y proyección ortogonal(Charro Arévalo and Valencia Armijos 2007).

1.1.2 Visualización de objetos geológicos en 3D

En un sistema de visualización, el usuario puede controlar los parámetros de dichas transformaciones y de ese modo explorar distintos aspectos de los datos. Un modelo de referencia simple para describir las etapas del proceso de visualización está compuesto por tres transformaciones, transformaciones de datos, transformaciones visuales y transformaciones de vista.

Las **transformaciones de datos** pueden ser de dos tipos básicos: conversión de formatos particulares a tablas de datos y manipulación de dichas tablas. En primer lugar, los conjuntos de datos, generalmente multivariados y de gran tamaño, se encuentran en formatos específicos del dominio de la aplicación y deben ser organizados con una estructura relacional para lograr mayor flexibilidad. Luego, contando con tablas de datos, se pueden aplicar distintas transformaciones que actúan sobre los valores de las tablas o sobre sus estructuras (eliminación de variables, derivación de nuevas variables, clasificación y ordenamiento) (Soto, Sánchez et al. 2007).

El centro del proceso de visualización son las **transformaciones visuales**, ya que, mediante ellas los datos adquieren una forma gráfica. Las estructuras visuales poseen propiedades gráficas, las cuales son moduladas para codificar información. Ejemplo de estructuras visuales son: volúmenes, polígonos, curvas y puntos. Por otro lado, entre las propiedades gráficas de estas se encuentran: el color, transparencia, textura, forma, tamaño, orientación y posición, siendo algunas más efectivas que otras para la codificación de determinados tipos de datos.

El principal desafío en el desarrollo de aplicaciones es encontrar una forma de representación efectiva y eficiente para representar los datos que se desean visualizar. Cabe destacar que después de realizar las **transformaciones visuales**, sólo se dispone de las estructuras de datos en las cuales se almacena la información geométrica y las propiedades gráficas de cada estructura visual (Soto, Sánchez et al. 2007).

- Generación de bloques: Los bloques, utilizados principalmente para representar volúmenes, se describen por la posición de su centro geométrico y su tamaño (definido en forma global). A partir de estos datos, se construye la representación gráfica de un modelo, para lo cual es necesario generar los polígonos que componen los bloques. Además, se debe determinar el

color asociado a cada bloque, el cual se utiliza para codificar la variable que el usuario desee estudiar.

- Generación de planos cortantes: Se generan planos alineados con los ejes coordenados con los cuales se muestrea el volumen de datos. Los planos permiten reducir la dimensión del problema tridimensional a dos dimensiones, evitando los problemas de oclusión de datos.

- Generación de superficie topográfica: Adicionalmente al modelo de bloques y la secuencia, es posible contar con información topográfica que se puede combinar con las representaciones anteriores. Los datos topográficos están disponibles como puntos de altura ubicados en el plano latitud - longitud. Luego, estos puntos aislados se utilizan para generar una malla de elementos triangulares y finalmente, se calculan las normales promedio en los vértices de los triángulos para generar una representación de buena calidad gráfica.

Las **transformaciones de vista** permiten generar diferentes vistas de las estructuras visuales. Este tipo de transformación involucra rotaciones, traslaciones, acercamientos, recorte geométrico y distorsiones. Estas transformaciones son aplicadas durante la proyección de las estructuras visuales en imágenes, utilizando algoritmos de computación gráfica (Soto, Sánchez et al. 2007).

- Transformaciones geométricas: En computación gráfica las rotaciones, traslaciones y acercamientos se conocen como transformaciones geométricas y son elementos estándar de las aplicaciones. Desde el punto de vista de la interacción, proveen un medio de manipulación mediante el cual se pueden observar modelos desde distintas posiciones, facilitando el entendimiento de la geometría estudiada.

- Filtros geométricos: El denominado filtro geométrico permite limitar la representación del modelo de bloques, de modo que incluya sólo aquellos bloques cuyas posiciones se encuentran dentro de un rango de coordenadas. Esto permite lidiar con la complejidad tridimensional de los modelos, en particular con el problema de oclusión de bloques al representarlos como tales.

1.2 Análisis y caracterización de los principales software mineros usados en Cuba

En la actualidad los especialistas cubanos utilizan principalmente para el análisis y procesamiento de los datos geológicos los software mineros: GEMCOM GEMS, GEMCOM SURPAC y DATAMINE, conjuntamente con estos se usan algunos de producción nacional, tales como: TIERRA, CORTE, SIM y MICRONIQ.

En esta sección se analizarán las herramientas mencionadas anteriormente, identificando en cada caso las principales funcionalidades y propiedades con las que cuentan las mismas con respecto a la visualización de datos geológicos.

1.2.1 Software mineros internacionales

GEMCOM GEMS es el único *software* integrado de principio a fin que ha sido diseñado específicamente para depósitos estratificados. Mediante la integración exhaustiva de todos los aspectos de una explotación, desde la exploración hasta la rehabilitación, GEMS asegura que sus recursos se evalúan con precisión y se explotan con eficiencia, mejorando su productividad y rentabilidad a través de todo el ciclo de vida de la explotación. GEMS fue desarrollado en conjunto con la industria minera y es utilizado por las mayores empresas del área en todo el mundo.

En cuanto al aspecto de la visualización de los datos geológicos, GEMS utiliza todos los datos disponibles para interpretar el depósito, tales como: datos geofísicos, estructurales, calidades, litología y lavado.

GEMS cuenta con herramientas de modelado recurrentes, rápidas y precisas, que construyen modelos completos en minutos, incluyendo la base del manto, espesor, calidad y otras superficies. Las herramientas especializadas para depósitos con fallas permiten modelar con precisión fallas inclinadas o curvas en entornos de falla normal o inversa (Ver Figura 1 y Figura 2) (Gemcom Software International 2012).

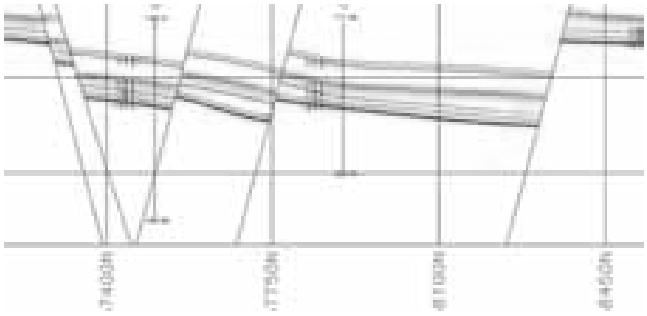


Figura 1 *Corte transversal a través de un modelo de manto con fallas, que incluye las trazas de las perforaciones.*

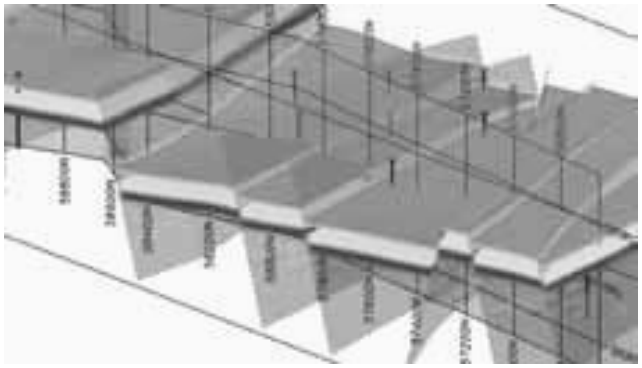


Figura 2 *Corte 3D a través de un modelo de manto con fallas.*

Visualiza curvas de nivel, secciones, perforaciones 3D, estadísticas e histogramas para maximizar la presentación de los recursos a los gerentes, ingenieros de minas o inversionistas (Ver Figura 3) (Gemcom Software International 2012).

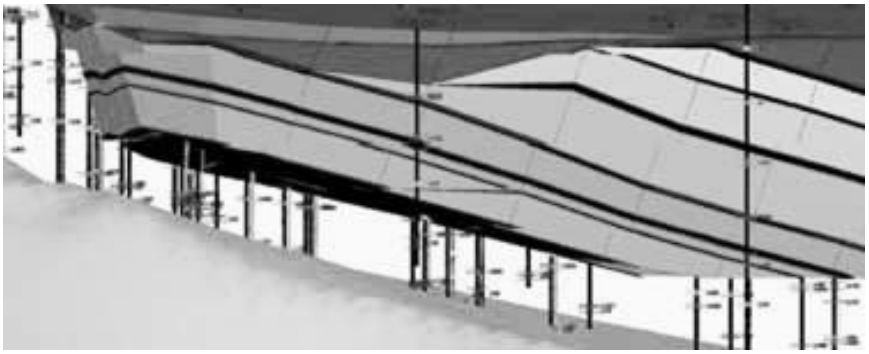


Figura 3 *Vista 3D de los bloques de reservas del modelo de mantos, listos para planificarlos con las perforaciones y datos geofísicos.*

Otro software para la minería, también de Gemcom Software International Inc., es el GEMCOM SURPAC. Según la empresa es el sistema informático de su tipo más usado en el mundo, brindando asistencia a proyectos de exploración y de explotación subterráneos y a cielo abierto en más de 90 países. SURPAC permite a los profesionales cuantificar y evaluar los depósitos de mineral, además de planificar la extracción de reservas de forma eficaz.

SURPAC posee potentes gráficos con lo cual permite a los geólogos determinar las características físicas de un depósito mineral con información limitada por medio geo-estadísticas y un ambiente integrado para la creación de modelos. Sus exhaustivas herramientas para modelos de alambre 3D permiten el desarrollo de un modelo representativo real de cualquier yacimiento (Ver Figura 4).

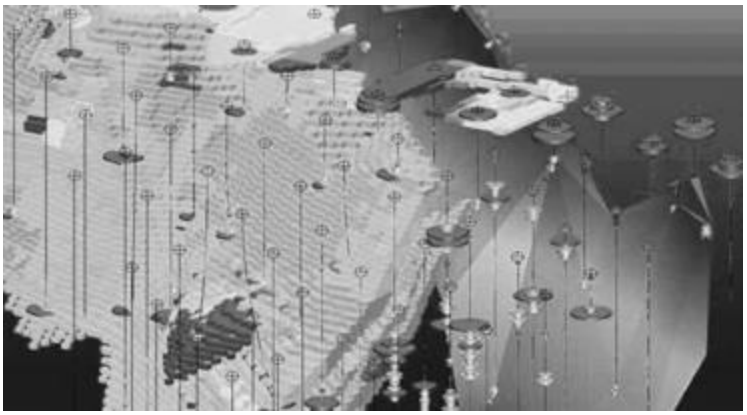


Figura 4 *Modelo de alambre del yacimiento que muestra un modelo de bloque de ley y datos de perforación.*

Las herramientas de modelado de bloques de SURPAC incluyen una amplia gama de funciones y son fáciles de usar. La validación de un modelo y la generación de cualquier nivel de informe se pueden ejecutar de forma rápida y eficiente. Interactúa con todos los datos del diseño de la explotación minera: perforaciones, yacimientos existentes y modelos de superficie; delimitación optimizada del yacimiento, modelos de cuadrícula y de bloque, coloreados de acuerdo con la distribución por ley y muchos más (Ver Figura 5) (Gemcom Software International 2012).

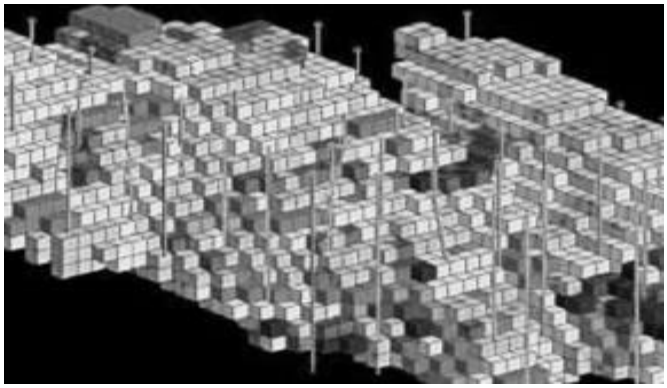


Figura 5 *Modelo de bloque SURPAC limitado por la zona del yacimiento y coloreado de acuerdo con la ley.*

Otro software líder en la minería es DATAMINE, con más de 1000 sistemas en diario uso en más de 45 países a través del mundo. Los usos más comunes del sistema son; la captura y análisis de la información, exploración, geología, geoquímica, mecánica de rocas, topografía, modelado geológico, diseño de mina a cielo abierto y planeamiento minero subterráneo, y áreas relacionadas a los estudios ambientales.

El módulo de visualización de DATAMINE que se nombra “Visualizador 3D” (Ver Figura 6), puede usarse en: Geo-estadística, Modelo de bloques y Evaluación de reservas. Este módulo entrega una vista en 3D, la cual puede ser rotada, trasladada o acercada según las necesidades del usuario.

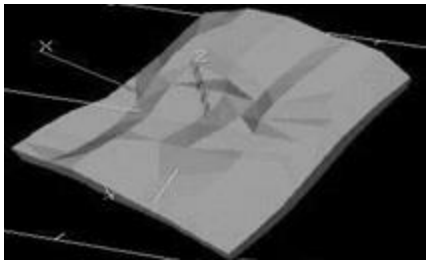


Figura 6 *Captura del visualizador de DATAMINE.*

Posee modelado con *Wireframes*[1], con lo cual se puede hacer la representación visual de un cuerpo mineral a partir de líneas que pueden ser generadas automáticamente y/o editadas para lograr una

[1]**Wireframes**: Algoritmo de renderización del que resulta una imagen semitransparente, de la cual solo se dibujan las aristas de la malla que constituye al objeto. De ahí su nombre.

mayor precisión, posteriormente se triangula, y se obtiene el cuerpo mineral (Gemcom Software International 2012).

1.2.2 Software mineros nacionales

Los yacimientos lateríticos[2] por sus características necesitan de una minería particular para su explotación, por tal motivo en Cuba se desarrolló el software TIERRA que sintetiza y automatiza parte de una metodología para el pronóstico, planificación y control de la minería en yacimientos lateríticos, usado mayormente en la empresa "Comandante Ernesto Guevara de la Serna". Algunas funcionalidades en la sección de geología del software TIERRA son:

- **Cortes Geológicos**

Dados los datos primarios de la red de exploración que se almacenan en archivos *.TXT se realizan los primeros análisis. Se obtienen los resultados siguientes:

1. Tabla Resumen por Tipo de Mena[3].
2. Gráfico de los pozos según perfiles.
3. Gráfico tridimensional de los pozos.
4. Histogramas[4]

- **Cálculos Primarios**

Se presenta un submenú con las siguientes opciones:

1. Tabla de Cálculos Primarios.
2. Tablas para yacimiento a partir de *.PR1.
3. Gráfico de comportamiento geoquímico por pozo.

[2]**Yacimientos lateríticos**: *Latisols*, *latosols* o suelos lateríticos están constituidos principalmente por lateritas, las que son el resultado de un proceso conocido como lateritización. La laterita se forma en la superficie de la corteza terrestre en las zonas tropicales cálidas y húmedas, y está enriquecida con hierro y aluminio.

[3]**Mena**: Mineral de base del que es posible extraer otro mineral de mayor pureza e importancia económica.

[4]**Histograma**: Resumen gráfico de la variación de un conjunto de datos. Es una representación gráfica de una variable en forma de barras, donde la superficie de cada barra es proporcional a la frecuencia de los valores representados.

4. Regularización de la Red de un Bloque: Puesto que las mediciones en el sentido vertical no constituyen una red regular, mediante estimaciones se obtiene una red totalmente regular para cada bloque la cual se usa con diferentes fines en otras opciones.
5. Horizontes: En este caso se presenta la posibilidad de obtener gráficos de isofranjas[5], gráficos tridimensionales, perfiles, áreas y volúmenes en un área de un bloque con respecto a los Techos Topográficos y del Mineral y los Pisos del Mineral y del Pozo.
6. Perfiles con isofranjas de la Red de Exploración: se pueden obtener perfiles en diferentes direcciones y en los mismos se presentan isofranjas de % de Ni, Fe y Co así como del Tipo de Mena.

Otras aplicaciones, tales como: MICRONIQ, CORTE y SIM, también son usados en la industria minera cubana, los mismos cuentan con soporte para la visualización, mas no presentan muchas funcionalidades que faciliten el trabajo de los especialistas.

1.2.3 Conclusiones acerca del estudio realizado

Una vez terminado el análisis y caracterización de las aplicaciones para la minería utilizadas en Cuba, se han identificado funcionalidades que debería presentar el módulo de visualización para el sistema Syam, tales como:

- Creación y visualización se secciones.
- Visualización de perforaciones 3D.
- Permitir rotar, trasladar o acercar la vista 3D, según necesidades del usuario.
- Importar y visualizar datos en varios formatos, siendo este proceso transparente para el usuario.

1.3 Tecnologías y herramientas para la implementación

El desarrollo del módulo de visualización forma parte del proyecto productivo que se lleva a cabo en el proyecto SMC, por lo que se acoge a las decisiones generales tomadas por la dirección del proyecto

[5]**Isofranjas**: Forma de representar datos en mapas. El empleo de las isofranjas es preferido por algunos porque depende sólo del método de estimación utilizado.

acerca de la utilización de tecnologías y metodologías para el desarrollo. Por este factor también estará determinado el uso de patrones de diseño y de arquitectura.

El uso de librerías para la visualización gráfica está guiado a cumplir con el requisito no funcional estudio previo realizado en este trabajo acerca del estado de las tecnologías usadas con este fin.

1.3.1 Metodología de desarrollo

El desarrollo de los sistemas informáticos debe basarse en metodologías para lograr que el *software* cumpla con las características que se requiere que tenga. El objetivo del uso de metodologías de desarrollo es elevar la calidad del *software* a través de una mayor transparencia y control sobre el proceso. Es una labor difícil decidir cuál es la metodología de desarrollo que mejor se adapta a la situación concreta a la que se enfrenta el proyecto en cuestión para obtener los resultados esperados.

El desarrollo del módulo de visualización será guiado por la metodología de desarrollo de *software* Proceso Unificado de *Rational* (RUP por sus siglas en inglés), ya que es la metodología que se aplica al sistema Syam, al cual se integrará el módulo de visualización una vez terminado. Además de que el análisis y diseño del módulo fue realizado utilizando esta metodología.

RUP es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas de software, para diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de aptitud y diferentes tamaños de proyectos(Jacobson, Booch et al. 2000).

RUP es un proceso que se repite a lo largo de una serie de ciclos que forman la vida de un sistema, donde cada ciclo concluye con una versión del producto para los clientes y posee cuatro fases: inicio, elaboración, construcción y transición. Cada fase se divide a su vez en iteraciones. Con cada una de estas fases se persiguen hitos, a los que se le debe dar cumplimiento con la finalización de la misma.

Fases de RUP:

1. Inicio: El objetivo en esta etapa es determinar la visión del proyecto.
2. Elaboración: En esta etapa el objetivo es determinar la arquitectura óptima.
3. Construcción: El objetivo es llegar a obtener la capacidad operacional inicial.

4. Transición: El objetivo es llegar a obtener el *release*[6] del proyecto. Vale mencionar que el ciclo de vida que se desarrolla por cada iteración, es llevado bajo disciplinas.

RUP utiliza el Lenguaje Unificado de Modelado (UML por sus siglas en inglés) para preparar todos los esquemas de un sistema de software. UML es una parte esencial de RUP. Los aspectos definitorios de RUP se resume en tres frases claves: dirigido por casos de uso, centrado en la arquitectura, e iterativo e incremental (Jacobson, Booch et al. 2000).

Dentro de esta metodología existen varios roles en los distintos flujos de trabajo que la conforman, cada uno juega un papel importante en el ciclo de desarrollo. Uno de los roles de gran importancia es el de implementador o desarrollador.

El implementador es el encargado de, metafóricamente, darle vida a las soluciones propuestas para satisfacer las necesidades del cliente y validar el correcto funcionamiento de las mismas. Entre sus tareas se encuentran:

- Analizar el comportamiento en tiempo de ejecución.
- Desarrollar productos de trabajo de instalación.
- Ejecutar pruebas de desarrollador.
- Implementar elementos de diseño.
- Implementar pruebas de desarrollador.
- Implementar los elementos de comprobación.

Estas funciones se resumen en la implementación y las pruebas de los componentes de prueba y los subsistemas correspondientes. Para el correcto cumplimiento de sus responsabilidades debe tener habilidades de programación, conocimiento del sistema o aplicación que se somete a prueba, familiaridad con las herramientas de prueba y de automatización de prueba.

[6] ***Release*:** Versión estable de una aplicación informática lista para desplegar y ser entregada a los clientes.

1.3.2 Herramienta CASE

Se puede definir como Herramientas CASE como un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de *software*, y desarrolladores, durante todo el ciclo de vida de desarrollo de un *software*.

Visual Paradigm 6.4

Visual Paradigm es una herramienta para el diseño y modelado de aplicaciones usando UML, es libre y profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. La herramienta UML CASE también proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML. Soporta UML versión 2.1, permite modelado colaborativo con CVS y Subversión, generación de código, ingeniería inversa, generación de bases de datos (transformación de diagramas entidad-relación en tablas de la base de datos), importación y exportación a ficheros XML, distribución automática de diagramas, entre otras característica (Prada Nicot and Sánchez González 2009).

1.3.3 Lenguajes y bibliotecas de clases

Lenguaje para el modelado

El Lenguaje Unificado de Modelado (Unified Modeling Language, **UML**) es un lenguaje estándar para escribir planos de software. UML puede utilizarse para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de software (Jacobson, Booch et al. 2000).

UML es apropiado para modelar desde sistemas de información en empresas hasta aplicaciones distribuidas basadas en la Web, e incluso para sistemas empotrados de tiempo real muy exigentes. Es un lenguaje muy expresivo, que cubre todas las vistas necesarias para desarrollar y luego desplegar tales sistemas. Aunque sea expresivo, UML no es difícil de aprender ni de utilizar. Aprender a aplicar UML de modo eficaz comienza por crear un modelo conceptual del lenguaje, lo cual requiere aprender

tres elementos principales: los bloques básicos de construcción de UML, las reglas que dictan cómo pueden combinarse esos bloques y algunos mecanismos comunes que se aplican a lo largo del lenguaje (Jacobson, Booch et al. 2000).

UML es sólo un lenguaje y por tanto es tan sólo una parte de un método de desarrollo de *software*. UML es independiente del proceso, aunque para utilizarlo óptimamente se debería usar en un proceso que fuese dirigido por los casos de uso, centrado en la arquitectura, iterativo e incremental (Jacobson, Booch et al. 2000).

Lenguaje de programación

C++ es un lenguaje de programación, diseñado a mediados de los años 1980, por Bjarne Stroustrup, como extensión del lenguaje de programación C. La principal característica de C++ es que es un lenguaje orientado a objetos, y por tanto soporta las capacidades fundamentales de la Programación Orientada a Objetos: Abstracción, Encapsulamiento, Modularidad, y la Jerarquía.

C++ además, brinda soporte para la programación genérica (*templates*), y tiene una enorme compatibilidad con el C principalmente por la gran cantidad de código C que comparten. C++ es un lenguaje de programación estandarizado (ISO/IEC 14882:1998), posee una biblioteca estándar, pero además, es un lenguaje muy potente, flexible y eficaz frente al resto de los lenguajes orientados a objetos, características que han hecho que se le considere como lenguaje universal, de propósito general y ampliamente utilizado tanto en el ámbito profesional como en el educativo. C++ está considerado por muchos como el lenguaje más potente, debido a que permite trabajar tanto a alto como a bajo nivel.

Marco de trabajo

Qt es un amplio marco de trabajo para el desarrollo que incluye clases, librerías y herramientas para la producción de aplicaciones de interfaz gráfica en C++, es multiplataforma por lo que puede ser utilizado en sistemas operativos, tales como: MacOS, GNU/Linux y Windows. Con Qt se pueden desarrollar ricas aplicaciones gráficas, incluye soporte de nuevas tecnologías como OpenGL, XML, Bases de Datos, programación para redes y mucho más. Qt tiene ventajas sobre otras herramientas que proveen un recubrimiento para C++, ya que, al estar implementada directamente en C++, no tiene ninguna dificultad en adaptar los *Widgets*, la calidad en tiempo de ejecución es más alta, ya que no

necesita hacer uso de referencias innecesarias y además, no depende de ninguna otra biblioteca, aparte de sí misma.

La interfaz de programación de Qt está totalmente orientada a objetos; añade a C++ sus propias mejoras tales como: un mecanismo de comunicación entre objetos llamado "señales y ranuras" ("*signals and slots*"), las propiedades de cada objeto se pueden diseñar y consultar, potentes eventos y filtros de eventos, cadenas contextuales de traducción para una mejor internacionalización, un sofisticado soporte de temporizadores que hacen posible integrar elegantemente muchas tareas en una GUI orientada a eventos, un árbol jerárquico de objetos ordenados de un modo natural y con una fácil interpretación.

Biblioteca de clases para la visualización

VTK es una librería de clases de código abierto, dedicado al desarrollo de aplicaciones con gráficos 3D por computadora, procesamiento y visualización de imágenes. VTK consiste en una librería de clases escritas en C++ y varias capas de interfaces para lenguajes como: Tcl/Tk, Java y Python. Kitware, cuyo equipo creó y continúa ampliando esta librería de clases, ofrece apoyo profesional y servicios de consultoría para VTK. VTK es compatible con una amplia variedad de algoritmos de visualización incluyendo: escalar, vectorial, tensor, textura y métodos volumétricos y avanzadas técnicas de modelado tales como: modelado implícito, reducción de polígonos, el suavizado de malla, corte, contorneado y triangulación de *Delaunay*. VTK tiene un marco de visualización de información amplia, cuenta con un conjunto de *widgets* de interacción 3D, soporta el procesamiento paralelo, y se integra con diversas bases de datos en herramientas GUI como Qt y Tk. VTK es multiplataforma y funciona en Linux, Windows, MacOS y Unix (Kitware 2011).

El modelo gráfico de VTK posee un nivel de abstracción mucho mayor que el de otras librerías de renderización de imágenes como OpenGL o PEX. Esto se traduce en una mayor sencillez a la hora de implementar aplicaciones gráficas o de visualización con VTK. Además, las aplicaciones creadas empleando VTK pueden ser escritas directamente en Tcl, Java, Python o C++, lo que aumenta y facilita la posibilidad de implementar aplicaciones en poco tiempo.

Por otra parte, VTK es un sistema de visualización que no sólo permite visualizar geometría, sino que además soporta una amplia variedad de algoritmos de visualización, incluyendo métodos escalares,

vectoriales, tensores, de textura y volumétricos, además de otras modernas técnicas de modelado, como la reducción poligonal, el contorneado, la técnica de *marching cubes*[7], etc.

Visualización con VTK

En VTK, el procesamiento de los datos se realiza a través de un *pipeline*[8]. Una vez los datos entran en el pipeline, pueden ser afectados por multitud de procesos o transformaciones. Para leer los datos se debe emplear el *reader* (lector) apropiado para el tipo de dato. Una vez hecho esto, pueden aplicárseles diferentes tipos de filtros, que alteren los datos de la manera deseada, o definir el tipo de renderización entre otras muchas cosas.

VTK es la herramienta principal empleada en la realización de este proyecto, por lo que profundiza en su empleo y características en capítulos posteriores, aunque como muestra de sus posibilidades, se incluye a continuación un ejemplo de imagen obtenida a partir de estas librerías:

Figura 7 *Imagen procesada con VTK*

1.3.4 Herramientas para la implementación

Qt Creator 2.5

QtCreator es un entorno de desarrollo integrado (IDE por sus siglas en inglés) creado por *Trolltech* para el desarrollo de aplicaciones utilizando el *Framework* Qt. Es multiplataforma, soportando oficialmente los sistemas operativos:

[7]***Marching cubes*:** Es uno de los algoritmos de construcción de superficies para la visualización más avanzados. Descrito por Lorensen y Cline en 1987. Este algoritmo produce una malla de triángulos mediante el cálculo de iso-superficies a partir de datos discretos. Mediante la conexión de los parches de todos los cubos en el límite iso-superficie, se obtiene una representación de la superficie.

[8]***Pipeline*:** Segmentación (en inglés *pipelining*, literalmente tubería o cañería), es un método por el cual se consigue aumentar el rendimiento de algunos sistemas informáticos.

- GNU/Linux 2.6.x, para versiones de 32 y 64 bits con Qt 4.x instalado. Además hay una versión para Linux con gcc 3.3.
- MacOS X 10.4 o superior, requiriendo Qt 4.x
- Windows XP y superiores, requiriendo el compilador MinGW y Qt 4.4.3.

QtCreator posee un editor visual el cual posibilita a los desarrolladores y diseñadores crear aplicaciones más fluidas e intuitivas, combina la edición, depuración, gestión de proyectos, localización y herramientas de compilación. Todas las herramientas de Qt son gratuitas para descargar y usar, además puede ser usado en proyectos comerciales. Con 20 años de vida, QtCreator cuenta con abundante documentación y una gran comunidad online la cual nutre día a día la base de conocimientos mundial con tutoriales y hasta fragmentos de código, permitiendo que se puedan iniciar en el uso de QtCreator más desarrolladores cada día.

PostgreSQL 9.1

Es un servidor de base de datos relacional, libre. Tiene soporte total para transacciones, disparadores, vistas, procedimientos almacenados, almacenamiento de objetos de gran tamaño. Se destaca en ejecutar consultas complejas, consultas sobre vistas, sub-consultas y *joins*[9] de gran tamaño. Permite la definición de tipos de datos personalizados e incluye un modelo de seguridad completo. Como toda herramienta de *software* libre PostgreSQL tiene entre otras ventajas que cuenta con una gran comunidad de desarrollo en Internet, su código fuente está disponible sin costo alguno y algo muy importante es que dicha herramienta es multiplataforma. Además de sus ofertas de soporte, cuenta con una importante comunidad de profesionales y entusiastas de PostgreSQL de los que los centros de desarrollos pueden obtener beneficios y contribuir. Está disponible en casi cualquier Unix (34 plataformas en la última versión estable), y una versión nativa de Windows está actualmente en estado beta de pruebas.

Para complementar el uso de PostgreSQL se usa PostGIS, el cual es una extensión de base de datos espacial para el mencionado gestor de bases de datos. Añade soporte para objetos geográficos permitiendo consultas de ubicación para que se ejecuten en SQL. PostGIS ofrece muchas

[9] ***Joins*:** Se refiere a la sentencia JOIN del lenguaje SQL, utilizado para hacer consultas a bases de datos.

características que rara vez se encuentran en otras bases de datos espaciales de la competencia, tales como Oracle Locator/Spatial y SQL Server.

PostGIS es liberado bajo la Licencia Pública General de GNU (GPLv2). (Team PostGIS 2012)

1.4 Conclusiones parciales

Mediante el análisis realizado en este capítulo de las herramientas, lenguajes, bibliotecas de clases y tecnologías que serán usadas, decidió qué herramientas utilizar en el desarrollo del módulo de visualización.

También se hace un estudio del estado del arte de las aplicaciones dedicadas a la minería que se usan en Cuba, logrando identificar el comportamiento y algunas de las principales funcionalidades que debe ofrecer un *software* minero en cuanto a la visualización de datos geológico-mineros, y que servirán de base para el desarrollo inicial de la aplicación, enfocándolos además en las necesidades propias del sistema.

CAPÍTULO 2: DESCRIPCIÓN E IMPLEMENTACIÓN DE LA SOLUCIÓN PROPUESTA

En este capítulo se hace una valoración del diseño propuesto por el analista, se determinan las relaciones de otros módulos y componentes con sistema desarrollado. Además se describen las principales clases que son necesarias para la implementación de la aplicación, se obtienen los artefactos del rol de implementador que propone RUP, específicamente el diagrama de componentes y el diagrama de despliegue. Se analizan también los estilos y estándares de codificación utilizados en la implementación y que permiten una uniformidad en el código.

2.1 Valoración crítica del diseño propuesto por el analista

Para la elaboración de un *software* capaz de satisfacer las necesidades del usuario el analista del sistema propone el desarrollo de un módulo para integrarse a una aplicación de escritorio, lo que trae como una ventaja importante la capacidad de adaptarse sin dificultad a cualquier plataforma ya sea Windows, Linux. El diseño propuesto proporciona una interfaz agradable, simple y de fácil uso al usuario, además, se ajusta a los estándares establecidos para el desarrollo de un buen diseño.

Para guiar el desarrollo del módulo de visualización el analista del sistema propuso 52 requisitos funcionales encapsulados en 26 casos de uso, 6 requisitos no funcionales de usabilidad y 4 requisitos no funcionales de restricción de diseño.

A continuación de muestran el diagrama de casos de uso y el listado de requisitos funcionales y no funcionales propuesto por el analista del sistema.

2.1.1 Listado de requisitos funcionales y no funcionales

Requisitos funcionales

RF. 1: Mostrar el indicador de los ejes de coordenadas en el visualizador 3D.
RF. 2: Ocultar el indicador de los ejes de coordenadas en el visualizador 3D.
RF. 3: Mostrar el plano de proyección en el visualizador 3D.
RF. 4: Ocultar el plano de proyección en el visualizador 3D.

RF. 5: Crear plano de proyección.

RF. 6: Definir la orientación del plano de proyección.

RF. 7: Definir la dirección del plano de proyección dados 2 puntos.

RF. 8: Orientar la cámara en un plano XY.

RF. 9: Orientar la cámara en un plano XZ.

RF. 10: Orientar la cámara en un plano YZ.

RF. 11: Cambiar el color de fondo del área de visualización 3D.

RF. 12: Visualizar puntos.

RF. 13: Visualizar polilíneas.

RF. 14: Visualizar superficies topográficas mediante redes de triángulos.

RF. 15: Visualizar objetos geológicos mediante redes de triángulos.

RF. 16: Visualizar un objeto geológico en forma de alambrado.

RF. 17: Visualizar un objeto geológico en forma de sólido.

RF. 18: Visualizar objetos geológicos mediante redes de tetraedros.

RF. 19: Visualizar los pozos de perforación.

RF. 20: Mostrar el nombre y otro atributo de los pozos de perforación.

RF. 21: Ocultar los atributos de los pozos de perforación.

RF. 22: Visualizar los intervalos de muestreo de los pozos de perforación.

RF. 23: Colorear los intervalos de muestreos de los pozos de perforación según la leyenda definida.

RF. 24: Aplicar una leyenda a un objeto geométrico.

RF. 25: Mover hacia delante el plano de proyección.

RF. 26: Mover hacia atrás el plano de proyección.

RF. 27: Visualizar el contorno del volumen de visualización.

RF. 28: Ocultar el contorno del volumen de visualización.

RF. 29: Crear conjuntos de planos de perfiles o series.

RF. 30: Eliminar planos de vistas de perfiles.

RF. 31: Rotar la cámara.

RF. 32: Validar si un punto está en un plano dado.

RF. 33: Realizar ajuste al punto más cercano, dada una distancia radial en el espacio.

RF. 34: Realizar ajuste al punto más cercano, dada una distancia radial en el plano de proyección.

RF. 35: Aplicar una transformación lineal a un objeto geométrico.

RF. 36: Seleccionar objetos en un área radial.

RF. 37: Seleccionar objetos en un área rectangular.

RF. 38: Crear leyenda.

RF. 39: Modificar leyenda.

RF. 40: Eliminar leyenda.

RF. 41: Acercar un objeto.

RF. 42: Alejar un objeto.

RF. 43: Realizar paneo sobre un objeto.

RF. 44: Recentrar un objeto.

RF. 45: Visualizar todos los objetos en el área del visualizador.

RF. 46: Definir el área de influencia entre el plano de proyección y los planos paralelos del volumen de visualización.

RF. 47: Definir la distancia del área de influencia del plano de proyección en sentido positivo de la dirección del plano.

RF. 48: Definir la distancia del área de influencia del plano de proyección en sentido negativo de la dirección del plano.

RF. 49: Visualizar las coordenadas de la posición actual del puntero del mouse.

RF. 50: Importar datos topográficos desde un fichero con extensión *.txt.

RF. 51: Importar datos gráficos desde un fichero con extensión *.dxf.

RF. 52: Exportar datos topográficos a fichero con extensión *.txt.

Requisitos no funcionales

Usabilidad

RnF. 1: El sistema debe ser de tipo “Escritorio”.

RnF. 2: El sistema debe cumplir con el objetivo perseguido por el proyecto SMC.

RnF. 3: Se requiere como mínimo un servidor de base de datos de 2 GB de memoria RAM y un procesador INTEL PENTIUM IV o superior.

RnF. 4: Se demanda como mínimo que las PCs clientes posean 512 MB de memoria RAM, un procesador INTEL PENTIUM IV o superior y 256 MB de video.

RnF. 5: Se debe brindar una capacitación a los usuarios para que puedan hacer uso adecuado del sistema.

RnF. 6: Las funcionalidades principales del sistema deben estar orientadas a iconos para un mayor reconocimiento por parte del usuario.

Restricciones de diseño

RnF. 7: El sistema debe ser desarrollado haciendo uso del lenguaje de programación C++.

RnF. 8: La solución se debe desarrollar haciendo uso del sistema gestor de bases de datos PostgreSQL.

RnF. 9: El sistema debe hacer uso de la biblioteca de clases VTK.

RnF. 10: El producto de *software* final debe diseñarse sobre una arquitectura cliente-servidor.

Diagrama de casos de uso del sistema

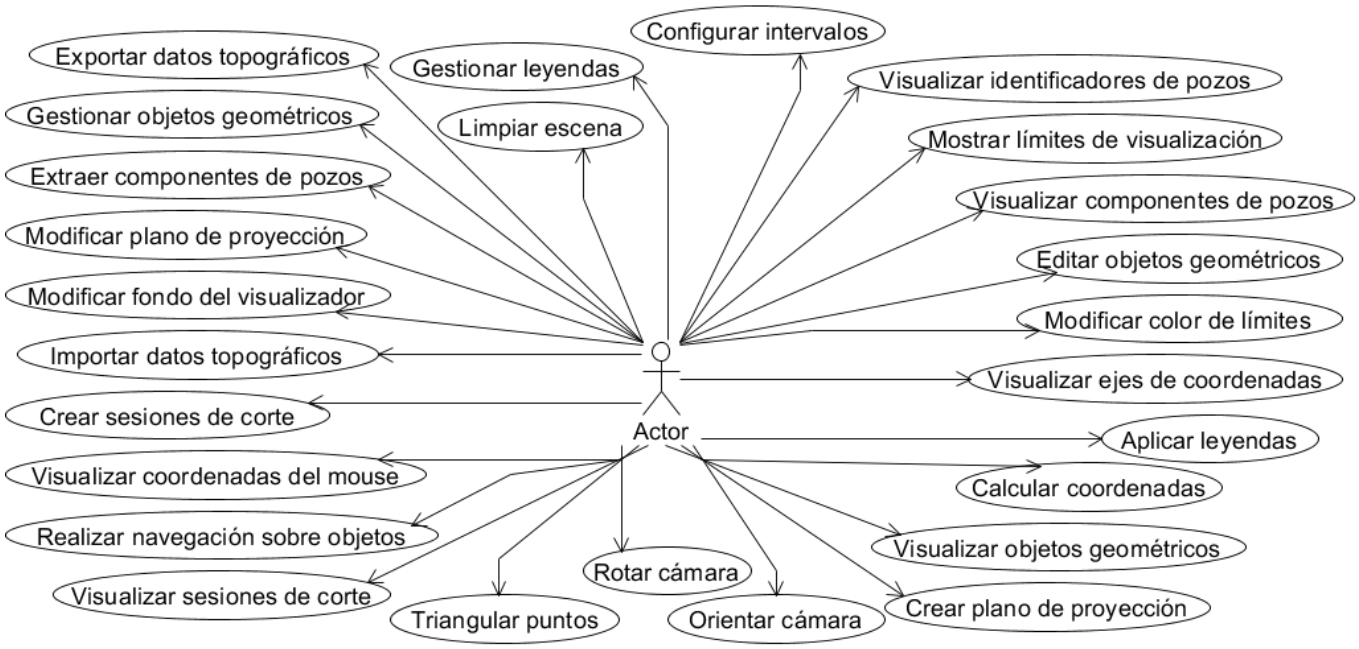


Figura 8 *Diagrama de casos de uso propuesto por el analista*.

2.2 Estándares de codificación

Los estándares de codificación son reglas específicas que se aplican a la hora de escribir el código fuente de las aplicaciones, estos reducen perceptiblemente el riesgo de que los desarrolladores introduzcan errores en el código. Los estándares de codificación no destapan problemas existentes, evitan más bien que los errores ocurran. Con su desarrollo ayudan al programador a producir códigos con alta calidad, al igual que a entender y a utilizar el código por otros programadores (Pressman 2005).

Después de un estudio realizado sobre los estándares de codificación y dado el hecho de que el módulo de visualización formará parte del producto final Syam, se decidió utilizar para el desarrollo del módulo de visualización los estándares de codificación definidos para el sistema Syam. De esta forma se logra uniformidad en el código fuente, y facilita el entendimiento y soporte al código escrito por otros desarrolladores. Los estándares aplicados se encuentran definidos en el documento Estándares de Codificación del sistema Syam.

2.3 Arquitectura del sistema

Con el propósito brindar una visión comprensible de la arquitectura general del módulo de visualización, se describe primero de forma general la arquitectura del sistema Syam el cual contiene al módulo mencionado.

El sistema Syam presenta una arquitectura Cliente-Servidor, en la cual del lado del servidor cuenta con un servidor de bases de datos, y del lado del cliente el sistema Syam, comunicándose entre si mediante el protocolo TCP/IP y usando el driver de Qt para PostgreSQL. La arquitectura interna del sistema Syam es basada en componentes, donde cada componente representa a un módulo en la aplicación, y contiene una interfaz para la interacción con el mismo.

El módulo de visualización se encontrará integrado al sistema Syam, en conjunto con otros módulos, tales como: Base de datos y Núcleo de la aplicación, con los cuales sostiene dependencias para lograr su correcto funcionamiento. Además tiene dependencias con los componentes gmCommon y gmLog que se encuentran en el paquete *libs* que es parte también del sistema Syam. El módulo utiliza la

librería de clases para la visualización gráfica de imágenes VTK. Todos estos elementos se encuentran dispuestos sobre el *framework* de desarrollo Qt.

A continuación se muestra en la Figura 9 una vista de las dependencias del módulo y todos los componentes que colaboran para su funcionamiento.

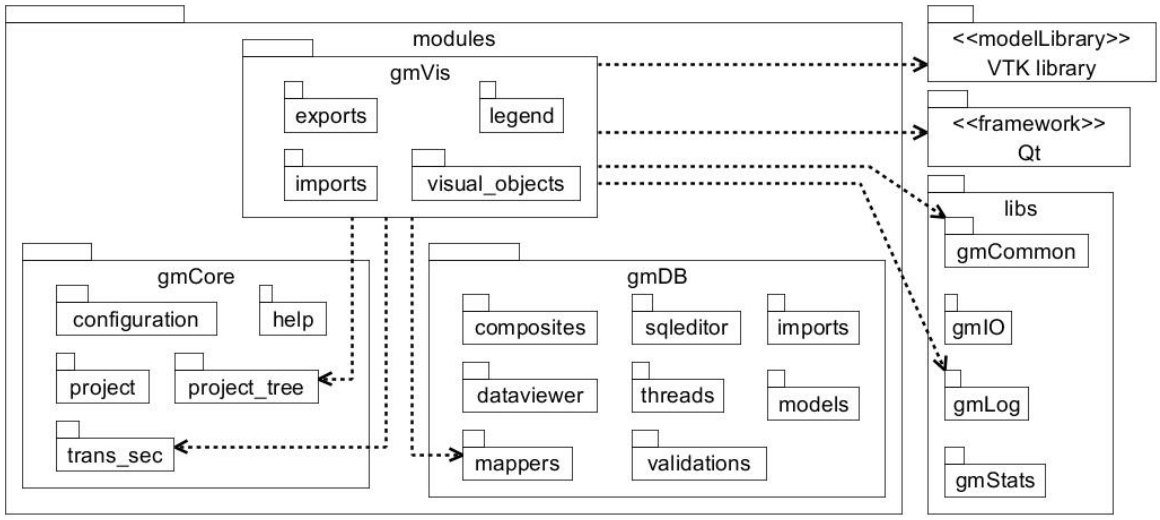


Figura 9 *Vista de la arquitectura del módulo de visualización.*

2.4 Análisis de posibles implementaciones, componentes o módulos ya existentes

Para poder dar solución al problema propuesto en el trabajo, es necesario hacer uso de otros módulos que se encuentran implementados como es el caso del módulo de bases de datos el cual se encarga de la interacción con el servidor de bases de datos y a su vez provee a las restantes partes del sistema acceso a este servicio. Haciendo uso del módulo de bases de datos, desde el módulo de visualización, se recupera la información que será visualizada. Otra parte esencial del sistema es el núcleo, en él se encuentran las clases que se encargan del control integral de la aplicación, gestionando el flujo de información entre módulos.

Se utilizan también la librería de clases definida en la aplicación, la cual provee todas clases obtenidas de la abstracción de los datos, estas clases son usadas por el módulo de bases de datos para la recuperación de los datos, así como para el almacenamiento de los mismos. Se utiliza además la librería de clases externa VTK, la cual es una librería especializada para la visualización y

renderizado de imágenes; en el módulo de visualización se utiliza para la representación en tres dimensiones de los datos geológicos contenidos en la base de datos del sistema.

2.4.1 Estrategia de integración

Según (Fowler 2006): *"La **integración continua** es una práctica de desarrollo de software en la cuál los miembros de un equipo integran su trabajo frecuentemente, como mínimo de forma diaria. Cada integración se verifica mediante una herramienta de construcción automática para detectar los errores de integración tan pronto como sea posible. Muchos equipos creen que este enfoque lleva a una reducción significativa de los problemas de integración y permite a un equipo desarrollar software cohesivo de forma más rápida."*.

Precisamente **integración continua** es la estrategia de integración a utilizar en la implementación del módulo de visualización.

A continuación se muestran que pasos se siguen en la implementación siguiendo las indicaciones de la integración continua: (Fowler 2006)

- Se comienza obteniendo una copia local del código fuente del repositorio de control de versiones, donde se guarda de forma integrada.
- Tras obtener la copia local se procede a hacer los cambios necesarios en el código fuente.
- Una vez finalizados los cambios se llevará a cabo una construcción en la máquina de desarrollo. Esta fase se encargará de compilar el código, empaquetarlo y realizar las pruebas de forma manual. Sólo si todas estas tareas se realizan sin error se considerará que la construcción ha sido correcta. Si hay errores, habrá que solucionarlos.
- Tras una construcción correcta ya se puede pensar en entregar los cambios al repositorio. Puesto que pueden haber otros cambios por parte de otros desarrolladores mientras se trabaja de forma local, lo primero será actualizar la copia local y reconstruir el proyecto. En caso de conflicto con los nuevos cambios aparecerán errores de compilación o durante las pruebas que se deben solucionar antes de poder actualizar el repositorio con nuestros cambios.

- Una vez entregado el código fuente, se volverá a construir la línea principal de desarrollo del proyecto en la máquina de integración. Si no hay errores se podrá decir que los cambios se han llevado a cabo. En caso de que apareciesen errores, habría que solucionarlos. Esta construcción automática en la máquina de integración puede ser realizada de forma manual o de forma automática mediante alguna herramienta.

El resultado de todo este proceso es un proyecto que funciona correctamente y tiene pocos errores. Todo el mundo desarrolla a partir de un código estable y trata de mantenerse lo más cerca de él como para que las integraciones con él no lleven demasiado tiempo. Se tarda menos tiempo en arreglar errores porque estos aparecen más rápidamente. En caso de que aparezcan conflictos entre desarrolladores, estos serán detectados rápidamente. En este momento lo más importante será solucionar estos errores tan pronto como sea posible.

2.5 Despliegue de la solución propuesta

El modelo de despliegue es un modelo de objetos que describe la distribución física del sistema en términos de cómo se distribuye la funcionalidad entre los nodos de cómputo. El modelo de despliegue se utiliza como entrada fundamental en las actividades de diseño e implementación debido a que la distribución del sistema tiene una influencia principal en su diseño (Jacobson, Booch et al. 2000).

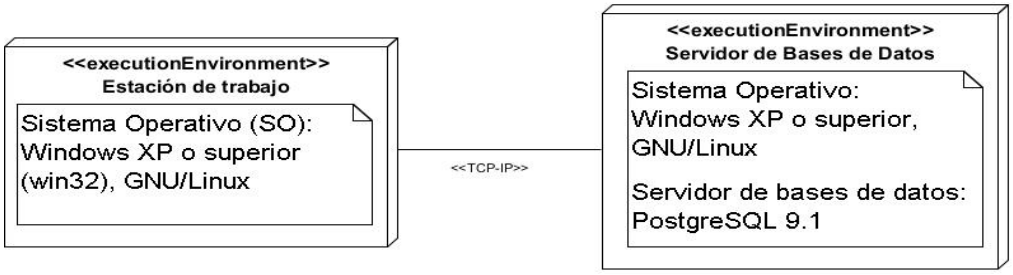


Figura 10 *Diagrama de despliegue*

El sistema Syam, el cual contiene al módulo de visualización implementado, será desplegado en un ambiente como el que se describe en la Figura 10. Será necesaria una estación de trabajo con sistema operativo Windows XP ó superior, o alguna distribución de GNU/Linux, en la cual se ejecutará Syam. Además se necesita de un servidor de bases de datos, con las propiedades que se describen en el diagrama.

2.6 Modelo de implementación

El modelo de implementación describe cómo los elementos del modelo de diseño, tales como las clases, se implementan en términos de componentes, como ficheros de código fuente, ejecutables, etc. El modelo de implementación describe también como se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación y en el lenguaje o lenguajes de programación utilizados, y como dependen los componentes unos de otros (Jacobson, Booch et al. 2000).

Los diagramas de componentes muestran los componentes de *software* que constituyen una parte reusable, sus interfaces, y sus interrelaciones. Un paquete en un diagrama de componentes representa una división física del sistema.

A continuación el diagrama de componentes propuesto:

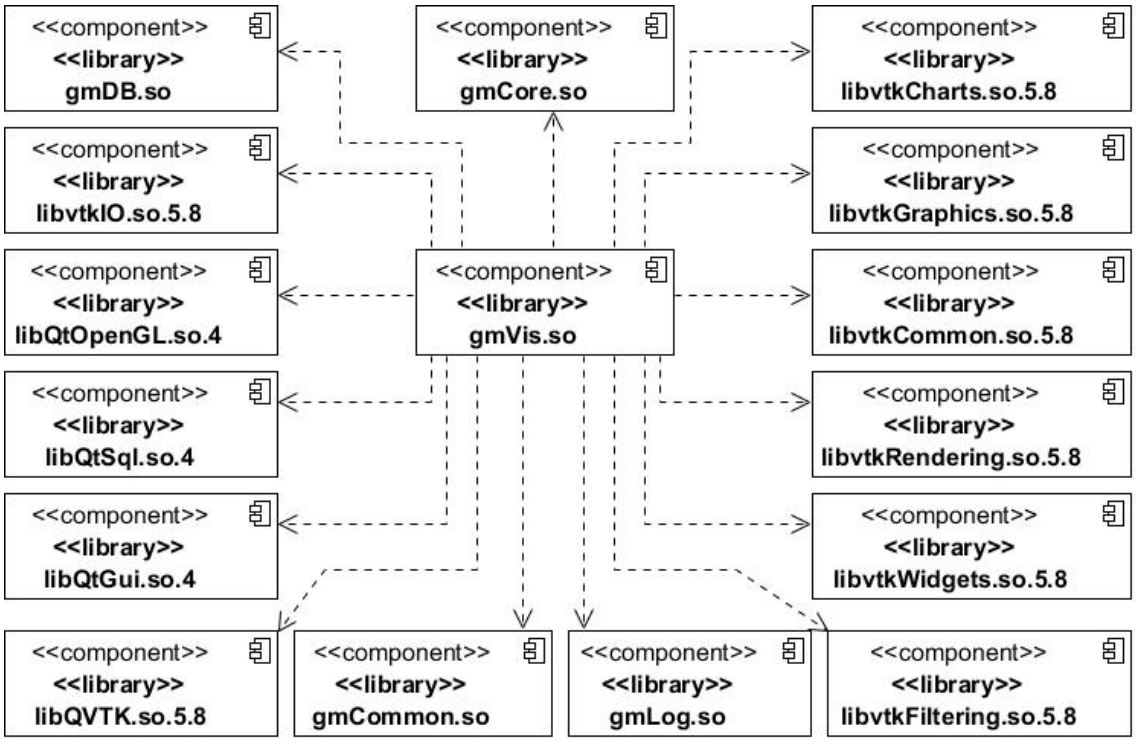


Figura 11 *Diagrama de componentes del módulo de visualización.*

Dado que la aplicación es multiplataforma, para el caso de los sistemas GNU/Linux la extensión de archivo para las librerías es ".so" y para el caso de la familia de sistemas operativos Windows la

extensión de archivo para las librerías es ".dll". El hecho de que la aplicación sea compilada para dos tipos de sistemas operativos no modifica las dependencias de los componentes, a no ser por la extensión de los archivos, en este caso librerías.

Para la implementación del módulo se aplicaron patrones de diseño definidos durante el análisis y diseño del mismo. Los patrones de diseño son principios generales de soluciones que aplican ciertos estilos que ayudan a la creación de *software*. Los patrones de diseño ayudan a los desarrolladores a establecer un lenguaje común entre el diseño de cada uno. Los patrones se dividen de la siguiente manera:

- Patrones GRASP[10]: Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en formas de patrones.
- Patrones GOF[11]: Los patrones GOF están dirigidos al desarrollo de sistemas orientados a objetos.

En la implementación del módulo se aplicaron los siguientes patrones GRASP: experto, control, controlador, alta cohesión y bajo acoplamiento.

A continuación se exponen ejemplos donde se evidencia la aplicación de los patrones GRASP y GOF:

El patrón **experto** guía cómo asignar las responsabilidades a las clases, se puede constatar su uso en la clase gmVisualObjectV de la cual se muestra el código fuente de su archivo cabecera en el Anexo #4 Código fuente del fichero cabecera de la clase gmVisualObjectV.

La clase gmVisualObjectV representa un objeto visual cualquiera. Sobre los objetos visuales se pueden realizar algunas operaciones para modificar su aspecto en la escena de visualización, tales como: cambiar el color, cambiar la visibilidad o modificar su tamaño. La clase gmVisualObjectV es la experta en la información acerca de un objeto visual, por tanto se ubican en ella las responsabilidades antes mencionadas. Con la correcta aplicación del patrón **experto** se garantiza también la aplicación

[10]**GRASP:** Patrones de Asignación de Responsabilidades (*General Responsibility Assignment Software Patterns*).

[11]**GOF:** Banda de cuatro (*Gang of Four*).

del patrón **alta cohesión** en las clases, ya que el mismo permite asignar responsabilidades de manera que la información que almacena una clase sea coherente y esté relacionada con ella misma.

Por otro lado el patrón **creador** determina que clase debe ser la responsable de la creación de un objeto concreto. Para lograrlo se tienen en cuenta que clases contienen la información necesaria para realizar la creación del objeto, cuál usa directamente las instancias creadas del objeto, almacena o maneja varias instancias de la clase, contiene o agrega la clase.

Un ejemplo donde se puede evidenciar la aplicación del patrón **creador** es en la clase frmDialogImportPoints, mediante la cual se importan ficheros que contienen la información necesaria para crear objetos de tipo gmPointsV; por lo que la misma se encarga de crear instancias de estos objetos.

A continuación un ejemplo de código fuente donde se evidencia el uso del patrón antes mencionado:

```
void frmDialogImportPoints::modifiedTable(QRegExp exp, int startLine)
{
    points =  new gmPointsV();
    gmPoint * point;
    vtkSmartPointer<vtkPoints> pointsP = vtkSmartPointer<vtkPoints>::New();
    vtkSmartPointer<vtkCellArray> cellsP = vtkSmartPointer<vtkCellArray>::New();
    vtkSmartPointer<vtkPolyData> polyDataP = vtkSmartPointer<vtkPolyData>::New();
    vtkSmartPointer<vtkPolyDataMapper> mapperP = vtkSmartPointer<vtkPolyDataMapper>::
    vtkSmartPointer<vtkActor> actorP = vtkSmartPointer<vtkActor>::New();
    vtkIdType pid[1];
    QLinkedList<QVariant> row;
    QTextCodec *codec = QTextCodec::codecForName(ui->cbxCodec->currentText().toStdStr
    QTextStream in(&file);
    in.setAutoDetectUnicode(false);
    in.setCodec(codec);
    QString cad = "";
    QStringList tableFields;
    QStringList header;
    band = true:
```

El patrón **controlador** también usado en la implementación del módulo, determina que clase es la encargada de gestionar un evento del sistema. En este caso la clase que controla los eventos del sistema es la clase gmVisModule, la cual es la fachada de todo el módulo, de la cual se muestra el código fuente del fichero cabecera en el Anexo #1 Capturas del módulo de visualización en funcionamiento.

Haciendo uso del patrón **bajo acoplamiento** se garantiza que entre las clases habrá la menor dependencia posible. Lograr un bajo acoplamiento en un sistema está determinado por la correcta aplicación los patrones mencionados anteriormente.

Por otra parte también se aplican los patrones GOF, en este caso instancia única (*singleton*) y el patrón observador. El primero es un patrón creacional que tiene como propósito garantizar una única instancia de una clase, proporcionando un punto de acceso global a la misma (Welicki 2011). Un ejemplo de su utilización en el módulo de visualización se muestra mediante el siguiente fragmento de código fuente de la clase gmVisModule:

```
gmVisModule *gmVisModule::instance()
{
    if(_instance == 0)
        _instance = new gmVisModule;
    return _instance;
}
```

En la solución que se propone se hace uso de este patrón con el objetivo de lograr que la comunicación entre los módulos del sistema siempre se haga a través de las mismas instancias.

Para el trabajo con las interfaces visuales se hace uso del patrón **observador**, mediante el mecanismo que implementa el marco de trabajo Qt, las señales y ranuras (*signals* y *slots*). Este mecanismo permite controlar las acciones realizadas por el usuario desde la interfaz visual.

2.7 Descripción de las nuevas clases y sus principales funcionalidades

Durante la implementación del módulo de visualización se definieron nuevas clases y funciones con el objetivo de lograr el correcto funcionamiento del módulo de visualización. Fueron implementadas 29 clases nuevas y 21 interfaces visuales nuevas con sus respectivas clases controladoras. En los anexos 2, 3 y 4 se muestra el código fuente de los archivos cabecera de las clases gmVisModule, gmVisualizerWidget y gmVisualObjectV respectivamente, estas clases son algunas de las más importantes y con mayores responsabilidades en el módulo.

A continuación se hace una descripción de algunas de las funcionalidades de las clases gmVisModule, gmVisualizerWidget y gmVisualObjectV.

Clase: gmVisModule	
Tipo de clase: Controladora	
Breve descripción: Es la clase controladora del módulo de visualización, por lo que es la que posibilita el flujo de información del módulo con otras partes de la aplicación.	
Descripción de funcionalidades	
staticgmVisModule * instance()	Este método permite la interacción con el módulo de visualización desde fuera de él. Devuelve la misma instancia del módulo en cada llamada que se hace al mismo.
voidinitMenus()	Crea el menú “Visualización”, agregándole las opciones y funcionalidades del módulo. Añade a la barra de menús principal de la aplicación el menú del creado.
voidinitToolbars()	Crea la barra de herramientas del módulo, agregándole los botones y funcionalidades principales. Añade a la aplicación la barra de herramientas del módulo de visualización.
voidinsertingData(gmProjectItem*)	Permite guardar los datos que se han generado y/o modificado hacia la base de datos de la aplicación.
voidcalculateCoords()	Hace referencia a un procedimiento almacenado en la base de datos, el mismo calcula las coordenadas de profundidad de cada pozo, para poder hacer la visualización del mismo.
voidcreateProfiles()	Crea cortes o secciones de la vista actual, que pueden ser guardas en la base de datos, para su posterior uso.

voidvisCollars ()	Visualiza en el visor las cabezas de los pozos almacenados en la base de datos del proyecto.
voidvisIntervals()	Representa en los pozos visualizados los intervalos definidos atendiendo a la concentración de minerales en las muestras.
voidsetOrientationProjectionPlane()	Cambia la orientación del plano de proyección mostrado en el visor.
voidextractCollars()	Permite exportar las cabezas de los pozos hacia un fichero de texto.

Tabla 1 *Descripción de la clase gmVisModule.*

Clase: gmVisualizerWidget	
Tipo de clase: Controladora	
Breve descripción: Esta clase representa al visor del módulo de visualización. Por tanto es la que se encarga del control de todas las funcionalidades que necesiten visualizar datos de forma tridimensional.	
Descripción de funcionalidades	
voidVisualiceBoxModel()	Visualiza en el visor un cubo dentro del cual se enmarcaran todos los objetos que sean visualizados.
void VisualicePlaneView(double *center, double *normal)	Visualiza enmarcado dentro del cubo en el visor, el plano de proyección que se usará en varias operaciones sobre los datos que se visualicen.
vtkRenderer* getRenderer()	Devuelve el renderizador que usa actualmente el visor.
voidsetClipping(gmProfile *profile)	Permite la visualización de una sección en el visor, justo después de haberla creado y sin guardar y/o recuperada desde la base de datos.

voidremoveClipp()	Restaura la vista en el visor, removiendo la sección representada.
double* getExtremeBoundsModel()	Devuelve las ocho coordenadas que representan los límites del cubo.

Tabla 2 *Descripción de la clase gmVisualizerWidget.*

Clase: gmVisualObjectV	
Tipo de clase: Entidad	
Breve descripción: Esta clase representa un objeto visual cualquiera. La clase gmVisualObjectV es la raíz de una jerarquía de clases, de ella heredan clases que representan a objetos visuales especializados, tales como: gmPointsV, gmPoligonsV, entre otras.	
Descripción de funcionalidades	
vtkActor* getVisualObjectActor();	Crea un actor y lo devuelve, que será usado para representar el conjunto puntos en la escena del visor.
vtkActor2D* getVisualLabels();	Crea un actor en 2D que contiene etiquetas, las cuales serán visualizadas en la escena.
void changeColorProperty(const QColor &);	Cambia el color del objeto visual por uno especificado por parámetros.
virtual void changeSize(const int &) = 0;	Cambia el tamaño del objeto visual representado. Este método es implementado por las clases que hereden de gmVisualObjetcV
void setVisible(bool pVisible);	Cambia la visibilidad del objeto visual, ocultándolo o mostrándolo en la escena.
void setLegend(gmLegend* legend);	Aplica una leyenda al objeto visual, la misma determinará la apariencia del objeto visual atendiendo a las propiedades del objeto geológico que representa el mismo.

Tabla 3 *Descripción de la clase gmPointsV.*

2.8 Conclusiones parciales

En este capítulo se identificaron elementos importantes que describen la solución propuesta y guían la implementación del módulo de visualización tridimensional.

Con la descripción detallada de la arquitectura del módulo y el ambiente donde será integrado se podrá implementar el módulo eficientemente. Además mediante la especificación del estándar de codificación a utilizar en la implementación del módulo, se permite una mayor comprensión del código fuente, ya que provee de una herramienta que permite uniformidad a la hora de escribir el programa.

El modelo de implementación permitió tener una idea más clara de la composición interna de los subsistemas, pues mostró el funcionamiento interno de los módulos a través de los Diagramas de Componentes, conociéndose además los elementos que lo componen.

CAPÍTULO 3: VALIDACIÓN DE LA SOLUCIÓN PROPUESTA

Para alcanzar la calidad requerida, un producto de *software* tiene que transitar por un proceso exhaustivo de pruebas. Al *software* se le pueden aplicar diversos tipos de pruebas, cada cual con un objetivo específico y estrategia bien definidas, con las cuales se podrán identificar posibles errores que puedan existir en el *software*. En este capítulo se abordarán las características de las pruebas de caja blanca y caja negra, particularizando principalmente en las pruebas que se le realizarán al módulo de visualización.

3.1 Descripción general de las pruebas

En el flujo de trabajo de la prueba se verifica el resultado de la implementación probando cada construcción, incluyendo tanto construcciones internas como intermedias, así como las versionas finales del sistema a ser entregadas a terceros (Jacobson, Booch et al. 2000).

La prueba es un conjunto de actividades que se planean con anticipación y se realizan de manera sistemática. Por tanto, se debe definir una plantilla para las pruebas del *software* (Pressman 2005).

El objetivo de la etapa de la prueba de componentes es descubrir defectos probando componentes de programas individuales. Estos componentes pueden ser funciones, objetos o componentes reutilizables. Durante las pruebas del sistema, estos componentes se integran para formar subsistemas o el sistema completo. En esta etapa, la prueba del sistema debería centrarse en establecer que el sistema satisface sus requisitos funcionales y no funcionales, y no se comporta de forma inesperada. Inevitablemente, los defectos en los componentes que no se han detectado durante las primeras etapas de las pruebas se descubren durante las pruebas del sistema (Sommerville 2005).

Las pruebas del *software* implican ejecutar una implementación de un sistema terminado con datos de prueba. Se examinan las salidas del mismo y su entorno operacional para comprobar que funciona tal y como se requiere. Las pruebas son una técnica dinámica de verificación y validación (Sommerville 2005).

La prueba no puede asegurar la ausencia de errores; sólo puede demostrar que existen defectos en el *software*.

Existen dos tipos fundamentales de pruebas unitarias que se pueden aplicar a un *software* después de terminado, las pruebas de caja negra y las pruebas de caja blanca. A continuación se describen estos tipos de pruebas.

Pruebas de caja blanca

Al desarrollar un nuevo *software* o sistema de información, la primera etapa de pruebas a considerar es la etapa de pruebas unitarias o también llamada pruebas de caja blanca, estas pruebas también son llamadas pruebas modulares ya que permiten determinar si un módulo del programa está listo y correctamente terminado, estas pruebas no se deben confundir con las pruebas informales que realiza el programador mientras esta desarrollando el módulo (Oré B. 2009).

Pruebas de caja negra

Se refiere a las pruebas que se llevan a cabo sobre la interfaz del *software*, por lo que los casos de prueba pretenden demostrar que las funciones del *software* son operativas, que la entrada se acepta de forma adecuada y que se produce una salida correcta, así como que la integridad de la información externa se mantiene. Esta prueba examina algunos aspectos del modelo fundamentalmente del sistema sin tener mucho en cuenta la estructura interna del *software*.

La prueba de Caja Negra no es una alternativa a las técnicas de prueba de la Caja Blanca, sino un enfoque complementario que intenta descubrir diferentes tipos de errores a los encontrados en los métodos de la Caja Blanca.

Muchos autores consideran que estas pruebas permiten encontrar:

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a las Bases de Datos externas.
- Errores de rendimiento.
- Errores de inicialización y terminación.

Para desarrollar la prueba de caja negra existen varias técnicas, entre ellas están: (Pressman 2005)

1. Técnica de la ***Partición de Equivalencia***: esta técnica divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software.
2. Técnica del ***Análisis de Valores Límites***: esta Técnica prueba la habilidad del programa para manejar datos que se encuentran en los límites aceptables.
3. Técnica de ***Grafos de Causa-Efecto***: es una técnica que permite al encargado de la prueba validar complejos conjuntos de acciones y condiciones.

3.2 Diseño de casos de prueba que permitan validar la solución propuesta

Siguiendo lo que indica la metodología RUP, el rol de implementador desarrolla los componentes de software y efectúa las pruebas de desarrollador, generando el artefacto “Prueba de desarrollador”. Este artefacto abarca el trabajo tradicionalmente pensado bajo las categorías siguientes: Pruebas de unidad, parte de las Pruebas de integración, y algunos aspectos de lo que se denomina Pruebas del sistema. El objetivo de la prueba de desarrollador es proporcionar la implementación de un subconjunto de pruebas necesarias de forma efectiva y eficaz.

Para las pruebas al módulo de visualización se diseñaron e implementaron pruebas de caja negra aplicando la técnica de partición equivalente, diseñándose quince casos de prueba.

A continuación se muestran dos casos de prueba para los casos de uso Obtener Vista y Visualizar componente de los pozos respectivamente.

Caso de prueba para el caso de uso Obtener Vista

Descripción general

El actor desea obtener una de las 5 vistas que son posibles visualizar de la pantalla de visualización.

Condiciones de ejecución

Se ha activado la pestaña del visualizador.

SC Obtener Vista			
Escenario	**Descripción**	**Respuesta del sistema**	**Flujo central**

EC 1.1 Vista superior	En este escenario el usuario selecciona obtener la “Vista Superior” del área de visualización	Vista Superior donde se muestra las características del objeto visualizado.	Paso 1: Muestra una vista ortogonal superior del área de visualización, enfocando la cámara en dirección perpendicular al plano XY.
EC 1.2 Vista frontal	En este escenario el usuario selecciona obtener la “Vista frontal” del área de visualización	Vista frontal donde se muestra las características del objeto visualizado.	Paso1: Muestra una vista ortogonal frontal del área de visualización, enfocando la cámara en dirección perpendicular al plano XZ.
EC 1.3 Vista lateral izquierda	En este escenario el usuario selecciona obtener la “Vista lateral izquierda” del área de visualización	Vista lateral izquierda donde se muestra las características del objeto visualizado.	Paso 1: Muestra una vista ortogonal lateral izquierda del área de visualización, enfocando la cámara en dirección perpendicular al plano YZ.
EC 1.4 Vista lateral derecha	En este escenario el usuario selecciona obtener la “Vista lateral derecha” del área de visualización	Vista lateral derecha donde se muestra las características del objeto visualizado.	Paso 1: Muestra una vista ortogonal lateral derecha del área de visualización.
EC 1.5 Vista trasera	En este escenario el usuario selecciona obtener la “Vista trasera” del área de visualización	Vista trasera donde se muestra las características del objeto visualizado.	Paso 1: Muestra una vista ortogonal trasera del área de visualización, enfocando la cámara en dirección perpendicular al plano XZ.

Tabla 4 *Caso de prueba Obtener Vista.*

Para el caso de prueba descrito no existen variables, ya que no hay entrada de datos por parte del usuario.

Caso de prueba para el caso de uso Visualizar componente de los pozos

Descripción general

El actor decide mediante una de las opciones visualizar los collares o intervalos de los pozos.

Condiciones de ejecución

Se ha activado la pestaña del visualizador.

Existe al menos un pozo en la Base de Datos del proyecto en que se trabaja.

SC Visualizar componente de los pozos			
Escenario	**Descripción**	**Respuesta del sistema**	**Flujo central**
EC 1.1 Visualizar Collares	En este escenario el usuario puede visualizar los collares almacenados en el sistema, dando clic encima de los *Items* del árbol de proyecto.	Visualiza los collares seleccionado por el usuario.	Paso 1: Visualiza, según las propiedades definidas para los puntos, los collares de los pozos existentes en el Visualizador. Paso 2: Continúa en el paso 3 del flujo básico "Visualizar componente de los pozos".
EC 1.2 Visualizar Intervalos	En este escenario el usuario puede visualizar los intervalos de muestras y litología de los collares seleccionados.	Visualiza los intervalos de muestras y litología de los pozos seleccionados por el usuario.	Paso 1: Visualiza, de acuerdo a los intervalos configurados, los intervalos de muestras y litologías, de cada uno de los pozos existentes en el Visualizador Paso 2: Continúa en el paso 3 del flujo básico "Visualizar componente de los pozos".

Tabla 5 *Caso de prueba Visualizar componente de los pozos.*

Para el caso de prueba descrito no existen variables, ya que no hay entrada de datos por parte del usuario.

3.3 Resultados de las pruebas

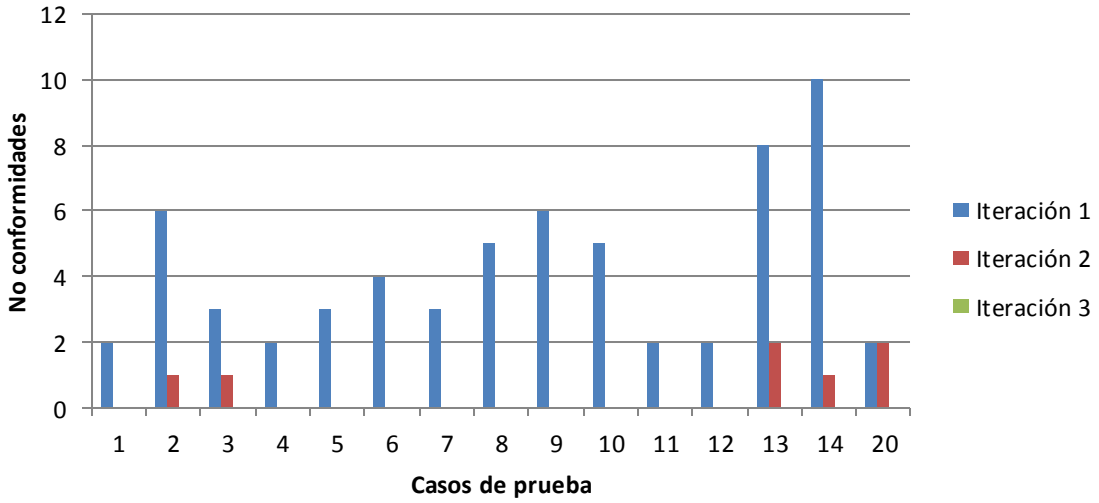
El módulo de visualización se probó en tres iteraciones, las cuales se describen a continuación:

Iteración 1. Se probaron los 15 casos de prueba, obteniéndose 36 no conformidades, entre inconsistencias en el sistema, errores y casos de prueba mal redactados. Modificados los casos de prueba, y arreglando los errores encontrados, se pasa a la segunda iteración.

Iteración 2. Se volvió a probar los 15 casos de prueba, obteniéndose 8 no conformidades. Después de satisfacer las no conformidades, se pasa a la tercera iteración.

Iteración 3. Se probaron 5 casos de prueba, no encontrándose errores en esta iteración. Concluyendo así el proceso de prueba.

A continuación se muestran los resultados de las pruebas reflejados en un gráfico de barras:



Luego de realizadas las pruebas se concluye que el sistema cumple con los requisitos y funcionalidades requeridas. En el anexo #1 se muestran algunas capturas del módulo en funcionamiento.

3.4 Conclusiones parciales

El uso de las pruebas de caja negra, específicamente las de partición equivalente permitieron diseñar casos de pruebas robustos, que permitieron identificar errores en la implementación del módulo de visualización.

Con la implementación de los casos de pruebas diseñados en este capítulo se asegura que el módulo de visualización para el sistema Syam cumple con los requisitos propuestos en el análisis y diseño propuesto al implementador.

CONCLUSIONES GENERALES

Teniendo en cuenta los objetivos trazados en la investigación, los requisitos funcionales planteados y en aras de darle cumplimiento al problema propuesto, se ha arribado a las siguientes conclusiones:

- Mediante el análisis de la arquitectura y el diseño del módulo, se logró tener una idea más clara de la composición interna del módulo así como de la aplicación donde se encuentra integrado, analizando además, las tecnologías utilizadas para su implementación.

- Se logró implementar el módulo de visualización del sistema Syam que permite visualizar tridimensionalmente los objetos geológicos con los cuales trabaja el producto, logrando así que los especialistas puedan hacer un análisis satisfactorio de los recursos minerales.

- Las pruebas realizadas al módulo permitieron demostrar que está listo para su liberación y despliegue en un ambiente de producción. Se comprueba además que cumple con los requisitos especificados por el analista del sistema, por tanto el módulo provee las funcionalidades que permitirán a especialistas mineros visualizar espacialmente los datos contenidos en la base de datos del sistema.

- Con la obtención de los artefactos asociados al rol de implementador que están definidos por la metodología utilizada (RUP), se facilita el trabajo de futuros implementadores, al tener toda la documentación detallada y disponible.

GLOSARIO DE TÉRMINOS

Caso de Uso (CU): Es una descripción de la secuencia de interacciones que se producen entre un actor y el sistema, cuando el actor usa el sistema para llevar a cabo una tarea específica. Expresa una unidad coherente de funcionalidad, y se representa en el Diagrama de Casos de Uso.

Diagrama de Casos de Uso (DCU): Muestra la relación entre los actores y los casos de uso del sistema. Representa la funcionalidad que ofrece el sistema en lo que se refiere a su interacción externa.

Framework: Esquema, esqueleto o patrón para el desarrollo y/o la implementación de una aplicación.

Mineral: Sustancia natural que se diferencia del resto por su origen inorgánico, su homogeneidad, composición química preestablecida y que corrientemente ostenta una estructura de cristal.

Multi-plataforma: Sistema informático que corre sobre varios sistemas operativos, sin prescindir de ninguna de sus funcionalidades.

Prospección: Es la búsqueda de yacimientos, que se hace en base a mapas de distinto tipo, fotografías aéreas, imágenes satelitales, antecedentes mineros, geológicos, geofísicos, geoquímicos, catastrales, económicos. La ejecución de las tareas de prospección (trabajos de campo y de laboratorios) está en manos de geólogos especialistas, que cuentan con la ayuda de la tecnología apropiada para cada caso, vehículos, equipos, instrumental, laboratorios.

Requisito: Petición de una acción que se considera necesaria.

Sistema empotrado: Sistema informático que se encuentra físicamente incluido en un sistema de ingeniería más amplio al que supervisa o controla. Los sistemas empotrados se encuentran en multitud de aplicaciones, desde la electrónica de consumo hasta el control de complejos procesos industriales.

Software: Es el equipamiento lógico o soporte lógico de un sistema informático, que comprende el conjunto de los componentes lógicos necesarios que hacen posible la realización de tareas específicas, en contraposición a los componentes físicos que son llamados hardware.

Yacimiento Mineral: Es la acumulación natural de minerales en la corteza terrestre, en forma de uno o varios cuerpos minerales, los cuales en este estado, pueden ser objeto de extracción y explotación industriales, en la actualidad o en un futuro inmediato.

RECOMENDACIONES

Se recomienda que en futuras versiones se le agregue al módulo soporte multi-hilos para el aprovechamiento de microprocesadores de varios núcleos.

Agregar al módulo funcionalidades para el modelado 2D de yacimientos minerales.

REFERENCIAS BIBLIOGRÁFICAS

Charro Arévalo, C. and V. W. Valencia Armijos (2007). Modelo tridimensional de la historia geológica del volcán Cotopaxi. Quito.

Eralte, A. (2010). "Definicion de Render. ¿Que es Renderizacion? ." from http://www.arquigrafico.com/definicion-de-render-que-es-renderizacion.

Fowler, M. (2006). "Continuous Integration." from http://martinfowler.com/articles/continuousIntegration.html.

Gemcom Software International, I. (2012). from www.gemcomsoftware.com.

Iturriaga Morales, L. J. (2004). "Modelado 3D. Visualización Fotorrealista." 2013, from http://webspace.webring.com/people/ui/iturriagal/modelado.html.

Jacobson, I., G. Booch, et al. (2000). El lenguaje unificado de modelado.

Jacobson, I., G. Booch, et al. (2000). The unified software development process. Madrid, PEARSON EDUCATION, S.A.

Kitware. (2011). "VTK." from http://www.vtk.org/.

Oré B., A. (2009). "PRUEBAS UNITARIAS." Retrieved 09-04-2013, 2013, from http://www.calidadysoftware.com/testing/pruebas_unitarias1.php.

Prada Nicot, H. and K. Sánchez González (2009). Desarrollo de los componentes Puesto de Trabajo y Pagos Adicionales del subsistema Capital Humano integrado al sistema integral de gestión CEDRUX. La Habana.

Pressman, R. S. (2005). "Software Engineering: A Practitioner's Approach."

Ribarsky, W. and J. D. Foley (1994). Next-generation Data Visualization Tools.

Sommerville, I. (2005). Software Engineering, PEARSON EDUCACIÓN. S.A.

Soto, L., R. Sánchez, et al. (2007). Desarrollo de un Sistema de Visualización para la Planificación Minera. Chile, Universidades de Talca, Concepción y Chile.

Team PostGIS. (2012). "Spatial and Geographic objects for PostgreSQL."

Welicki, L. (2011). "El Patrón Singleton ", from http://msdn.microsoft.com/es-es/library/bb972272.aspx#EAAA.

BIBLIOGRAFÍA CONSULTADA

Ribarsky, W. and J. D. Foley (1994). Next-generation Data Visualization Tools.

Belete Fuentes, O. (2000). Perfeccionamiento del cálculo de volumen en los yacimientos lateríticos cubanos. Minería y Geología. XVII.

Jacobson, I., G. Booch, et al. (2000). The unified software development process. Madrid, PEARSON EDUCATION, S.A.

Quintero, A. R. (2000). "Visión General de la Programación."

Fuentes, L., J. M. Troya, et al. (2001). "Desarrollo de Software Basado en Componentes."

Letelier Torres, P. (2002). Desarrollo de Software Orientado a Objeto usando UML. Valencia.

Iturriaga Morales, L. J. (2004). "Modelado 3D. Visualización Fotorrealista." 2013, from http://webspace.webring.com/people/ui/iturriagal/modelado.html.

Pressman, R. S. (2005). "Software Engineering: A Practitioner's Approach."

Sawyer, S. (2005). "Resampling Data: Using a Statistical Jackknife."

Sommerville, I. (2005). Software Engineering, PEARSON EDUCACIÓN. S.A.

Juristo, N., A. M. Moreno, et al. (2006). "Técnicas de evaluación de software."

Charro Arévalo, C. and V. W. Valencia Armijos (2007). Modelo tridimensional de la historia geológica del volcán Cotopaxi. Quito.

Gómez, O. S. (2007). "Paradigma de programación dirigido por eventos."

Gutiérrez, J. (2007). "Introducción al Proceso de Pruebas.".

Soto, L., R. Sánchez, et al. (2007). Desarrollo de un Sistema de Visualización para la Planificación Minera. Chile, Universidades de Talca, Concepción y Chile.

Oré B., A. (2009). "PRUEBAS UNITARIAS." Retrieved 09-04-2013, 2013, from http://www.calidadysoftware.com/testing/pruebas_unitarias1.php.

Palacio, L. G. (2009). "Método para generar casos de prueba funcional en el desarrollo de software."

Prada Nicot, H. and K. Sánchez González (2009). Desarrollo de los componentes Puesto de Trabajo y Pagos Adicionales del subsistema Capital Humano integrado al sistema integral de gestión CEDRUX. La Habana.

Aristegui O., J. L. (2010). "Los casos de prueba en la prueba de software." Lámpsakos 3: 27-34.

Dr. Fernández-Medina Patón, E. (2010). "Pruebas del Software." from http://alarcos.inf-cr.uclm.es/doc/ISOFTWAREI/.

Eralte, A. (2010). "Definición de Render. ¿Que es Renderización?" from http://www.arquigrafico.com/definicion-de-render-que-es-renderizacion.

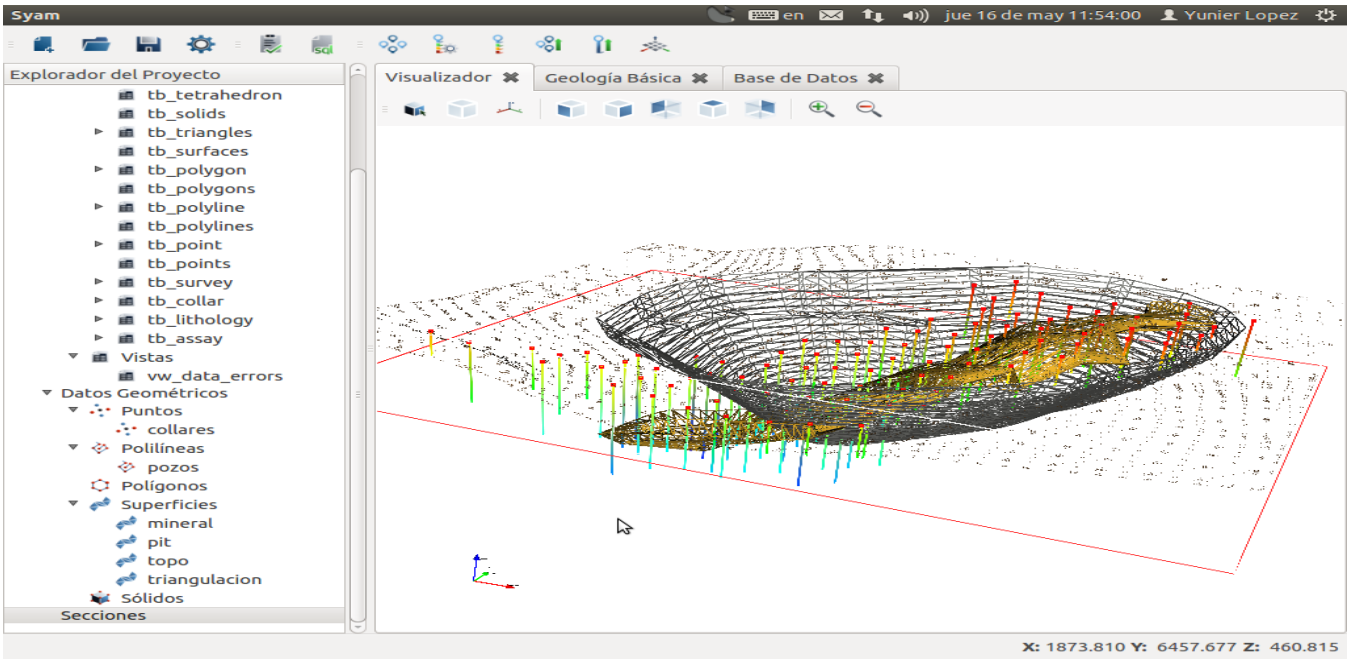
Welicki, L. (2011). "El Patrón Singleton ", from http://msdn.microsoft.com/es-es/library/bb972272.aspx#EAAA.

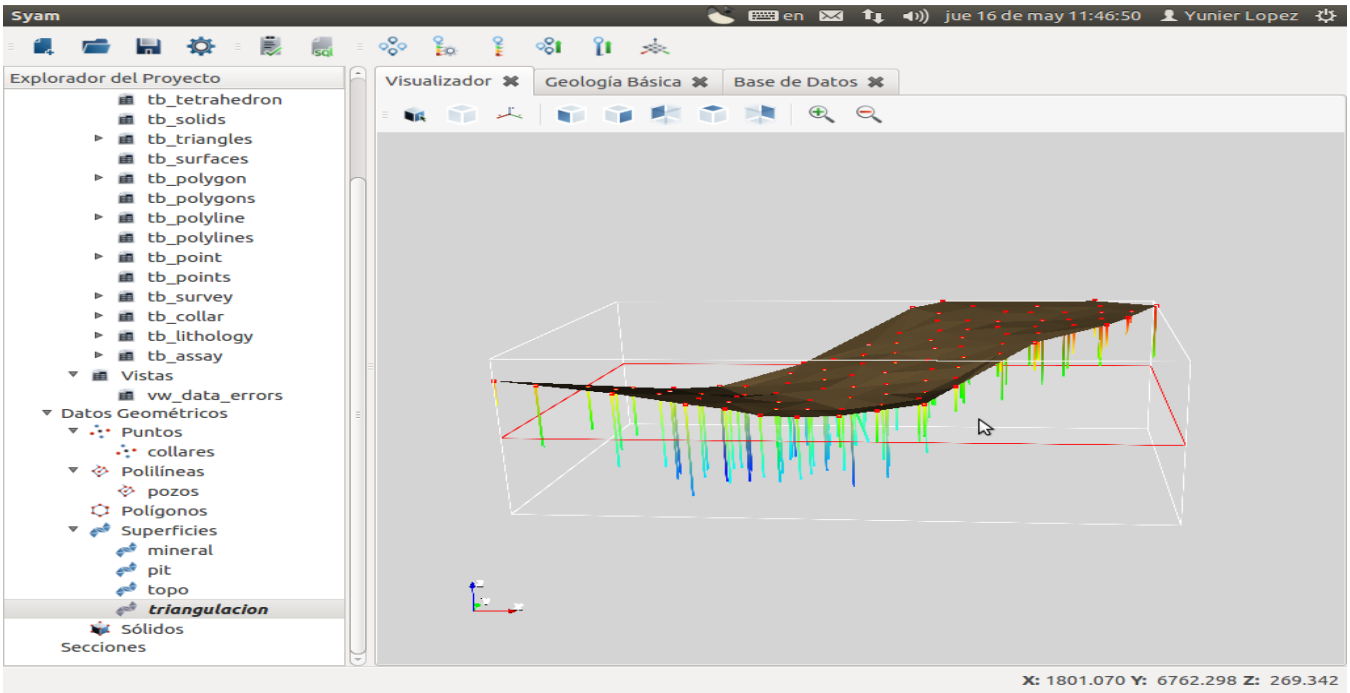
Kitware. (2011). "VTK." from http://www.vtk.org/

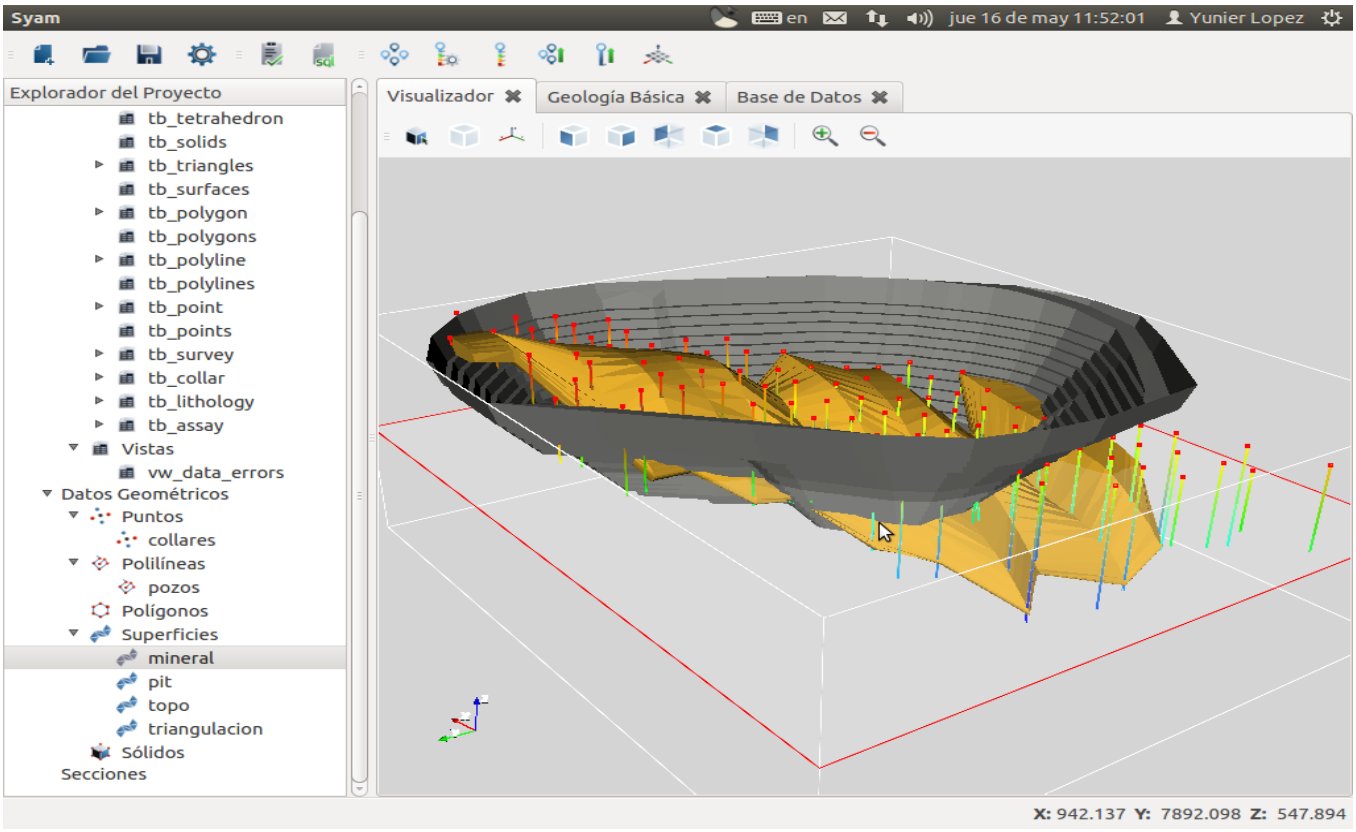
Gemcom Software International, I. (2012). from www.gemcomsoftware.com

ANEXOS

4.1 Anexo #1 Capturas del módulo de visualización en funcionamiento.







4.2 Anexo #2 Código fuente del fichero cabecera de la clase gmVisModule.

```
#ifndef GMVISMODULE_H
#define GMVISMODULE_H
#include "gmvis_global.h"
#include "../../libs/gmCommon/gmmodule.h"
#include "../../libs/gmCommon/gmprofile.h"
#include "visualObjects/gmtrianglesv.h"
#include "visualObjects/gmpointsv.h"
#include "visualObjects/gmpolylinesv.h"
#include "visualObjects/gmpolygonsv.h"
#include "visualObjects/gmlinesv.h"
#include <QProgressBar>
class gmProjectItem;
```

```
//class gmDefaultTreeItem;
class gmVisualObjectV;
class gmFrmConfigureIntervals;
class GMVIS_EXPORT gmVisModule : public gmModule
{
    Q_OBJECT
public:
    ~gmVisModule();
    int id() const;
    QString name() const;
    QString longName() const;
    gmEnums::Library library() const;
    QString version() const;
    QStringList dependencies() const;
    QList<QWidget*> centralWidgets();
    QList<QMenu*> centralMenus();
    QWidget* configuration();
    QWidget* coordinates();
    gmEnums::Module module();
    void showVisualObject(QString, gmVisualObjectV*);
    void reset();
    void createComponents();
    void initComponents();
    void setCoordsActions(bool);
    void insertingData(gmProjectItem*);
    static gmVisModule * instance(); //Patrón Singleton
    QAction * actionCalculateCoord();
    void removeVisualFromRenderer(gmVisualObjectV*);
    void addActionToVisMenu(QAction*);
    void restoreVisual();
```

```
    gmProfile *actualSectionView();
    void updateActualSectionView();
protected:
    static gmVisModule * _instance;
    gmVisModule(QObject *parent = 0);
private:
    void initForms();
    void initActions();
    void initMenus();
    void initToolbars();
    QMap<QString, gmVisualObjectV*> _visualObjects;
    gmFrmConfigureIntervals* config;
    gmPolylinesV* intervals;
    gmPointsV* collars;
public slots:
    void showDialogImportTri_File();
    void showDialogImportPoints_File();
    void showDialogImportDXF_File();
    void showDialogImportGrid_File();
    void showDialogImportPolylines_File();
    void showDialogExportPoints();
    void showDialogExportTri();

    void extractCollars();
    void extractSurveys();
    void calculateCoords();
    void calculateCoords(QProgressBar * pbar);
    void changeView(int);
    void visualChange();
    void refreshVisual();
```

```
        void triangulatePoints(gmProjectItem*);
        void zoomIn();
        void zoomOut();
        void setClipping();
        void manageLegend();
        void configureIntervals();
        void visIntervals();
        void visCollars();
        void visLabels();
        void setOrientationProjectionPlane();
        void setOrientationProjectionPlaneByVector(double * vector);
        void setOrientationProjectionPlaneByAngles(double * angles);
        void createProfiles();
        void addProfile(gmProfile* profile);
        void visProfile(gmProjectItem*prof);
signals:
        void importVisualFile(gmProjectItem*);
        void extractedData();
        void cleanedScene();
        void findLeafBranch(QString);
        void branchLeaf(QStringList leafName);
        void newProfileAdded(gmProjectItem*);
};
#endif // GMVISMODULE_H
```

4.3 Anexo #3 Código fuente del fichero cabecera de la clase gmVisualizerWidget.

```
#ifndef GMVISUALIZERWIDGET_H
#define GMVISUALIZERWIDGET_H
#include <QVTKWidget.h>
#include <QColorDialog>
```

```
#include <QMessageBox>
#include <QEvent>
#include <QMouseEvent>
#include <gmdesignwidget.h>
#include <QAction>
#include "gmvis_global.h"
class vtkRenderer;
class vtkActor;
class vtkAxesActor;
class vtkAxisActor;
class vtkOrientationMarkerWidget;
class vtkCubeAxesActor;
class vtkCamera;
class DesignInteractor;
class gmCollarMapper;
class vtkPointSetToLabelHierarchy;
class vtkBoundingBox;
class vtkRenderWindow;
class gmDesignerWidget;
class vtkProp;
class vtkActor2D;
class gmVisualInteractor;
class vtkScalarBarActor;
class vtkLookupTable;
class vtkDataObject;
class gmProfile;
class gmVisualObjectV;
class GMVIS_EXPORT gmVisualizerWidget : public QVTKWidget
{
    Q_OBJECT
```

```
public:
    explicit gmVisualizerWidget(QWidget *parent = 0);
    ~gmVisualizerWidget();
    void VisualiceBoxModel();
    void VisualicePlaneView(double *center, double *normal);
    double* getModelCenter();
    double* getExtremeBoundsModel();
    double* getScale();
    vtkRenderer* getRenderer();
    void mouseMoveEvent(QMouseEvent *e);
    void wheelEvent(QWheelEvent *e);
    void Update(gmDesignerWidget *desWidget);
    void replacePlaneWidget(vtkActor* actor);
    void reset();
    void AddEvent(QString event);
    void InitializeProjectionPlane();
    void RemoveEvent(QString event);
    gmVisualInteractor* getInteractorStyle();
    gmDesignWidget* getDesignWidget();
    void setBarLookupTable(vtkLookupTable* table);
    void setDataObject(vtkDataObject* data);
    void setClipping(gmProfile *profile);
    void removeClipp();
    gmProfile *getActualSectionView();
private:
    vtkRenderer *renderer;
    vtkActor *planeactor;
    vtkActor2D* _scalarActor;
    //vtkActor *boxactor;
    vtkAxesActor *axesactor;
```

```
    vtkOrientationMarkerWidget  *widget;
    gmDesignWidget* designPlaneWidget;
    vtkCamera *camera;
    gmVisualInteractor  *interactorStyle;
    QMap<QString, vtkProp*> * _actorsOnScene;
    struct Camera{
        int actualView;
        double * initialFocalPoint;
        double * initialViewUp;
        double * initialPosition;
    } cameraState;
    vtkBoundingBox*  _boundings;
    double * normalAnterior;
    double * origenAnterior;
    gmProfile * actualSeccion;
    bool visBBox;
signals:
    void changeCoordinates(double x, double y, double z);
    void visualChange();
    void cleanedScene();
    void customView();
public slots:
    void visualizeIndicadorEjes();
    void visualizeScalarBar();
    void changeBackgroundColor();
    void changeBBoxColor();
    void visualizeBBox();
    void visualizePlane();
    void cleanScene();
    void superiorView();
```

```
        void lateralLeftView();
        void lateralRightView();
        void frontalView();
        void backView();
        void updateSectionView();
        void changeToPerspectiveProj();
        void changeToParallelProj();
};
#endif // GMVISUALIZERWIDGET_H
```

4.4 Anexo #4 Código fuente del fichero cabecera de la clase gmVisualObjectV

```
#ifndef GMVISUALOBJECTV_H
#define GMVISUALOBJECTV_H
#include <QString>
#include "../gmvis_global.h"
#include <QColor>
#include "../../../libs/gmCommon/gmlegend.h"
class vtkActor;
class vtkActor2D;
class vtkLookupTable;
class GMVIS_EXPORT gmVisualObjectV
{
protected:
        vtkActor* actorVisObject;
        vtkActor2D* actorLabels;
        virtual void makeActor() = 0;
        gmLegend* _legend;
        vtkLookupTable*  look;
        bool visible;
public:
```

```
    gmVisualObjectV();
    virtual ~gmVisualObjectV();
    vtkActor* getVisualObjectActor();
    vtkActor2D* getVisualLabels();
    void visibleObject(bool);
    void changeColorProperty(const QColor &);
    virtual void changeSize(const int &)   = 0;
    virtual void changePattern(const int &)   = 0;
    QColor getColorProperty();
    gmLegend* getLegend();
    void setLegend(gmLegend* legend);
    void reloadActor();
    vtkLookupTable*  getLookupTable();
    bool getVisible();
    void setVisible(bool pVisible);
};
#endif
```