UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

DESARROLLO DE UNA LÍNEA DE PRODUCTOS DE SOFTWARE DE GENERACIÓN DE MALLAS GEOMÉTRICAS

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERA CIVIL EN COMPUTACIÓN

VIOLETA NAOMI DÍAZ RÍOS

PROFESORA GUÍA:
NANCY HITSCHFELD KAHLER

MIEMBROS DE LA COMISIÓN:
MARÍA CECILIA BASTARRICA
ROMAIN ROBBES

SANTIAGO DE CHILE
OCTUBRE 2012

Resumen

Las líneas de productos de software constituyen un paradigma de desarrollo, en el cual se busca construir, a partir de un conjunto de elementos clave, sistemas que comparten características comunes, o similitudes, y a la vez mantienen características propias, o variabilidades. Un ejemplo de sistema que puede verse beneficiado con este paradigma es el software generador de mallas geométricas. Las mallas geométricas son poderosas herramientas que permiten representar un objeto como un conjunto de polígonos contiguos.

El presente trabajo de título tuvo por objetivo la construcción de una interfaz gráfica que permita al usuario la configuración y creación automática de productos correspondientes a generadores de mallas con distintas funcionalidades, utilizando la ingeniería de líneas de productos de software. El desarrollo consistió en extender la aplicación desarrollada en un trabajo de título anterior, que permitía configurar un producto, pero dado el acoplamiento del software base utilizado, todos los productos contenían el mismo código fuente.

El software base utilizado se compone de dos generadores de mallas ya existentes: Simulador de crecimiento de árboles, sistema que modela el crecimiento de un árbol mediante mallas geométricas, y Generador genérico de mallas, que implementa una secuencia de pasos básica y común para todos los generadores de mallas y provee un apoyo conceptual. Ambos productos permiten cargar, almacenar y manipular las mallas.

Durante el desarrollo de esta memoria, primero se realizó una revisión de las funcionalidades de ambos productos, para después identificar sus similitudes y variabilidades. Luego, estas características fueron agrupadas en tres grupos: Tipo de Aplicación, Manejo de Archivos y Algoritmos de procesamiento de mallas. Cada uno de estos grupos correspondió a una sección de la interfaz desarrollada.

La interfaz de usuario fue construida de manera que permita la configuración de los productos en tiempo de compilación y de ejecución, mediante la generación de dos archivos de configuración. Esto implicó un análisis sobre el código fuente del software base, en el cual se detectó un alto acoplamiento entre varias de sus clases; por lo tanto, fue necesario realizar una intervención sobre el código de estas clases, para poder desacoplarlas y así compilarlas por separado, excluyendo las clases innecesarias.

Como resultado, la interfaz desarrollada permite configurar y crear automáticamente productos generadores de mallas geométricas, de manera que incluyan sólo las funcionalidades escogidas por el usuario. Para trabajos futuros, se propone una re-ingeniería mayor del software base Simulador de crecimiento de árboles, que lo adapte para su utilización en el contexto de una linea de productos de software; además, es posible la adición de nuevos algoritmos y funcionalidades a la línea de productos ya construida.

Agradecimientos

En primer lugar, gracias a mis padres, Jorge y Verónica, por su amor y apoyo incondicionales durante mis años de estudio; de no ser por ellos, este trabajo no existiría. Agradezco también a Horacio, mi hermano, por su alegre compañía y comprensión en todo momento.

Gracias a mi profesora guía, Nancy Hitschfeld, por su tiempo, dedicación, disposición y paciencia. Fue un completo agrado ser su alumna.

Gracias a la Comisión Nacional de Investigación Científica y Tecnológica (CONICYT) por el financiamiento de este trabajo a través del proyecto Fondecyt N° 1120495: "Improving the functionality and performance of meshing tools".

Gracias a mis amigos y compañeros: Héctor, Felipe M., Cristóbal, Ariel, Sebastián, Felipe C., Mario, Cristián, Alejandra, Luis Felipe, Soledad, Felipe L., Luis, Pía, Liliana, Francisca, Úrsula y Mabel. Agradezco de corazón su infinita paciencia y su apoyo constante, que me mantuvieron en pie hasta llegar al final.

Finalmente, gracias a Hernán. Gracias por tu amor, gracias por permanecer a mi lado día a día, gracias por impulsarme siempre a ser más, sin dejar de ser yo misma. Gracias por compartir conmigo cada momento de esta recta final, y por levantarme cada vez que lo necesité. Las palabras se quedan cortas para describir lo que significas para mí. Te amo profundamente.

Índice general

Índice de figuras

Capítulo 1
Introducción

1.1. Antecedentes generales

En la actualidad, el desarrollo de sistemas de software es cada vez más complejo, debido a la rápida evolución de la tecnología y a la creciente importancia que estos sistemas van cobrando en nuestras vidas. La ingeniería de software ha ofrecido durante décadas distintas técnicas para mejorar la eficiencia en el proceso de desarrollo y la calidad de los productos finales, minimizando el costo de tiempo y recursos materiales. Una de estas técnicas es la reutilización [1], la cual consiste en desarrollar productos de software lo suficientemente flexibles como para que puedan ser utilizados más de una vez en desarrollos futuros, necesitando una mínima cantidad de modificaciones. En base a esta estrategia, nació un nuevo paradigma en el desarrollo de software, llamado "Líneas de Productos de Software" (en adelante LPS).

El Instituto de Ingeniería de Software de la Universidad Carnegie Mellon (SEI) define de la siguiente manera una LPS [2]: "Es un conjunto de sistemas de software que comparten un conjunto común y gestionado de características que satisfacen las necesidades específicas de un segmento de mercado particular o misión, y que son desarrolladas de forma prescrita a partir de un conjunto común de elementos clave". Como lo indica la definición, una LPS se basa en una plataforma, o conjunto de elementos clave, para producir sistemas de software que comparten características comunes (llamadas similitudes o *commonalities*), y que a la vez mantienen características propias (llamadas variabilidades o *variabilities*).

Un ejemplo de sistema que podría verse beneficiado con este paradigma de desarrollo es el software generador de mallas geométricas. Las mallas geométricas son herramientas poderosas que permiten representar de manera discreta dominios u objetos de geometría compleja [5]; en particular, las mallas de superficie representan un objeto como un conjunto de polígonos contiguos. Actualmente, son cada vez más utilizadas en áreas tales como gráfica de juegos, la industria del cine y simulaciones científicas. Un ejemplo de malla de superficie se muestra en la Figura 1.1.

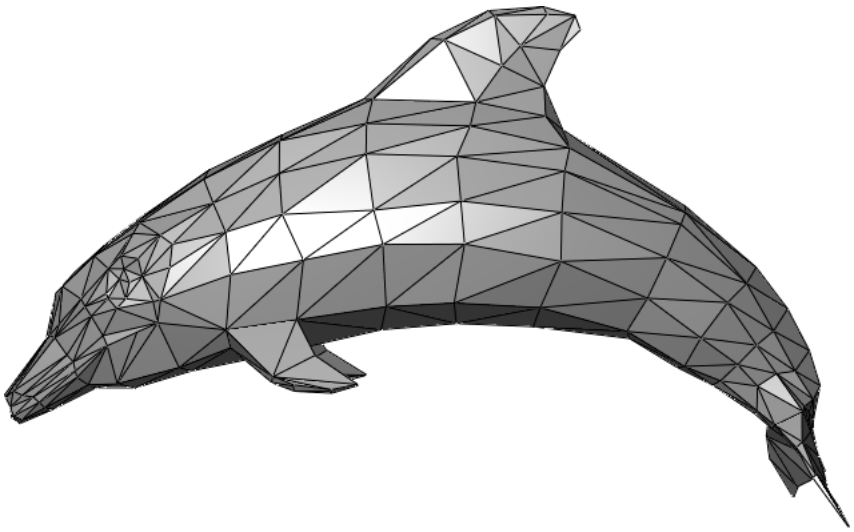

Figura 1.1: Malla de triángulos que representa un delfín.

Como los generadores de mallas geométricas utilizan funcionalidades similares de manipulación de mallas, resulta interesante construir una línea de productos de software de generación de mallas geométricas, la cual simplificaría el desarrollo de nuevos productos, junto con la extensión de los ya existentes.

1.2. Motivación

El desarrollo de un software generador de mallas geométricas involucra la toma de decisiones con respecto a las características comunes a todos los productos de esta especie, por ejemplo qué tipo de malla se usará (triángulos o cuadriláteros), qué formatos de archivo de entrada o salida se incluirán, qué algoritmos se aplicarán sobre la malla (refinamiento, desrefinamiento, mejoramiento o deformación), etc. Esto genera una gran variedad de alternativas de productos, lo cual hace indispensable que sistemas como éstos implementen estas características de manera flexible, permitiendo su configuración de manera rápida y sencilla por parte de los usuarios. Para lograr esto, existe la posibilidad de desarrollar un sólo sistema que contenga todas las características posibles, pero esto se hace infactible debido a que el número de ellas va creciendo año a año. Además, el producto nunca alcanzaría la etapa de producción, ya que los tiempos de desarrollo y mantenimiento para agregar los nuevos aspectos aumentarían considerablemente.

Una solución promisoria para este problema es desarrollar aplicaciones basadas en una LPS, las cuales permiten la reutilización de componentes de herramientas ya construidas. Esto conlleva los siguientes beneficios [2]:

- Disminución en los costos del software.
- Disminución en el esfuerzo requerido para la producción del software.
- Incremento en la calidad.
- Disminución en el tiempo de publicación del software al mercado.

- Aumento en la flexibilidad del software para adoptar otros usos y así entrar en nuevos mercados.

El presente tema de memoria, teniendo esta solución en mente, pretende extender el trabajo realizado en una memoria de título anterior [10]. En dicha memoria se desarrolló un sistema llamado *Meshing Tool Generator*, el cual consiste en una interfaz gráfica que permite configurar un software generador de mallas geométricas a través de la toma de decisiones por parte del usuario, quien escoge las funcionalidades a las cuales quiere acceder al utilizar el producto final. Si bien este sistema constituye un primer acercamiento al desarrollo de una LPS, genera productos con implementaciones iguales que sólo cambian en la organización del menú; por lo tanto, aún existe la necesidad de generar una configuración a nivel de la compilación de clases, es decir, crear productos cuyas clases sean las estrictamente necesarias según las funcionalidades que el usuario escoja.

La base para el desarrollo de la nueva LPS se compone de las siguientes aplicaciones ya existentes:

- **Simulador de crecimiento de árboles**: Esta aplicación, también llamada *Tree Growth Simulator* (en adelante TGS), fue desarrollada en tres trabajos de título anteriores [4, 5, 9]. Apoyando el trabajo de un grupo de profesionales de las áreas de Matemática y Ciencias de la Computación, se encarga de modelar computacionalmente el crecimiento de un árbol, en particular de la superficie cilíndrica de su tronco y las deformaciones de sus ramas, a través de mallas geométricas de triángulos o cuadriláteros. Esta aplicación permite la creación, visualización, modificación, carga y almacenado de dichas mallas, funcionalidades que buscan representar la mayor cantidad de propiedades observadas en la naturaleza al crecer el tronco y las ramas de un árbol. En la Figura 1.2 se muestra la interfaz de la aplicación, modelando el tronco de un árbol a través de una malla de triángulos.

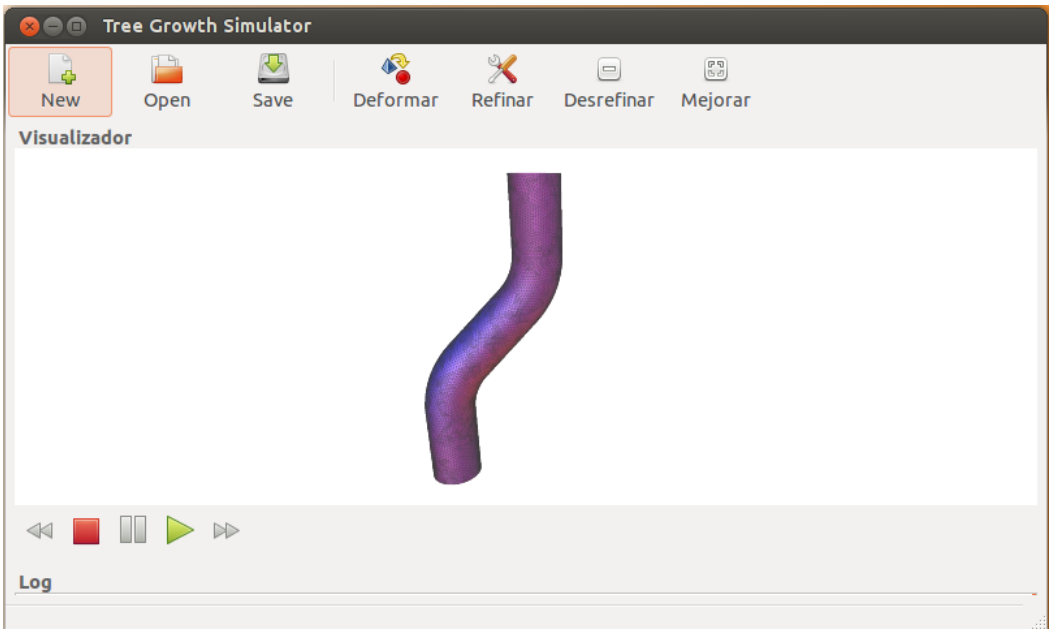

Figura 1.2: Interfaz del software Simulador de crecimiento de árboles.

- **Generador genérico de mallas** [1]: Sistema que implementa una familia de productos generadores de mallas, el cual identifica una secuencia de pasos básica y común a todos ellos: generar una malla inicial que se ajuste a la geometría del dominio, generar una malla intermedia, refinar o mejorar esta malla para satisfacer el criterio de calidad requerido y generar una malla final. Este sistema constituye un apoyo conceptual, tanto para la aplicación anterior como para la nueva LPS.

1.3. Objetivo general

El objetivo general de esta memoria es la construcción de una interfaz gráfica que permita al usuario la configuración y creación automática de productos que corresponden a generadores de mallas con distintas funcionalidades, utilizando para esto la ingeniería de líneas de productos de software.

1.4. Objetivos específicos

- Revisión y mejora de la actual implementación del software TGS, disminuyendo el acoplamiento entre sus clases y eliminando dependencias innecesarias para así facilitar su configuración.
- Inclusión de los conceptos del Generador genérico de mallas geométricas en la línea de productos de software a generar.
- Determinar los puntos de similitud y variabilidad entre los productos generadores de mallas, y en base a ellos generar un árbol de decisión para que el usuario escoja las funcionalidades que necesite.
- Extensión de la aplicación *Meshing Tool Generator* para que, en tiempo de ejecución, despliegue las alternativas consistentes con las decisiones que va tomando el usuario.
- Extensión de la aplicación *Meshing Tool Generator* para que efectúe la configuración de aplicaciones a nivel de la compilación de clases y no sólo en la construcción de la interfaz gráfica.
- Adaptación de la aplicación *Meshing Tool Generator* para que despliegue las opciones en idioma inglés.

1.5. Contenido de la memoria

El presente informe está estructurado en los siguientes capítulos:

- **Introducción**: Introducción al problema.
- **Antecedentes**: Antecedentes que facilitan el entendimiento del problema y su solución.
- **Análisis**: Descripción del estado y complejidad del software Simulador de crecimiento de árboles al inicio del desarrollo del presente trabajo. Identificación de las clases que dificultan la configuración y, por lo tanto, deben ser modificadas.

- **Diseño**: Descripción de las decisiones de diseño tomadas durante la realización del presente trabajo. Modelo de características para las aplicaciones del dominio, y diseño de la interfaz y de clases para la nueva versión de *Meshing Tool Generator*.
- **Implementación de *Meshing Tool Generator***: Descripción en detalle de las clases que componen la aplicación *Meshing Tool Generator*, su comportamiento y algunos ejemplos de su funcionamiento.
- **Ejemplos y evaluación de resultados**: Ejemplos de productos generados a través de la aplicación *Meshing Tool Generator*, y comparación de los resultados obtenidos.
- **Conclusiones**: Conclusiones con respecto al trabajo realizado, y discusión sobre el posible trabajo futuro que se puede desarrollar.

Capítulo 2
Antecedentes

En este capítulo se exponen los antecedentes necesarios para facilitar el entendimiento del trabajo realizado. Se explican los temas estudiados en la revisión bibliográfica que precedió este trabajo, y se revisan tanto el diseño como las funcionalidades del software TGS.

2.1. Líneas de productos de software

La forma en que se producen los bienes ha cambiado significativamente con el paso del tiempo. En un comienzo, los productos eran manufacturados para clientes individuales, de acuerdo a sus necesidades específicas. Luego, a medida que el poder adquisitivo de los consumidores creció, también lo hizo la demanda de distintos tipos de productos. En este escenario, el empresario Henry Ford concibió el concepto de "Líneas de productos", el cual permitió la construcción de productos para un mercado masivo con menores costos que la producción manual [6], modificando de manera radical las costumbres y los hábitos de consumo de la sociedad.

Aun cuando al comienzo los consumidores estaban satisfechos con los bienes de producción masiva, no todos ellos veían completamente resueltas sus necesidades. Por esto, la industria se vio enfrentada a una demanda creciente por productos individualizados, dando inicio así a la "Personalización masiva", concepto que se define como la producción a gran escala de bienes hechos a la medida de las necesidades de los clientes. Para que este nuevo modelo no generara costos excesivos, las compañías introdujeron en su implementación el concepto de "plataforma", el cual se define como la base tecnológica sobre la cual otras tecnologías o procesos son elaborados, es decir, una especie de "ladrillo" con el cual se puede construir el producto deseado.

Las principales motivaciones que tienen las compañías para adoptar este paradigma son las siguientes:

- **Reducción en los costos de desarrollo**: Cuando las componentes pertenecientes a la plafatorma son reutilizadas en distintos sistemas, el costo de cada uno de ellos disminuye. Sin embargo, antes de que dichas componentes se puedan reutilizar, es necesaria una inversión inicial para crearlas. A pesar de esto, estudios empíricos indican que al haber desarrollado tres sistemas, los costos de producirlos individualmente o

mediante líneas de productos se igualan. De allí en adelante, la alternativa de líneas de productos es más barata.

- **Aumento de la calidad**: Las componentes de la plataforma son revisadas y testeadas durante el desarrollo de múltiples productos, ya que deben funcionar correctamente como parte de todos ellos. Este proceso de aseguramiento de calidad implica una probabilidad mayor de detección y posterior corrección de errores, incrementando así la calidad de los productos que integran.

- **Reducción del "Time to Market"**: Tal como ocurre con los costos, al usar líneas de productos el tiempo que tarda un sistema en salir al mercado es mayor en un principio, dado a la inversión inicial de tiempo que se necesita para crear las componentes de la plataforma; sin embargo, una vez pasado este obstáculo, los productos tardan considerablemente menos en ser terminados.

- **Mejora en la estimación de costos**: Al mantener un conjunto de componentes base, es más fácil calcular el precio de los productos a desarrollar, pues dichas componentes son ya conocidas. Además, las posibles componentes adicionales serán de menor tamaño y por lo tanto menos complejas de estimar.

- **Usabilidad**: Al mantener componentes comunes entre sistemas, los clientes se ven beneficiados, pues pueden familiarizarse con su uso y adaptarse con facilidad al usar los distintos productos.

La combinación del modelo de personalización masiva y el concepto de plataforma permite la reutilización de una base común de tecnologías, y a la vez la satisfacción de las necesidades de los clientes. Esta combinación, aplicada al dominio de la producción de software, se denomina "Ingeniería de Líneas de Productos de Software" y consta de dos etapas: Ingeniería de Dominio e Ingeniería de Aplicación. Ambas etapas, a su vez, se componen de los siguientes subprocesos: Requerimientos, Diseño, Implementación y Pruebas. Dichos subprocesos siguen un orden cíclico, como se puede apreciar en la Figura 2.1 [6].

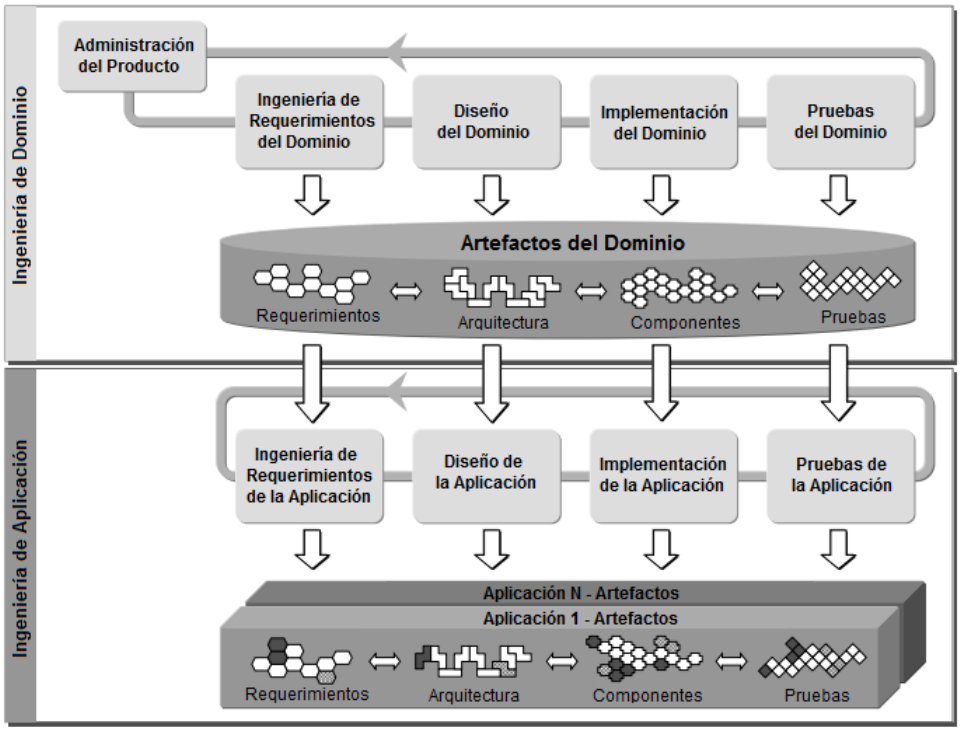

Figura 2.1: Flujo de procesos para la ingeniería de líneas de productos de software.

2.1.1. Ingeniería de Dominio

Esta etapa se encarga de elaborar la plataforma de componentes reutilizables, identificando similitudes y variabilidades. Esta plataforma, también llamada "modelo de dominio" consta de todo tipo de componentes de software (requerimientos, diseño, código, pruebas, etc.), y debe ser lo suficientemente flexible como para habilitar la construcción de una amplia gama de productos.

La ingeniería de dominio incluye, además de los ya mencionados subprocesos, el subproceso de "Administración del Producto", el cual se encarga de definir qué componentes serán incluidas en la línea de productos, tomando en cuenta los objetivos de la compañía y la estrategia de mercado. Su objetivo es producir una hoja de ruta (o *roadmap*) tecnológica, que detalla la lista de componentes a desarrollar y sus fechas de lanzamiento, junto con la lista de componentes y/o productos ya existentes que pueden ser reutilizados para establecer la plataforma.

La variabilidad es un concepto esencial dentro de la ingeniería de dominio. Es introducida en el proceso de Administración del Producto, cuando se identifican las características comunes y variables para todos los productos. Dado que el subproceso de Requerimientos del Dominio detalla la descripción de dichas características, la variabilidad también se incluye en él, y lo mismo ocurre para los demás subprocesos.

Para identificar y caracterizar la variabilidad, es necesario formular las siguientes preguntas [6]:

- **¿Qué es lo que varía?**: Permite definir los elementos o propiedades variables para el conjunto de productos. Dichos elementos o propiedades son denominados *sujetos variables*.
- **¿Por qué varía?**: Hay diferentes razones por las que un elemento o propiedad puede variar: necesidades de la empresa, leyes vigentes, motivos técnicos, etc.
- **¿Cómo varía?**: Permite definir las distintas formas que un elemento o propiedad puede tomar. Estas formas son instancias de los sujetos variables, y son denominadas *objetos variables*.

Una vez respondidas estas preguntas, es posible incorporar los sujetos y objetos variables en el contexto de una línea de productos de software, introduciendo así los conceptos de *punto variable*, que consiste en una representación de un sujeto variable dentro de los artefactos del dominio, y *variante*, cuya definición es la misma pero aplicada a los objetos variables [6].

Los puntos variables y las variantes se utilizan para definir la variabilidad de una línea de productos de software. Por esto, es esencial que la identificación de ambos elementos sea efectuada de forma sistemática; en primer lugar, es necesario identificar al sujeto variable en el mundo real, después definir el punto variable que representa a dicho sujeto en el dominio, y finalmente establecer las variantes posibles para el punto ya definido.

De esta misma manera se definen las semejanzas, identificando las características que forman parte de todos los productos, y que funcionan de la misma manera para todos ellos. En el proceso de ingeniería de dominio, la identificación de semejanzas precede a la de la variabilidad.

Existen distintos tipos de variabilidad; uno de ellos es la variabilidad en el tiempo, la cual se define como "la existencia de diferentes versiones de un artefacto, que son válidas en momentos distintos" [6]. Este tipo de variabilidad responde a un proceso natural para cualquier producto de software y los artefactos que lo componen: la evolución en el tiempo. Dentro de los artefactos del dominio de una línea de productos de software, la ubicación de los puntos variables es conocida; por lo tanto, introducir cambios en ellos a través del tiempo es relativamente sencillo.

Otro tipo de variabilidad es la variabilidad en el espacio, la cual se define como "la existencia de un artefacto en distintas formas en un mismo espacio de tiempo" [6]. Este concepto es vital, puesto que el objetivo de la ingeniería de líneas de productos de software es construir productos similares que varían dentro de un alcance determinado, y por lo general son comercializados al mismo tiempo.

También se puede definir la variabilidad dependiendo del origen de ésta, generando así los conceptos de variabilidad interna o externa. La variabilidad externa consiste en "la variabilidad de los artefactos del dominio que es visible a los clientes" [6]. Esto les brinda a los clientes el poder de decidir qué variantes necesitan y cuáles no. La variabilidad interna se define como "la variabilidad de los artefactos del dominio que no es visible para los clientes", y surge de la necesidad de implementar o refinar las variabilidades externas. Como al cliente no le interesan los aspectos de más bajo nivel de un sistema, no es necesario comunicarle las

decisiones tomadas al nivel de variabilidad interna.

Dado el hecho que mientras más variabilidades externas se definan, mayor será el número de variabilidades internas, es importante mantener al mínimo las primeras, ya que traen consigo un aumento en la complejidad y los costos de los sistemas.

Una de las técnicas más utilizadas dentro de la ingeniería de requerimientos del dominio es el "Análisis de Dominio Orientado a Características" (*Feature Oriented Domain Analysis*, o *FODA*, en inglés), que identifica y clasifica similitudes y variabilidades en términos de "características de los productos" (*Product Features* en inglés). Este análisis es después utilizado para el desarrollo de un modelo de características (*Feature Model* en inglés), sobre el cual se construye el conjunto de componentes reutilizables que permitirán la generación de múltiples productos pertenecientes al dominio; en este modelo, las similitudes se muestran como características obligatorias, mientras que las variabilidades se dividen en dos grupos: características alternativas, donde sólo una puede ser elegida para un producto, y características inclusivas, donde más de una puede ser elegida [3].

Un ejemplo de modelo de *Features* para generadores de mallas geométricas, construido con esta técnica, se puede observar en la Figura 2.3 [8]. Este modelo incluye *Features* para tres tipos distintos de generadores de mallas:

- **Mallas 2D**: Mallas compuestas por polígonos convexos simples, que permiten modelar superficies planas. En este modelo se consideran las mallas compuestas por triángulos, por cuadriláteros y mallas mixtas (compuestas por ambos tipos de figuras).
- **Mallas 3D**: Mallas compuestas por poliedros, que permiten modelar volúmenes. Existen mallas compuestas por tetraedros, prismas, pirámides y hexaedros, entre otros tipos de poliedros convexos simples.
- **Mallas 2.5D**: Mallas compuestas por polígonos convexos simples (triángulos y/o cuadriláteros), que permiten modelar superficies en el espacio 3D. La presente memoria permite crear una linea de productos generadores de mallas de este tipo.

En la Figura 2.2 se muestra un ejemplo de cada malla: la malla al lado izquierdo es una triangulación en el plano, la malla del centro es una triangulación de superficie de un cilindro y la malla de la derecha es una malla de volumen, compuesta de tetraedros, que representa un conejo.

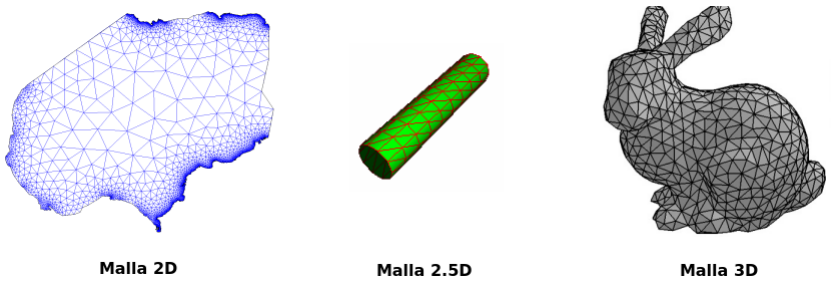

Figura 2.2: Ejemplos de mallas geométricas 2D, 2.5D y 3D.

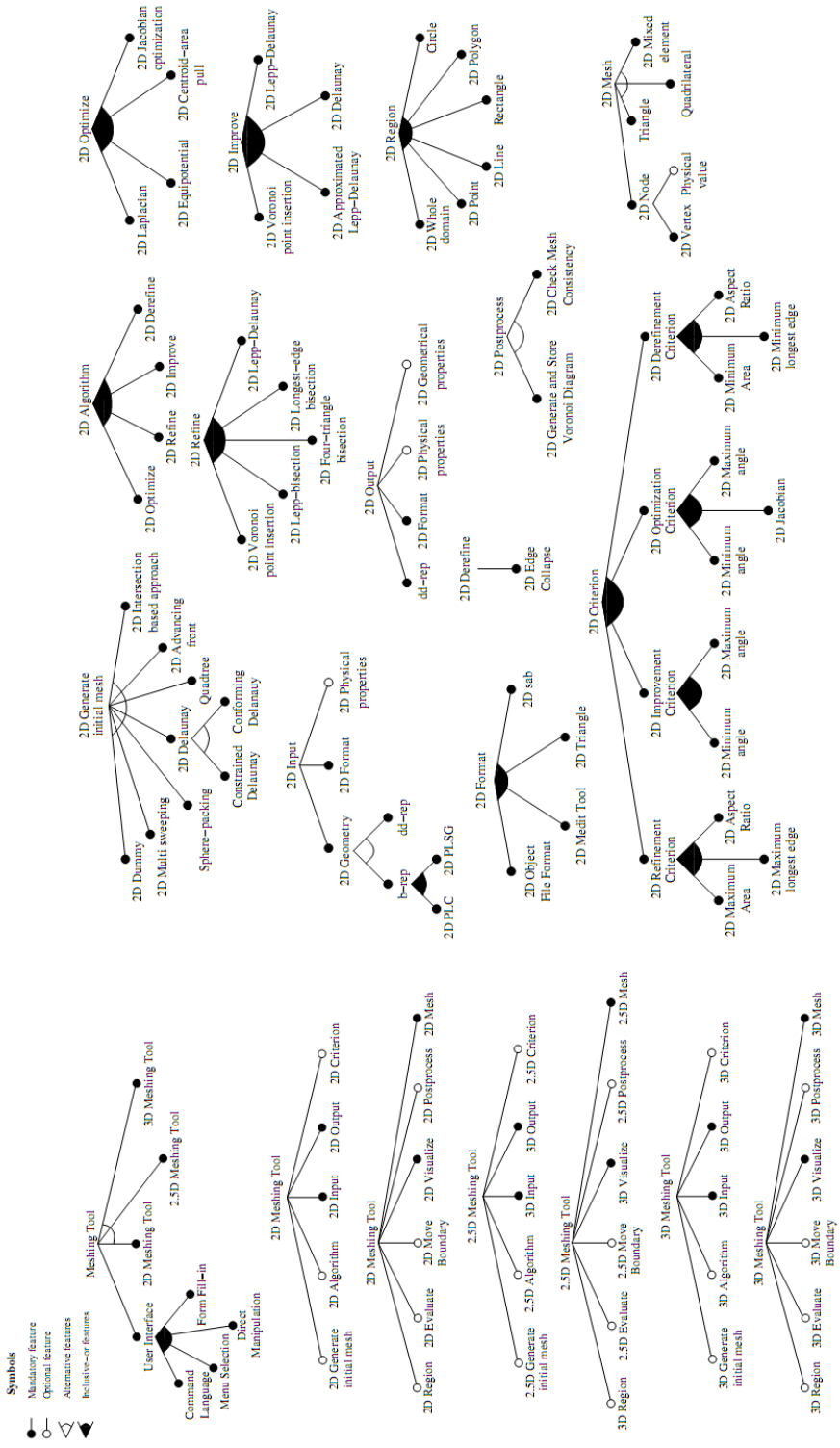

Figura 2.3: Modelo de *Features* para el dominio de mallas geométricas.

El modelo descrito se enmarca en la etapa de Ingeniería de Requerimientos del Dominio, mostrada en la Figura 2.1. Esta plataforma se utiliza como base para la línea de productos desarrollada en la presente memoria.

2.1.2. Ingeniería de Aplicación

Esta etapa es responsable de construir cada uno de los sistemas pertenecientes a la línea de productos a partir de la plataforma establecida en la Ingeniería de dominio. Su objetivo principal es lograr el mayor grado de reutilización posible de las componentes de dicha plataforma al definir y desarrollar un producto, documentando en el proceso los artefactos producidos (requerimientos, arquitectura, código, pruebas, etc.) y enlazándolos a estas componentes.

Como muestra la Figura 2.1, la Ingeniería de aplicación consta de cuatro subprocesos [6]:

- **Requerimientos**: Este proceso recibe como entrada los requerimientos del dominio junto con la hoja de ruta obtenida en el proceso de Administración del Producto. A partir de ambas componentes, busca desarrollar la especificación de requerimientos para el producto a construir, determinando qué requerimientos del dominio sirven para conseguir la implementación deseada y tomando en cuenta el hecho que puede haber requerimientos específicos que no hayan sido capturados durante la Ingeniería de Requerimientos del dominio.

- **Diseño**: Este proceso recibe como entrada la especificación de requerimientos desarrollada en la etapa anterior, junto con la arquitectura de referencia desarrollada en la Ingeniería de Diseño del dominio, y se encarga de producir el diseño de arquitectura para el producto a construir, mediante seleccionar y configurar las partes necesarias de la arquitectura del dominio, adaptándolas si es necesario según lo especificado en los requerimientos. Es importante notar que esta etapa difiere en gran manera del proceso de Diseño para un sólo sistema ya que, en vez de desarrollar la arquitectura desde cero, la deriva de una plataforma ya construida.

- **Implementación**: Este proceso se encarga de crear el producto, lo cual requiere seleccionar y configurar las componentes de software disponibles en la plataforma del dominio, además de implementar las componentes específicas para la aplicación a construir, adaptando estas últimas a las interfaces y módulos ya existentes. Tal como en los procesos anteriores, esta etapa se diferencia del proceso de Implementación para un sólo sistema, pues reutiliza partes ya construidas en vez de construir todo el sistema desde cero, tomando como base la arquitectura creada en el proceso de Diseño.

- **Pruebas**: Este proceso se encarga de validar la aplicación, verificando que su funcionamiento sea consistente con la especificación de requerimientos. Recibe como entrada la aplicación ya construida, junto con los elementos de prueba disponibles en la plataforma del dominio, y produce un informe con los resultados de todas las pruebas ejecutadas. Adicionalmente, los defectos detectados son documentados en detalle, lo cual provee una retroalimentación valiosa para la línea de productos, pues permite descubrir las configuraciones defectuosas y además repararlas.

2.2. Aplicaciones del dominio

En la presente sección, se describen las dos aplicaciones generadoras de mallas geométricas que constituyen la base para el desarrollo de la nueva línea de productos, y componen el dominio de esta línea.

2.2.1. Simulador de crecimiento de árboles (*Tree Growth Simulator*)

El software TGS fue desarrollado en la Universidad de Chile, en el marco de tres memorias de título anteriores [4, 5, 9], basándose en los conceptos descritos en el Generador genérico de mallas. Fue desarrollado en el lenguaje C++, y su objetivo principal es el estudio y modelamiento computacional del crecimiento de árboles y la deformación de sus ramas, en particular de la superficie cilíndrica de sus troncos, a través de mallas geométricas de triángulos o cuadriláteros.

En términos generales, permite crear mallas iniciales simples, cargar mallas desde distintos formatos de entrada y guardarlas también en varios formatos de salida. Además, permite la visualización gráfica de las mallas y su modificación a través de diversos algoritmos.

De forma más detallada, la aplicación cuenta con las siguientes funcionalidades:

- **Crear una malla**: Es posible crear mallas a partir de una médula (una curva generatriz de la malla que modela el centro del tronco del árbol) y mallas cilíndricas.
- **Cargar una malla**: Se puede cargar mallas en distintos formatos de entrada.
- **Visualizar una malla**: La aplicación dibuja la malla al ser creada o cargada. El usuario puede elegir si desea ver o no las caras y/o arcos de la malla, y además mover la cámara de manera de acercarla, alejarla o rotar la malla.
- **Guardar una malla**: Las mallas pueden ser guardadas en distintos formatos de salida.
- **Deformar una malla**: La deformación consiste en el desplazamiento de cada nodo de la malla según la concentración de hormona de crecimiento presente en él, y en una dirección asociada al nodo. El usuario indica el porcentaje de la concentración que se moverá cada nodo en un paso, y el número de pasos que se moverá la malla. Este desplazamiento puede producir colisiones dentro de la malla, por lo que se implementan tres algoritmos para tratarlas: *No Verificación*, que simplemente desplaza la malla, *Verificación Local*, que recorre pares de caras vecinas e intenta corregir las inconsistencias haciendo *edge flipping*, y *Verificación Nodos Vecinos*, que recorre los nodos y, si descubre que el desplazamiento provoca una inconsistencia en la malla (por ejemplo, la intersección de dos o más caras), intenta corregir su trayectoria.
- **Refinar una malla**: Esto se logra aumentando el número de caras de la malla, disminuyendo el área de las caras originales. El usuario elige qué algoritmo se usará (entre *Lepp-Delaunay* y *Delaunay Longest Edge Bisection*), junto con su criterio de detención, el cual puede ser: ángulo mínimo, área cara máxima o largo arco máximo.

- **Desrefinar una malla**: Esto se logra disminuyendo el número de caras de la malla, aumentando el área de las caras sobrevivientes. Actualmente, el único algoritmo implementado es *Edge Collapse*, el cual recorre la malla buscando caras que no cumplan el criterio para aplicar al algoritmo; luego detecta el menor arco de cada cara seleccionada y lo colapsa en un nodo, eliminando las dos caras adyacentes a él. El usuario elige el criterio de detención del algoritmo, el cual puede ser: área cara mínima, o largo arco mínimo.

- **Mejorar una malla**: Es posible aplicar un algoritmo de mejoramiento de la calidad de la malla. Actualmente, el único algoritmo implementado es *Mejora Delaunay*, el cual recorre la malla buscando pares de triángulos vecinos que no cumplan el criterio de Delaunay (es decir, existen vértices al interior del círculo circunscrito a él). Luego, reemplaza el arco común de los triángulos por el otro posible (*edge flipping*), haciendo que ambos triángulos cumplan con el criterio.

- **Distribuir hormona**: Este procedimiento simula la distribución de la hormona de crecimiento a lo largo del tronco de un árbol, utilizando un algoritmo aleatorio. Este algoritmo asigna un valor escalar a cada nodo de la malla, el cual es usado por el algoritmo de deformación para hacer crecer o deformar las ramas del árbol.

- **Desplegar la información de la malla**: La aplicación muestra un cuadro con la siguiente información sobre la composición de la malla: número de caras, arcos y nodos, área mínima, máxima y promedio de las caras, largo mínimo, máximo y promedio de los arcos, y ángulo mínimo y máximo. Estos datos son de utilidad para el usuario, pues le permiten decidir si refinar o desrefinar la malla.

El modelo de clases de este sistema se puede apreciar en la Figura 2.4.

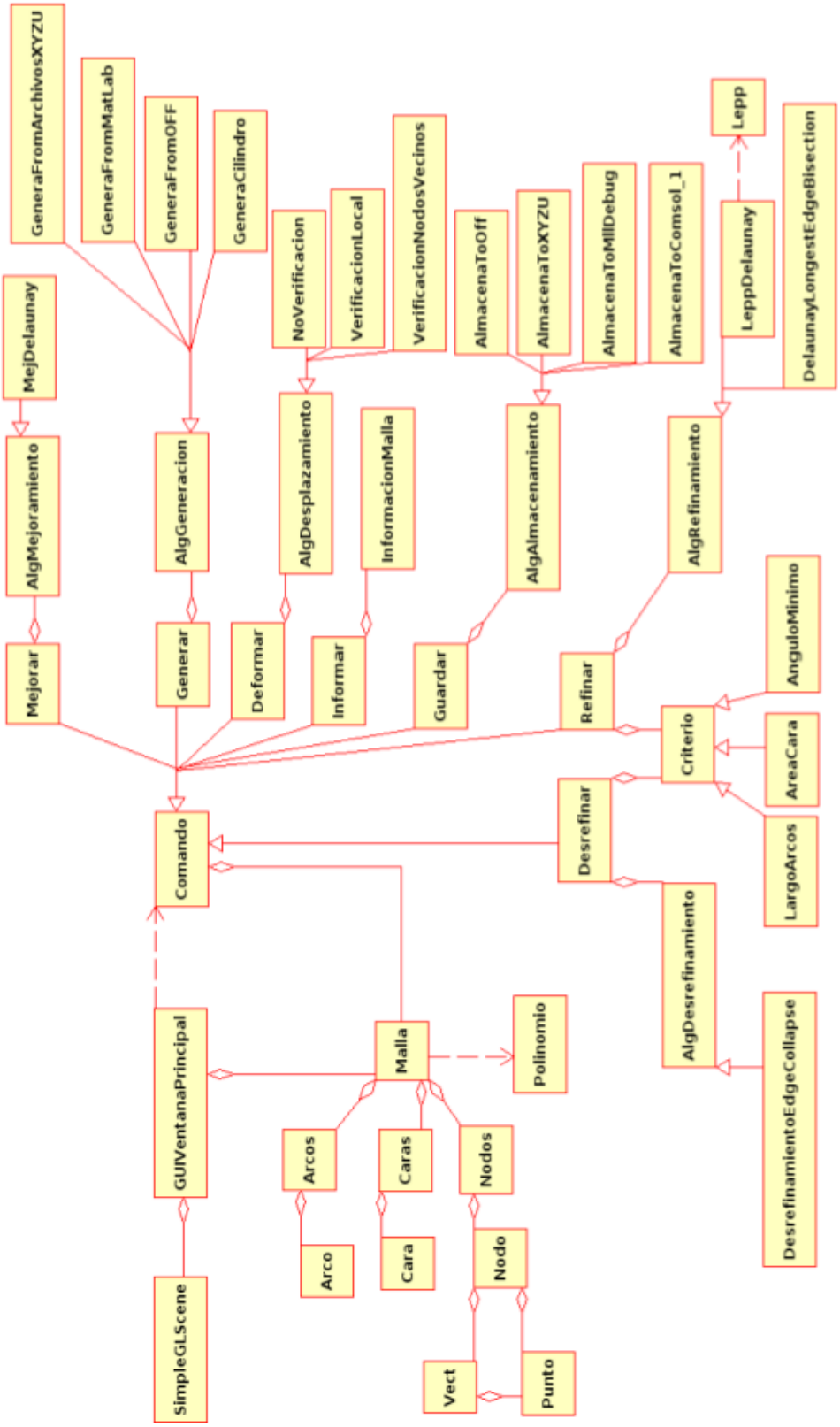

Figura 2.4: Diagrama de clases del software TGS.

Este diagrama corresponde al desarrollado por Nicolás Silva [9]; en principio se escogería la última versión, pero esta alternativa se descartó por poseer un alto acoplamiento entre las funcionalidades de mallas de triángulos y cuadriláteros.

2.2.2. Generador genérico de mallas

Esta aplicación genera mallas 2D y constituye una base conceptual para el software TGS. Incluye todas las funcionalidades descritas en la sección anterior, exceptuando las siguientes (por estar estrictamente relacionadas con el modelamiento de árboles):

- **Crear una malla**
- **Distribuir hormona**

2.3. *Meshing Tool Generator*: Primer acercamiento a una LPS

El trabajo de esta memoria se basa en la aplicación desarrollada por Gonzalo Urroz en su trabajo de título [10]. Esta aplicación, llamada *Meshing Tool Generator*, consiste en una interfaz que permite al usuario elegir la configuración de las funcionalidades que desee al utilizar cualquiera de las aplicaciones del dominio.

Este sistema utiliza como base las siguientes dos aplicaciones de dominio: el software TGS, y un sistema de animación de rostros llamado *FaceAnimator* [11], que permite la simulación del comportamiento de rostros humanos. Incluye los siguientes elementos de variabilidad o *variabilities*:

- **Algoritmo para Generar Malla Inicial**. Su única variante es: Registration Method.
- **Algoritmo de Refinamiento**. Sus variantes son: Delaunay Longest Edge Bisection, Lepp-Delaunay.
- **Algoritmo de Desrefinamiento**. Su variante es: Edge-collapse.
- **Algoritmo de Deformación**. Sus variantes son: Crear animación y Deformación CD3.
- **Algoritmo de Mejoramiento**. Su variante es: Mejoramiento de Delaunay.
- **Algoritmo para Guardar Malla Final**. Su variante es: Formato de Almacenamiento.
- **Idioma**. Corresponde al idioma en el cual son presentados los mensajes y las interfaces gráficas. Sus variantes son: Inglés y Español.

En el sistema *Meshing Tool Generator*, el usuario selecciona las variantes para cada punto variable a través de un menú. Luego, con los valores seleccionados, el programa escribe un archivo llamado `build_option.txt`, en el cual se encuentran las instrucciones para construir el producto deseado.

Sin embargo, este sistema utiliza versiones muy acopladas de las aplicaciones del dominio; las arquitecturas de ambos sistemas no permiten la separación de sus componentes de manera que éstos sean independientes y puedan ser compilados utilizando sólo las clases asociadas a la configuración elegida. Esto deriva en la inclusión de la mayoría de las clases al momento de compilar el producto, lo cual genera aplicaciones con implementaciones iguales, que sólo cambian en la organización del menú. Por lo tanto, aunque hay una sola interfaz para seleccionar distintas componentes, se termina incluyendo más código del necesario. Esto hace necesario un rediseño de las clases que implementan los distintos puntos de variabilidad, de manera de disminuir el acoplamiento, es decir, que ya no exista una dependencia innecesaria entre las clases que se incluirán en la compilación final y las candidatas a ser retiradas.

Otro importante aspecto mejorable de *Meshing Tool Generator* es que, durante su ejecución, la interfaz principal de los productos generados se mantiene igual sin importar las alternativas elegidas por el usuario; esto aumenta la complejidad y disminuye la robustez del sistema, lo cual se puede solucionar a través de interfaces separadas que muestren sólo las opciones disponibles para el producto que se está generando.

Capítulo 3
Análisis

En el presente capítulo se explica en qué parte del flujo de procesos de líneas de productos de software, mostrado en la Figura 2.1, se inserta la presente memoria, junto con las tareas a realizar en cada paso. Además, se describe y analiza el estado y complejidad del software TGS, el cual será usado como base para construir la línea de productos.

3.1. Etapas a seguir para el desarrollo de la línea de productos generadores de mallas

Dado que la Ingeniería de Líneas de Productos de Software consta de dos etapas, Ingeniería de Dominio e Ingeniería de Aplicación, se explicará cómo se enmarcan los pasos a trabajar en la presente memoria dentro de ambas.

3.1.1. Ingeniería de Dominio

Dentro de esta etapa, se utilizó como base la plataforma ya existente, compuesta de los siguientes dos elementos:

- El modelo de *features* mostrado en la Figura 2.3, que constituye una plataforma de requerimientos para el dominio de los software generadores de mallas geométricas, y provee una definición de sus similitudes y variabilidades.
- El código del software TGS, el cual constituye una plataforma de componentes ya desarrolladas que pueden formar parte de un software generador de mallas geométricas.

A partir de ambos elementos, y tomando en cuenta la definición de las dos aplicaciones que forman parte de la nueva línea de productos (Software TGS y Generador genérico de mallas), se llevaron a cabo los cuatro subprocesos descritos en el capítulo 2:

- **Requerimientos**: Esta etapa consistió en examinar el modelo de *features* ya existente y determinar qué partes de dicho modelo corresponden al software TGS, y cuáles corresponden al Generador genérico de mallas.
- **Diseño**: Si bien no existe un diseño de arquitectura para el software base, en la presente memoria se realizó un diseño de las componentes necesarias para cada producto a construir, tomando en cuenta los requerimientos generados en el paso anterior.

- **Implementación**: Este subproceso involucró el análisis del código existente, identificando las clases más acopladas, es decir, las que impiden una correcta separación de componentes, e interviniéndolas para solucionar este problema y así generar una plataforma de componentes reutilizables que sean útiles al momento de construir un producto generador de mallas.

- **Pruebas**: No existía un conjunto predefinido de pruebas para el software base, por lo cual fue necesario crear una batería de casos de prueba, que permitiesen validar que una malla generada por un producto construido a partir de la línea de productos es idéntica a una generada por el software base.

3.1.2. Ingeniería de Aplicación

La aplicación construida en la presente memoria, llamada *Meshing Tool Generator*, se encarga de llevar a cabo automáticamente el subproceso de Implementación, ensamblando las componentes necesarias para construir un producto generador de mallas geométricas, dependiendo de lo que el usuario necesite. Luego, se ejecutan manualmente los casos de prueba creados en la etapa de Ingeniería de Dominio.

3.2. Aplicación de métricas sobre el software TGS

Las métricas de calidad son mediciones que permiten la evaluación de un sistema, ya sea de su código o del diseño de clases. Estas mediciones son interpretadas para obtener una visión de la calidad de dicho sistema, y los resultados de esta interpretación conducen a la realización de cambios que permitan una mayor reutilización de las componentes del sistema (lo cual implica una mayor mantenibilidad y flexibilidad), una reducción de su complejidad, un aumento de la cohesión de sus módulos, etc.

Dada la utilidad de estos indicadores de calidad, es importante evaluarlos para el software TGS. Esto fue logrado gracias a la herramienta Understand C++ [1], la cual permitió obtener las siguientes métricas a nivel de aplicación:

Métrica	Valor
N° de clases	77
N° de archivos	132
N° de líneas	12958
N° de líneas en blanco	1754
N° de líneas de código	9017
N° de líneas de comentarios	1609
Radio comentarios/código	0.18

Tabla 3.1: Métricas del software TGS, a nivel de aplicación.

[1] http://www.scitools.com/index.php

Además, fueron obtenidas las siguientes métricas a nivel de clase [7]:

- **LCOM (Carencia de cohesión en los métodos)**: Esta métrica utiliza el concepto de "grado de similaridad" entre dos métodos, el cual se define como el conjunto de atributos a los que ambos acceden. LCOM se obtiene mediante la resta de los siguientes dos valores: el número de pares de métodos cuya similaridad es cero, y el número de pares de métodos cuya similaridad es distinta de cero. Un valor alto de LCOM implica falta de cohesión, lo cual puede indicar que la clase está compuesta de elementos no relacionados, y por lo tanto debe ser dividida en dos o más clases; además, la complejidad y la probabilidad de errores durante el desarrollo aumentan. Es deseable, por lo tanto, una alta cohesión entre los métodos de una clase, pues esto fomenta el uso de encapsulación.

- **DIT (Profundidad de herencia)**: Se define como la profundidad de la clase en la jerarquía de herencia. Se utiliza como medida de la complejidad de una clase y de su potencial reuso; esto es debido a que cuanto más profunda se encuentra una clase en la jerarquía, mayor será el número de posibles métodos heredados.

- **CBO (Acoplamiento entre objetos)**: Se dice que una clase está "acoplada" con otra cuando utiliza sus métodos o atributos; esta métrica se define como el número de clases con las cuales está acoplada una determinada clase, sin tomar en cuenta las clases relacionadas por herencia. Un valor alto de CBO implica una menor modularidad, extensibilidad y reuso para la clase, además de un mayor esfuerzo en su mantención; por lo tanto, es deseable mantener el valor de esta métrica tan bajo como sea razonable, pues así se reduce la complejidad, se aumenta la modularidad y se promueve la independencia entre clases.

- **NOC (Número de hijos)**: Corresponde al número de subclases subordinadas a una clase en la jerarquía de herencia. Un mayor valor de esta métrica implica un mayor reuso para la clase, además de un mayor impacto de ésta en el desarrollo.

- **RFC (Respuesta de una clase)**: Consiste en el número de métodos de una clase que pueden ser potencialmente ejecutados, en respuesta a un mensaje recibido por un objeto. Un mayor valor de RFC implica una mayor complejidad en la comunicación entre componentes u objetos, y por lo tanto un mayor esfuerzo para la depuración y pruebas del código.

- **WMC (Peso de los métodos de una clase)**: Esta métrica se obtiene calculando la complejidad para cada método de una clase, y sumando todos los valores obtenidos. Se utiliza como indicador de cuánto tiempo y esfuerzo son requeridos para desarrollar y mantener la clase. Además, a medida que crece el número de métodos para una clase, el potencial de reuso disminuye. Por lo tanto, es deseable mantener el valor de WMC tan bajo como sea razonable.

- **LOC (Número de líneas de código)**: Corresponde al número de líneas activas de código (líneas ejecutables) en una clase. Este valor es utilizado para evaluar la comprensibilidad, reusabilidad y mantenibilidad del código.

Las métricas obtenidas para las clases más importantes se muestran en la Tabla 3.2.

Clase	LCOM	DIT	CBO	NOC	RFC	WMC	LOC
Arco	50	0	0	0	17	24	105
Arcos	50	0	1	0	13	23	115
Cara	62	0	1	0	20	37	166
Caras	47	0	1	0	11	17	90
Comando	50	0	1	8	4	3	15
Deformar	0	1	5	0	7	6	24
Desrefinar	0	1	5	0	7	6	28
DistribuirHormona	75	1	8	0	11	25	131
Generar	0	1	8	0	9	11	58
Guardar	16	1	6	0	7	7	28
GUIDialogCambios_glade	80	1	16	1	5	2	168
GUIDialogInformacion_glade	66	1	11	1	3	2	590
GUIDialogNuevaMalla_glade	75	1	15	1	4	2	330
GUIDialogRefinar_glade	80	1	14	1	5	2	186
GUIVentanaPrincipal	63	2	27	0	60	76	466
GUIVentanaPrincipal_glade	96	1	22	1	29	2	487
Malla	77	0	9	0	70	268	1703
Mejorar	0	1	3	0	7	3	17
Nodo	81	0	3	0	21	38	154
Nodos	52	0	4	0	18	45	196
Refinar	0	1	6	0	7	8	32

Tabla 3.2: Métricas del software TGS, a nivel de clase.

3.3. Evaluación de los resultados obtenidos

Es necesario notar que la métrica que más afecta al desarrollo de una línea de productos de software es CBO, puesto que, si una clase posee un alto valor de dicha métrica, será muy difícil separarla de las clases asociadas a ella, y por lo tanto será muy difícil generar una configuración que implique excluir alguna de estas clases.

Luego de estudiar el código del software TGS y analizar las métricas obtenidas (en particular CBO), se observó que existe un alto nivel de acoplamiento y muy baja cohesión en las clases encargadas de construir la interfaz del programa, en particular en las correspondientes a la ventana principal (`GUIVentanaPrincipal` y `GUIVentanaPrincipal_glade`). En particular, la clase `GUIVentanaPrincipal`, que implementa la lógica detrás de la interfaz, contiene llamadas a métodos y constructores de prácticamente todas las demás clases, lo cual la hace extremadamente difícil de mantener y extender. Esto además dificulta la configuración del software TGS, ya que si se desea remover alguna funcionalidad, como por ejemplo refinar una malla, no basta con quitar los archivos correspondientes, dado que la clase `GUIVentanaPrincipal` los necesita y por lo tanto la compilación del software fallará.

Dado que esta clase es de gran tamaño (466 líneas de código), se dificulta la posible adición de funcionalidades a la aplicación, pues sería necesario agregar los métodos correspondientes a `GUIVentanaPrincipal`.

Otro aspecto a considerar es que cada clase que representa un algoritmo o funcionalidad dentro de TGS contiene llamadas a los constructores de sus variantes asociadas (definidas usando herencia), lo cual dificulta la elección de una u otra variante, pues si se elimina alguna de las clases invocadas ocurrirá lo mismo que en el caso anterior: la compilación del software fallará. Un ejemplo de esto se puede ver en la Figura 3.1, la cual muestra el constructor de la clase `Refinar`.

```
#include "refinar.h"
#include "leppdelaunay.h"
#include "delaunaylongestedgebisection.h"
#include "angulominimo.h"
#include "areacara.h"
#include "largoarcomaximo.h"

Refinar::Refinar(Malla *m, int algoritmo, int criterio, double valorCriterio): Comando(m) {
    assert(m);

        this->algoritmo=NULL;
        this->criterio=NULL;

        if(algoritmo==0)
                this->algoritmo=new LeppDelaunay();
        else if(algoritmo==1)
                this->algoritmo=new DelaunayLongestEdgeBisection();

        if (criterio==0)
                this->criterio=new AnguloMinimo(valorCriterio);
        else if (criterio==1)
                this->criterio=new AreaCara(valorCriterio);
        else if (criterio==2)
                this->criterio=new LargoArcoMaximo(valorCriterio);

        assert(this->algoritmo && this->criterio);
        }
```

Figura 3.1: Constructor de la clase *Refinar* del software TGS.

En este caso, el constructor contiene referencias a los constructores de las clases `LeppDelaunay` y `DelaunayLongestEdgeBisection`, que constituyen las variantes del algoritmo de refinamiento, y a los constructores de las clases `AnguloMinimo`, `AreaCara` y `LargoArcoMaximo`, que corresponden a las variantes del criterio de detención del algoritmo. Esto ocurre en todas las demás clases que representan algoritmos o funcionalidades que poseen variantes, que corresponden a las siguientes:

- `Generar`: Contiene referencias a los constructores de las clases `GeneraFromOFF`, `GeneraFromMatlab`, `GeneraFromArchivosXYZU` y `GeneraFromComsol_1`, que corresponden a las variantes del algoritmo de carga de mallas, y a los constructores de las clases `GeneraCilindro` y `GeneraFromMedula`, que corresponden a las variantes del algoritmo de generación de mallas.
- `Guardar`: Contiene referencias a los constructores de las clases `AlmacenaToOff`, `AlmacenaToXYZU` y `AlmacenaToComsol_1`, que corresponden a las variantes del algoritmo de almacenamiento de mallas.

- `Desrefinar`: Contiene referencias a los constructores de las clases `AreaCara` y `LargoArcoMinimo`, que corresponden a las variantes del criterio de detención del algoritmo de desrefinamiento.
- `Deformar`: Contiene referencias a los constructores de las clases `NoVerificacion`, `VerificacionLocal` y `VerificacionNodosVecinos`, que corresponden a las variantes del algoritmo de tratamiento de posibles colisiones.

Por lo tanto, si se desea configurar el software TGS, es imperativo modificar todas estas clases y así poder incluir sólo las variantes deseadas en el producto final. En el caso de la clase `GUIVentanaPrincipal`, es necesario modificarla para poder incluir sólo las funcionalidades deseadas en el producto final.

Las modificaciones mencionadas permiten una configuración en tiempo de compilación; sin embargo, es necesario además facilitar la configuración en tiempo de ejecución. Esto requiere modificar las clases que implementan las interfaces de la aplicación, que son las siguientes:

- `GUIVentanaPrincipal_glade`: Corresponde a la interfaz principal, y debe ser modificada para que muestre sólo los botones correspondientes a las funcionalidades incluidas en la aplicación, ocultando los que no sean necesarios.
- `GUIDialogCambios_glade`: Corresponde a la ventana que se despliega al seleccionar la opción *Deformar* en la interfaz principal, y debe ser modificada para que muestre sólo las variantes deseadas para el algoritmo de deformación.
- `GUIDialogDesrefinar_glade`: Corresponde a la ventana que se despliega al seleccionar la opción *Desrefinar* en la interfaz principal, y debe ser modificada para que muestre sólo las variantes deseadas para el criterio de detención para el algoritmo de desrefinamiento.
- `GUIDialogNuevaMalla_glade`: Corresponde a la ventana que se despliega al seleccionar la opción *New* en la interfaz principal, y debe ser modificada para que muestre sólo las variantes deseadas para el tipo de figura.
- `GUIDialogRefinar_glade`: Corresponde a la ventana que se despliega al seleccionar la opción *Refinar* en la interfaz principal, y debe ser modificada para que muestre sólo las variantes deseadas para el algoritmo de refinamiento y el criterio de detención.

Capítulo 4
Diseño

En este capítulo se detallan las decisiones de diseño tomadas en el desarrollo del presente trabajo. En primer lugar, se describe el modelo de *Features* desarrollado a partir de las aplicaciones del dominio; en segundo lugar, se explica el diseño de la interfaz para la configuración y creación automática de productos, y por último se explica el diseño de clases correspondiente a la nueva versión de *Meshing Tool Generator*.

4.1. Modelo de *Features*

Como el presente trabajo se ciñe al paradigma de Ingeniería de Líneas de Productos de Software, el primer paso debe ser generar la plataforma de componentes reutilizables para las aplicaciones del dominio, incluyendo las similitudes y variabilidades. Para esto, fue necesario examinar las aplicaciones del dominio e identificar las similitudes que ambas comparten, y las variabilidades que poseen.

Las similitudes encontradas fueron las siguientes:

- **Malla**: Ésta es quizás la más evidente, dado que ambas aplicaciones trabajan con mallas geométricas.
- **Visualizar malla**: Ambas aplicaciones permiten al usuario visualizar la malla en todo momento.
- **Ver información malla**: Ambas aplicaciones despliegan un cuadro con información sobre la composición de la malla cuando el usuario lo requiera.
- **Cargar malla**: Ambas aplicaciones son capaces de cargar una malla a partir de distintos formatos de entrada.
- **Guardar malla**: Ambas aplicaciones son capaces de guardar una malla en distintos formatos de salida.
- **Procesar malla**: Ambas aplicaciones tienen al menos un algoritmo de procesamiento de la malla.

De la misma forma, las variabilidades encontradas fueron las siguientes, explicadas con mayor detalle en el capítulo anterior:

- **Algoritmo para Generar Malla Inicial**: Una malla puede ser generada a partir de una médula (curva generatriz de la malla, que modela el centro del tronco del árbol), o a partir de un cilindro.
- **Algoritmo de Refinamiento**: Es posible refinar la malla, es decir, aumentar su número de caras, disminuyendo el área de las caras originales o el largo de los arcos hasta que todas las caras tengan el tamaño deseado.
- **Criterio de Refinamiento**: El algoritmo de refinamiento se ejecuta en tanto el criterio escogido no se cumpla.
- **Algoritmo de Desrefinamiento**: Es posible invertir el procedimiento anterior, es decir, disminuir el número de caras de la malla, aumentando el área de las caras originales.
- **Criterio de Desrefinamiento**: El algoritmo de desrefinamiento se ejecuta en tanto el criterio escogido no se cumpla.
- **Algoritmo de Deformación**: Es posible deformar la malla, es decir, desplazar sus nodos; además, se puede escoger el algoritmo de tratamiento que se dará a las posibles colisiones que se producirán luego de dicho desplazamiento.
- **Algoritmo de Mejoramiento**: Es posible mejorar la calidad de la malla, la cual es medida utilizando el criterio de Delaunay.
- **Algoritmo de Distribución de Hormona**: Es posible distribuir la hormona de crecimiento sobre la malla.

Una vez identificadas las similitudes y variabilidades, es necesario definir restricciones adicionales entre ellas, que no pueden ser modeladas como características. Son las siguientes:

- Si el tipo de producto escogido es el Generador genérico de mallas, las variabilidades Generar malla y Distribución de hormona deben ser excluidas.
- Si el tipo de producto escogido es el Generador genérico de mallas, el formato Matlab para cargar mallas debe ser excluido, pues sólo está relacionado con el modelamiento de árboles.
- Si el tipo de producto escogido es el Generador genérico de mallas, debe incluirse al menos una de las siguientes variabilidades: Algoritmo de Refinamiento, Algoritmo de Desrefinamiento, Algoritmo de Deformación, Algoritmo de Mejoramiento.
- Si el tipo de producto escogido es el software TGS, debe incluirse al menos una de las siguientes variabilidades: Algoritmo de Refinamiento, Algoritmo de Desrefinamiento, Algoritmo de Deformación, Algoritmo de Mejoramiento, Algoritmo de Distribución de Hormona.
- Si se escoge un algoritmo de generación de mallas, debe escogerse al menos una variante.
- Si se escoge un algoritmo de refinamiento, debe escogerse al menos un criterio.
- Si se escoge un algoritmo de desrefinamiento, debe escogerse al menos un criterio.

Teniendo las similitudes, variabilidades y restricciones definidas, ya es posible generar el modelo de *Features* que describe el dominio. La primera sección del modelo se puede ver en la Figura 4.1.

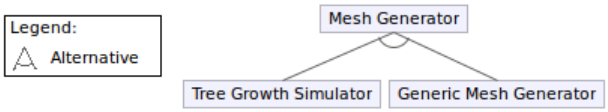

Figura 4.1: Primera sección del modelo de *Features*.

Aquí se muestra el primer nivel del modelo, que indica que la primera variabilidad identificada es el tipo de producto, la cual tiene dos variantes: Simulador de crecimiento de árboles y Generador genérico de mallas. Como ambas son características alternativas, una y sólo una de ellas puede ser escogida.

El resto del modelo está dividido en dos secciones, la primera correspondiente al software TGS y la segunda correspondiente al Generador genérico de mallas.

La sección que corresponde al software TGS se muestra en la Figura 4.2:

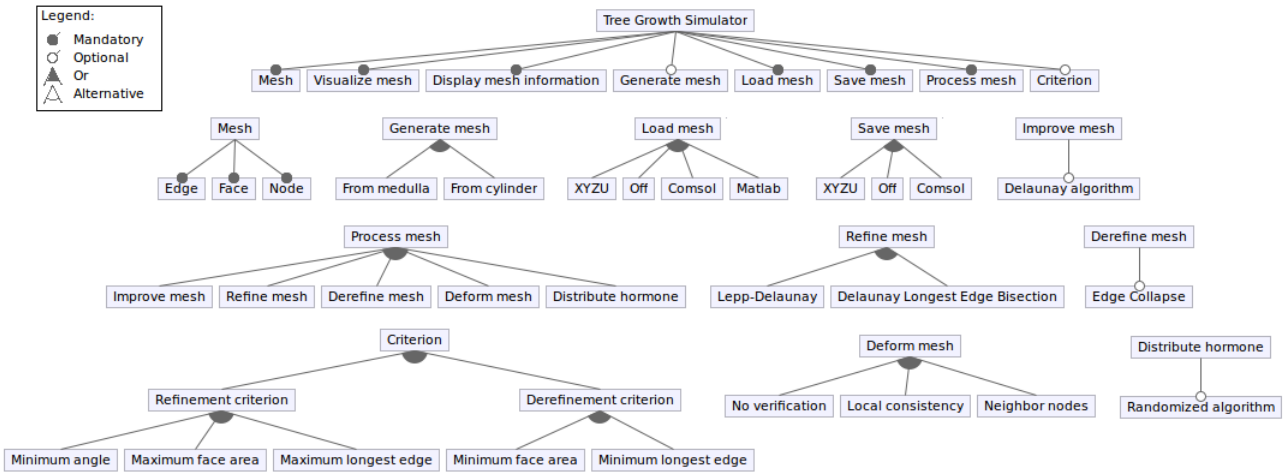

Figura 4.2: Sección del modelo correspondiente al software TGS.

Y la sección que corresponde al Generador genérico de mallas se muestra en la Figura 4.3:

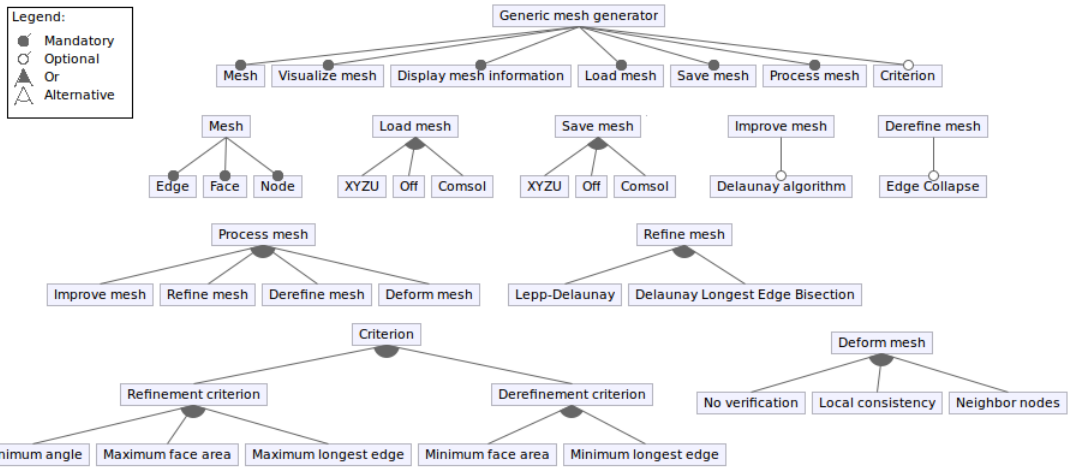

Figura 4.3: Sección del modelo correspondiente al Generador genérico de mallas.

Es necesario notar que ambos modelos de *features* incluyen las funcionalidades de Visualizar malla, Ver información malla, Cargar malla, Guardar malla, Refinar malla, Desrefinar malla, Mejorar malla y Deformar malla. Además, incluyen los criterios de detención para los algoritmos de refinamiento y desrefinamiento de mallas.

Junto con estas similitudes, se puede observar que sólo el modelo para el software TGS incluye el formato Matlab para cargar mallas, la funcionalidad de generar mallas a partir de una médula o un cilindro, y la funcionalidad de Distribuir hormona.

El modelo para el software TGS permite generar una variedad de diferentes productos que simulan el crecimiento de árboles. Para conocer el grado de variabilidad de este modelo y la cantidad de productos que permite generar, se utilizó la herramienta S.P.L.O.T. (*Software Product Lines Online Tools*) [1], la cual recibe un modelo de *features* junto a las restricciones que existen entre ellas y genera un análisis automatizado de dicho modelo. Los resultados para el modelo del software TGS se muestran en la Figura 4.4.

[1] http://www.splot-research.org/

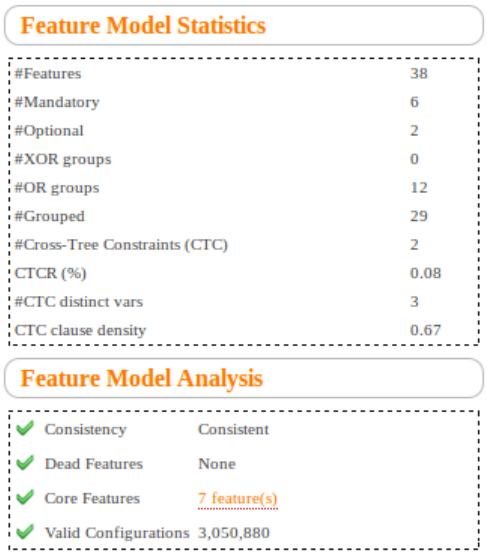

Figura 4.4: Resultados del análisis realizado por S.P.L.O.T. al modelo de *features* del software TGS.

De estos datos se destaca que existen 38 *features*, de las cuales 2 son opcionales y 6 son obligatorias, y están reunidas en 10 grupos. El número de restricciones lógicas es 2, y el número total de configuraciones es 3.050.880.

De la misma manera, el modelo para el Generador genérico de mallas permite generar una variedad de diferentes productos generadores de mallas geométricas. Los resultados para el modelo correspondiente a esta familia de aplicaciones se muestran en la Figura 4.5.

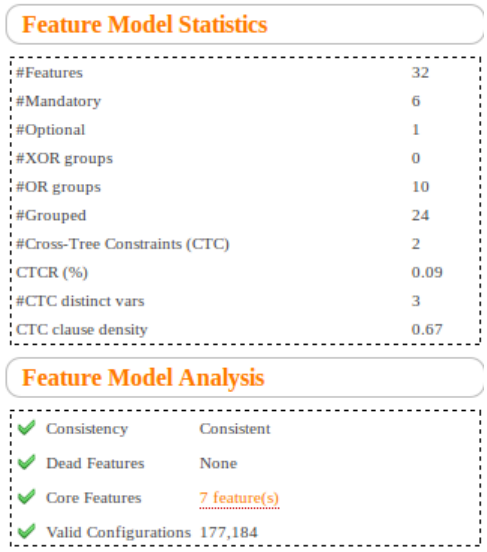

Figura 4.5: Resultados del análisis realizado por S.P.L.O.T. al modelo de *features* del Generador genérico de mallas.

De estos datos se desprende que existen 32 *features*, de las cuales una es opcional y 6 son obligatorias, y están reunidas en 10 grupos. El número de restricciones lógicas es 2, y el número total de configuraciones es 177.184.

4.2. Diseño de clases

Una vez realizado el modelo de *Features*, se procedió a realizar el diseño de clases para la nueva versión de *Meshing Tool Generator*. Para esto, fue necesario considerar que esta versión debe construir una configuración específica para generar uno de los productos del dominio, de las siguientes maneras:

- Generar automáticamente un archivo de configuración, que será procesado por *Meshing Tool Generator* en tiempo de ejecución e indicará qué funcionalidades deberán ser o no incluidas en la interfaz del producto.
- Generar automáticamente un archivo *header*, el cual será procesado en tiempo de compilación del producto, para evitar los posibles errores provocados por la ausencia de las clases excluidas.
- Generar automáticamente un archivo *makefile*, cuyo objetivo es indicar la compilación que deberá ser ejecutada para construir el producto, excluyendo las clases innecesarias.

El diagrama de clases de *Meshing Tool Generator* se muestra en la Figura 4.6.

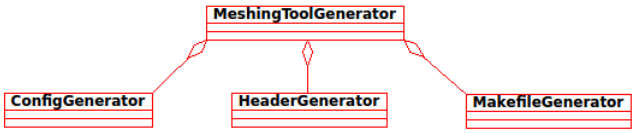

Figura 4.6: Diagrama de clases de la nueva versión de *Meshing Tool Generator*.

La clase correspondiente a la interfaz principal es `MeshingToolGenerator`, la cual tiene una relación de agregación con respecto a las otras tres clases: `ConfigGenerator`, encargada de generar el archivo de configuración; `MakefileGenerator`, encargada de generar el archivo *makefile*, y `HeaderGenerator`, encargada de generar el archivo *header*. Cada clase es explicada en detalle en el capítulo de Implementación.

4.3. Diseño preliminar de interfaces

Para diseñar la interfaz de la aplicación *Meshing Tool Generator*, se separaron las características del modelo ya expuesto en tres grupos:

- **Tipo de aplicación**
- **Manejo de archivos**: Generar malla, guardar malla, cargar malla.
- **Algoritmos de procesamiento de mallas**: Mejorar malla, refinar malla, desrefinar malla, distribuir hormona, deformar malla.

Cada uno de estos grupos corresponde a una sección de la interfaz, con lo cual se obtiene una configuración de la aplicación final en tres pasos. El diseño del primer paso se muestra en la Figura 4.7.

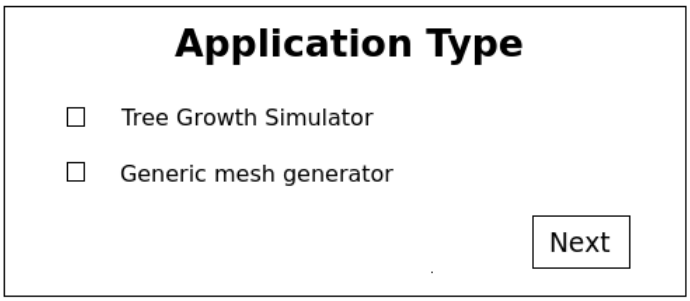

Figura 4.7: Diseño de interfaz del primer paso de *Meshing Tool Generator*.

La interfaz del segundo paso varía de acuerdo a lo seleccionado en el paso anterior. Si se escogió el software TGS, la interfaz corresponde a la mostrada en la Figura 4.8.

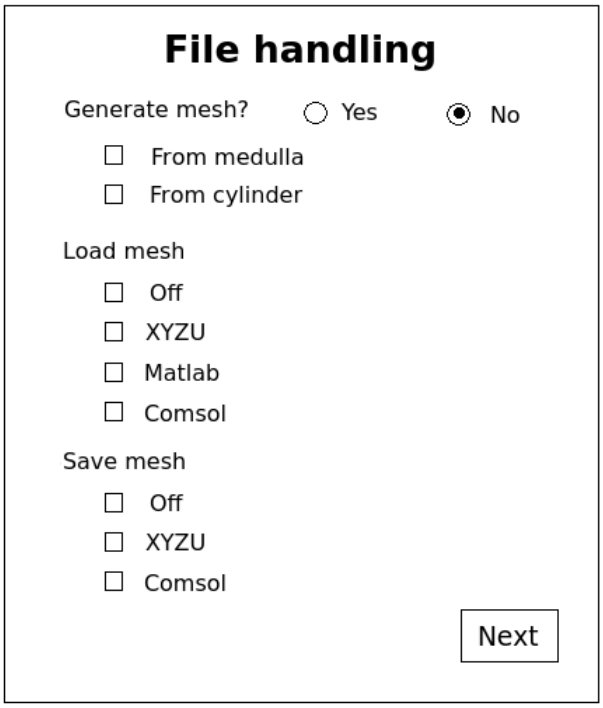

Figura 4.8: Diseño de interfaz del segundo paso de *Meshing Tool Generator*, al escoger el software TGS.

Si, por el contrario, la alternativa elegida fue el Generador genérico de mallas, en el segundo paso la opción de Generar malla y la alternativa Matlab (en Cargar malla) no son desplegadas, como se muestra en la Figura 4.9.

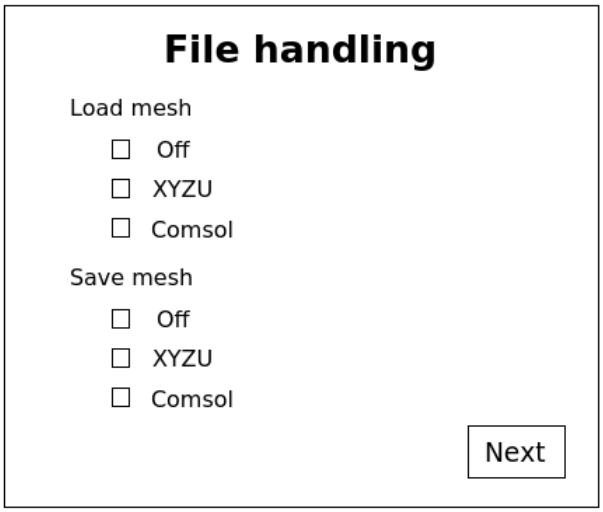

Figura 4.9: Diseño de interfaz del segundo paso de *Meshing Tool Generator*, al escoger el Generador genérico de mallas.

La interfaz del tercer paso también varía de acuerdo a lo seleccionado en el primero. Si se escogió el software TGS, la interfaz corresponde a la mostrada en la Figura 4.10.

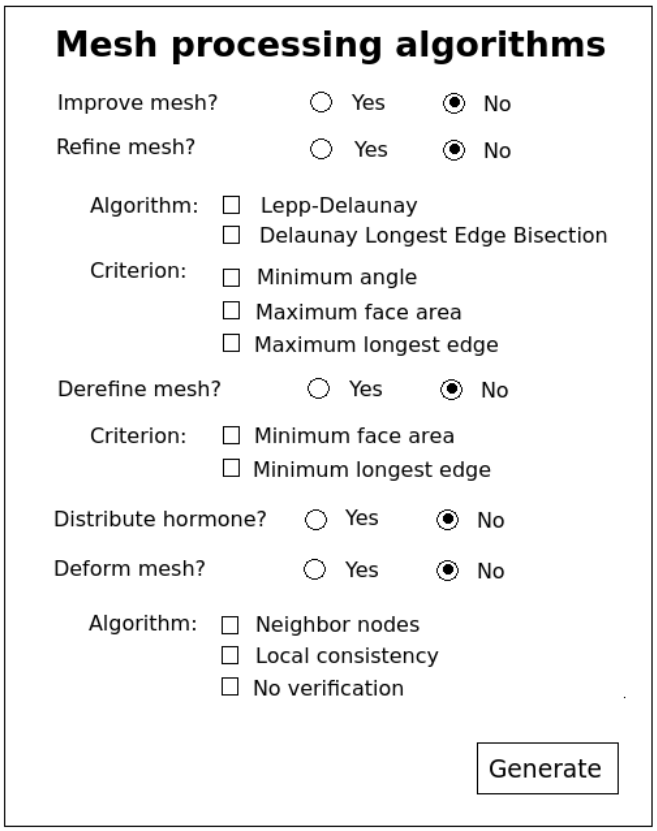

Figura 4.10: Diseño de interfaz del tercer paso de *Meshing Tool Generator*, al escoger el software TGS.

Si la alternativa elegida fue el Generador genérico de mallas, en el tercer paso la opción de Distribuir hormona no es desplegada, como se muestra en la Figura 4.11.

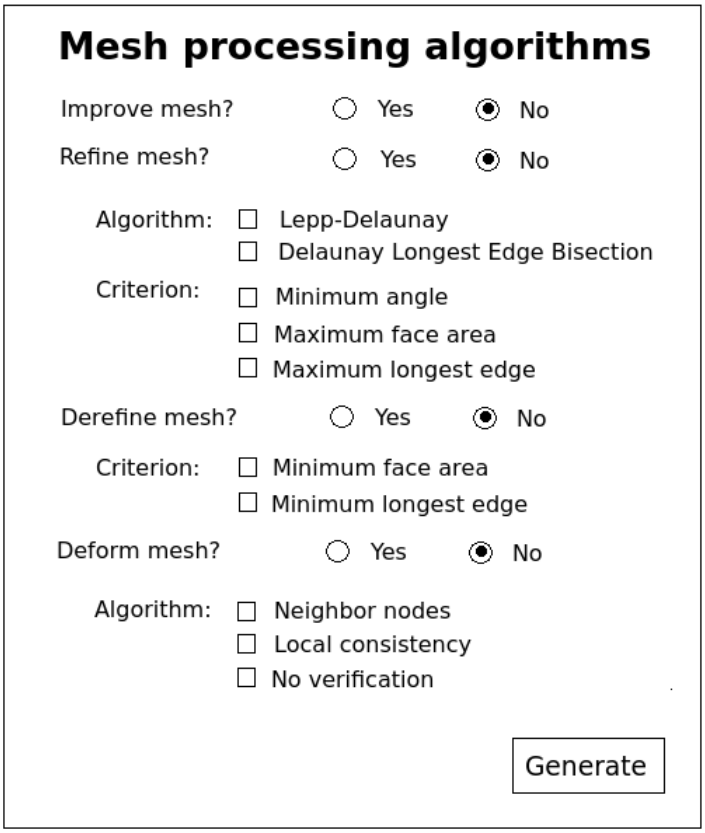

Figura 4.11: Diseño de interfaz del tercer paso de *Meshing Tool Generator*, al escoger el Generador genérico de mallas.

Cuando el usuario presiona el botón Generar, la aplicación finaliza y el archivo de configuración, el archivo *header* y el archivo *makefile* son construidos tomando en cuenta la información ingresada por el usuario; además, el archivo de configuración y el archivo *makefile* son copiados al directorio principal del producto base, y el archivo *header* es copiado al directorio que contiene el código fuente de dicho producto. Para compilar el producto final, el usuario debe ejecutar el archivo *makefile*.

Capítulo 5

Implementación de *Meshing Tool Generator*

En este capítulo se detallan los pasos seguidos para implementar las clases descritas en el capítulo anterior, y se describe la intervención realizada al código base.

5.1. Interfaz de usuario: clase *MeshingToolGenerator*

En primer lugar, se realizó el desarrollo de las interfaces cuyo diseño ya fue expuesto en el capítulo anterior. Para esto, se utilizó el programa diseñador de interfaces Glade [1], y luego se implementó la funcionalidad correspondiente en la clase `MeshingToolGenerator`, de modo de enlazar los tres pasos tal como fue descrito en el capítulo de Diseño. El código correspondiente a esta clase se detalla en el Anexo A.1, y su tamaño es de 348 líneas.

Para asegurar una correcta configuración del producto final, se implementaron las siguientes restricciones en la interfaz del segundo paso:

- Si el usuario escogió el software TGS en el primer paso, en el segundo debe escoger al menos un formato de entrada y un formato de salida para poder avanzar al siguiente paso. Además, si desea incluir la funcionalidad de Generar malla, debe seleccionar al menos un tipo de figura para poder avanzar.
- Si el usuario escogió el generador genérico en el primer paso, en el segundo debe escoger al menos un formato de entrada y un formato de salida para poder avanzar al siguiente paso.

De la misma manera, las siguientes restricciones fueron implementadas en la interfaz del tercer paso:

- Para poder generar la configuración, el usuario debe seleccionar al menos un algoritmo de procesamiento de malla para generar la configuración. Además, si desea incluir la funcionalidad de Refinar malla, debe seleccionar al menos un algoritmo y un criterio; si desea incluir la funcionalidad de Desrefinar malla, debe seleccionar al menos un criterio.

[1] http://glade.gnome.org/

- Si el usuario escogió el software TGS en el primer paso y además desea incluir la funcionalidad de Deformar malla, debe seleccionar al menos un algoritmo de tratamiento de posibles colisiones.

5.2. Generación del archivo de configuración: clase *ConfigGenerator*

Cuando el usuario presiona el botón `Apply` en la interfaz del tercer paso, la clase `MeshingToolGenerator` invoca a la clase `ConfigGenerator`; esta clase, tomando en cuenta las elecciones realizadas por el usuario, se encarga de construir un archivo de configuración utilizando la biblioteca Libconfig [2]. Esto constituye un cambio con respecto a la versión anterior de *Meshing Tool Generator*, pues ésta genera un archivo de texto plano, mientras que el formato provisto por Libconfig es más compacto y liviano que un archivo de texto plano o XML. El código correspondiente a esta clase se detalla en el Anexo A.2, y su tamaño es de 93 líneas.

La clase `ConfigGenerator` posee los siguientes métodos:

- `GenerarConfigStep1`: Este método recibe una variable booleana, que indica si el usuario seleccionó la opción del software TGS en el primer paso, e ingresa su valor, asociado a la llave `TreeGrowthSimulator`, al archivo de configuración.
- `GenerarConfigStep2`: Este método recibe tres arreglos asociativos, que almacenan los valores ingresados por el usuario para los formatos de carga, almacenamiento y generación de mallas, respectivamente; luego, ingresa estos valores, asociados a las llaves correspondientes, al archivo de configuración.
- `GenerarConfigStep3`: Este método recibe dos variables booleanas, que indican si el usuario seleccionó las opciones de Mejorar malla y Distribuir hormona, respectivamente, y cuatro arreglos asociativos, que almacenan los valores ingresados por el usuario para los algoritmos y criterios de refinamiento, criterios de desrefinamiento y algoritmos de verificación de inconsistencias, respectivamente; luego, ingresa estos valores, asociados a las llaves correspondientes, al archivo de configuración.
- `CopyFile`: Este método copia el archivo generado al directorio que contiene el código del software base.

Un ejemplo de archivo de configuración se puede ver en la Figura 5.1. En este caso, el usuario escogió el software TGS en el primer paso, todos los formatos para cargar, guardar y generar mallas en el segundo, y la funcionalidad de mejorar una malla en el tercero.

[2] http://www.hyperrealm.com/libconfig/

```
TreeGrowthSimulator = true;
LoadMesh :
{
  Comsol = true;
  Matlab = true;
  Off = true;
  Xyzu = true;
};
SaveMesh :
{
  Comsol = true;
  Off = true;
  Xyzu = true;
};
GenerateMesh :
{
  Cylinder = true;
  Medulla = true;
};
ImproveMesh = true;
RefineMesh = false;
DerefineMesh = false;
DistributeHormone = false;
DeformMesh = false;
```

Figura 5.1: Ejemplo de archivo de configuración generado por la clase *ConfigGenerator*.

El archivo generado es simple de procesar, pues las funcionalidades y demás características que deben ser incluidas tienen el valor `true`, mientras que todo lo demás tiene el valor `false`.

5.3. Generación del archivo *header*: clase *HeaderGenerator*

Como ya se mencionó en el capítulo de Análisis, un problema existente en el software TGS es el alto acoplamiento de sus clases, lo cual hace necesaria la inclusión de la mayoría de las clases al momento de compilar la aplicación, dificultando de esta manera la configuración en tiempo de compilación.

Con el objetivo de solucionar este problema se creó la clase `HeaderGenerator`, la cual posee los siguientes métodos:

- `Generate`: Este método procesa el archivo de configuración generado por la clase `ConfigGenerator` y construye un archivo *header*, el cual define una macro por cada funcionalidad o característica cuyo valor es `true`, a través de la directiva `#define`. De esta manera, la compilación del producto final será condicionada por los valores de las macros definidas, dejando fuera las porciones de código que correspondan a funcionalidades o características no deseadas.
- `CopyFile`: Este método copia el archivo *header* generado al directorio que contiene el código del software base.

El archivo *header* correspondiente al ejemplo de la sección anterior se muestra en la Figura 5.2. En él, se definen las macros correspondientes a las funcionalidades incluidas en el archivo de configuración, las cuales serán leídas por el preprocesador para que excluya de la compilación las porciones de código correspondientes a las demás funcionalidades.

```
#ifndef CONFIG_H_INCLUDED
#define CONFIG_H_INCLUDED

#define TREE_GROWTH
#define LOAD_COMSOL
#define LOAD_MATLAB
#define LOAD_OFF
#define LOAD_XYZU
#define SAVE_COMSOL
#define SAVE_OFF
#define SAVE_XYZU
#define GENERATE_CYLINDER
#define GENERATE_MEDULLA
#define IMPROVE

#endif // CONFIG_H_INCLUDED
```

Figura 5.2: Ejemplo de archivo *header* generado por la clase *HeaderGenerator*.

El código correspondiente a la clase `HeaderGenerator` se detalla en el Anexo A.3, y su tamaño es de 112 líneas.

5.4. Intervención en el código del software TGS

En primer lugar, se cambió el nombre del directorio principal, para que se llamara `MeshGenerator` en vez de `TreeGrowthSimulator`. Luego, fue necesario adaptar su código para que, en tiempo de ejecución, procese el archivo de configuración generado por la clase `ConfigGenerator` y fuera capaz de incluir las funcionalidades y variantes cuyo valor sea `true` y descartar las demás. Para esto, se creó una clase llamada `ConfigReader` (cuyo código se muestra en el Anexo A.5, y cuyo tamaño es de 53 líneas), la cual posee métodos que retornan los valores presentes en el archivo de configuración para cada una de las funcionalidades posibles. Teniendo esta nueva clase que permite leer el archivo de configuración en tiempo de ejecución, se modificaron las siguientes clases del software TGS:

- `GUIDialogNuevaMalla_glade`: Corresponde a la interfaz de la ventana de diálogo que se despliega cuando el usuario elige generar una malla nueva, la cual posee una lista desplegable que permite al usuario elegir el tipo de figura, entre médula y cilindro; esta lista fue adaptada para que muestre sólo las alternativas presentes en el archivo de configuración.

- `GUIDialogRefinar_glade`: Corresponde a la interfaz de la ventana de diálogo que se despliega cuando el usuario elige refinar la malla, la cual permite al usuario elegir el algoritmo de refinamiento, junto con su criterio de detención. Esta interfaz fue adaptada para que muestre sólo las alternativas presentes en el archivo de configuración.

- `GUIDialogDesrefinar_glade`: Corresponde a la interfaz de la ventana de diálogo que se despliega cuando el usuario elige desrefinar la malla, la cual permite al usuario elegir el criterio de detención del algoritmo de desrefinamiento. Esta interfaz fue adaptada para que muestre sólo las alternativas presentes en el archivo de configuración.
- `GUIDialogCambios_glade`: Corresponde a la interfaz de la ventana de diálogo que se despliega cuando el usuario elige deformar la malla, la cual permite al usuario elegir el algoritmo de tratamiento de posibles colisiones. Esta interfaz fue adaptada para que muestre sólo las alternativas presentes en el archivo de configuración.
- `GUIVentanaPrincipal_glade`: Corresponde a la interfaz principal del software TGS, la cual posee botones que permiten al usuario generar, cargar, guardar, deformar, refinar, desrefinar y mejorar una malla, además de aplicar el algoritmo de distribución de hormona sobre ella. Esta interfaz fue adaptada para que muestre sólo los botones que correspondan a las funcionalidades presentes en el archivo de configuración.

Para permitir la eliminación automática de las clases innecesarias en la compilación, fue necesario realizar una separación de las clases del software TGS, distinguiendo las que se necesitan siempre de las que cada variabilidad utiliza. Luego de esto, se adaptaron las siguientes clases del software TGS, para que incluyeran el archivo *header* generado (a través de la directiva `#include`), además de las clases correspondientes indicadas por las macros definidas en este archivo (a través de la directiva `#ifdef`):

- `Generar`: Dependiendo de las macros definidas en el archivo *header*, se incluyen las referencias a las clases `GeneraFromOFF`, `GeneraFromMatlab`, `GeneraFromArchivosXYZU`, `GeneraFromComsol_1` (variantes del formato de carga de mallas), y las referencias a las clases `GeneraCilindro` y `GeneraFromMedula` (variantes del formato de generación de mallas nuevas).
- `Guardar`: Dependiendo de las macros definidas en el archivo *header*, se incluyen las referencias a las clases `AlmacenaToOff`, `AlmacenaToXYZU` y `AlmacenaToComsol_1`, correspondientes a las variantes del formato de almacenamiento de mallas.
- `Refinar`: Dependiendo de las macros definidas en el archivo *header*, se incluyen las referencias a las clases `LeppDelaunay` y `DelaunayLongestEdgeBisection` (variantes del algoritmo de refinamiento), y las referencias a las clases `AnguloMinimo`, `AreaCara` y `LargoArcoMaximo` (variantes del criterio de detención).
- `Desrefinar`: Dependiendo de las macros definidas en el archivo *header*, se incluyen las referencias a las clases `AreaCara` y `LargoArcoMinimo`, correspondientes a las variantes del criterio de detención del algoritmo de desrefinamiento.
- `Deformar`: Dependiendo de las macros definidas en el archivo *header*, se incluyen las referencias a las clases `NoVerificacion`, `VerificacionLocal` y `VerificacionNodosVecinos`, correspondientes a las variantes del algoritmo de tratamiento de posibles colisiones.
- `GUIVentanaPrincipal`: Dependiendo de las macros definidas en el archivo *header*, se incluyen las referencias a las clases `GUIDialogNuevaMalla`, `GUIDialogCambios`, `GUIDialogAbrirMatlab`, `GUIDialogDistribuirHormona`, `GUIDialogRefinar`, `GUIDialogDesrefinar` y `Mejorar`.

A modo de ejemplo, se muestra la adaptación realizada a la clase `Guardar` en la Figura 5.3.

```
#include "config.h"
#include "guardar.h"

#ifdef SAVE_OFF
#include "almacenatooff.h"
#endif

#ifdef SAVE_XYZU
#include "almacenatoxyzu.h"
#endif

#ifdef SAVE_COMSOL
#include "almacenatocomsol_1.h"
#endif

Guardar::Guardar(Malla *m, int formato, string filename) : Comando(m) {
    algoritmo=NULL;
        this->filename=filename;

        if (formato==0) {
#ifdef SAVE_OFF
                algoritmo = new AlmacenaToOff();
#endif
        }
        else if (formato==2) {
#ifdef SAVE_XYZU
                algoritmo = new AlmacenaToXYZU();
#endif
        }
        else if (formato==3) {
#ifdef SAVE_COMSOL
                algoritmo = new AlmacenaToComsol_1();
#endif
        }

        assert(algoritmo);
        }
```

Figura 5.3: Adaptación realizada a la clase *Guardar*.

En primer lugar, se agregaron las directivas `#ifdef` al inicio del archivo, que indican que los archivos *header* correspondientes a los algoritmos de almacenamiento en formatos Off, XYZU y Comsol deben incluirse sólo si están definidas las macros `SAVE_OFF`, `SAVE_XYZU` y `SAVE_COMSOL`, respectivamente. Luego, las mismas directivas fueron agregadas antes de cada llamada al constructor de una variante, indicando que la compilación del producto tomará dicha llamada en cuenta solo si la macro correspondiente está definida.

5.5. Generación del archivo *makefile*: clase *Makefile-Generator*

Teniendo ya las configuraciones en tiempo de ejecución y compilación, falta solamente generar automáticamente el archivo *makefile*, el cual debe ser ejecutado para compilar la aplicación configurada. Para esto fue necesario utilizar la separación de clases mencionada en la sección anterior, incluyendo en el *makefile* las clases que se necesitan siempre y procesando el archivo de configuración para incluir las clases correspondientes a las funcionalidades y demás

variantes escogidas por el usuario. La clase encargada de esta tarea es `MakefileGenerator`; el código correspondiente a ella se muestra en el Anexo A.4, y su tamaño es de 162 líneas.

El archivo *makefile* correspondiente al ejemplo mencionado en las dos secciones anteriores se muestra en la Figura 5.4.

```
EXEC = treegrowthsimulator
CC = g++
LFLAGS = -O2 -lgsl -lgslcblas -lconfig++
PKG_CONFIG = `pkg-config gtkglextmm-1.2 --cflags --libs sigc++-2.0`

SOURCES = ./src/arco.cpp ./src/arcos.cpp ./src/cara.cpp ./src/caras.cpp ./src/comando.cpp
./src/configreader.cpp ./src/generar.cpp ./src/guardar.cpp ./src/GUIDialogInformacion.cc
./src/GUIDialogInformacion_glade.cc ./src/GUIVentanaPrincipal.cc
./src/GUIVentanaPrincipal_glade.cc ./src/informacionmalla.cpp ./src/informar.cpp
./src/malla.cpp ./src/nodo.cpp ./src/nodos.cpp ./src/polinomio.cpp ./src/punto.cpp
./src/segmenttriangleintersection.cpp ./src/segtriint.cpp ./src/SimpleGLScene.cc
./src/treegrowthsimulator.cc ./src/vect.cpp ./src/generafromcomsol_1.cpp
./src/generafrommatlab.cpp ./src/GUIDialogAbrirMatlab.cc ./src/GUIDialogAbrirMatlab_glade.cc
./src/generafromoff.cpp ./src/generafromarchivosxyzu.cpp ./src/almacenatocomsol_1.cpp
./src/almacenatooff.cpp ./src/almacenatoxyzu.cpp ./src/GUIDialogNuevaMalla.cc
./src/GUIDialogNuevaMalla_glade.cc ./src/generacilindro.cpp ./src/generafrommedula.cpp
./src/mejdelaunay.cpp ./src/mejorar.cpp

OUTPUT = salida

default:
        $(CC) $(SOURCES) $(LFLAGS) $(PKG_CONFIG) -o $(EXEC) 2> $(OUTPUT)
clean:
        rm -rf $(EXEC)
```

Figura 5.4: Ejemplo de archivo *makefile* generado por *Meshing Tool Generator*.

Capítulo 6
Ejemplos y evaluación de resultados

En el presente capítulo, se ejecuta el software *Meshing Tool Generator* para construir automáticamente dos productos generadores de mallas: un generador genérico y un software TGS. Se muestran los archivos de configuración generados, y además las interfaces de los productos construidos.

6.1. Configuración de un generador genérico de mallas

En este ejemplo se construirá un Generador genérico de mallas que incluya las siguientes funcionalidades y variantes:

- **Cargar malla**: Formatos *Off* y *Xyzu*.
- **Guardar malla**: Formatos *Off* y *Xyzu*.
- **Mejorar malla**
- **Refinar malla**: Todos los algoritmos y criterios.

Después de ejecutar el software *Meshing Tool Generator*, seleccionando las alternativas mencionadas en cada paso, se generó automáticamente el archivo de configuración que se muestra en la Figura 6.1, junto con el archivo *header* que se muestra en la Figura 6.2 y el archivo *makefile* que se muestra en la Figura 6.3.

```
TreeGrowthSimulator = false;
LoadMesh :
{
  Comsol = false;
  Matlab = false;
  Off = true;
  Xyzu = true;
};
SaveMesh :
{
  Comsol = false;
  Off = true;
  Xyzu = true;
};
GenerateMesh = false;
ImproveMesh = true;
RefineMesh :
{
  Algorithms :
  {
    LeppDelaunay = true;
    LongestEdge = true;
  };
  Criteria :
  {
    MaxArea = true;
    MaxLength = true;
    MinAngle = true;
  };
};
DerefineMesh = false;
DistributeHormone = false;
DeformMesh = false;
```

Figura 6.1: Archivo de configuración correspondiente al ejemplo de generador genérico.

```
#ifndef CONFIG_H_INCLUDED
#define CONFIG_H_INCLUDED

#define LOAD_OFF
#define LOAD_XYZU
#define SAVE_OFF
#define SAVE_XYZU
#define IMPROVE
#define REFINE_ALG_LEPPDELAUNAY
#define REFINE_ALG_LONGESTEDGE
#define REFINE_CRITERION_MAXAREA
#define REFINE_CRITERION_MAXLENGTH
#define REFINE_CRITERION_MINANGLE

#endif // CONFIG_H_INCLUDED
```

Figura 6.2: Archivo *header* correspondiente al ejemplo de generador genérico.

```
EXEC = meshgenerator
CC = g++
LFLAGS = -O2 -lgsl -lgslcblas -lconfig++
PKG_CONFIG = `pkg-config gtkglextmm-1.2 --cflags --libs sigc++-2.0`

SOURCES = ./src/arco.cpp ./src/arcos.cpp ./src/cara.cpp ./src/caras.cpp
./src/comando.cpp ./src/configreader.cpp ./src/generar.cpp ./src/guardar.cpp
./src/GUIDialogInformacion.cc ./src/GUIDialogInformacion_glade.cc
./src/GUIVentanaPrincipal.cc ./src/GUIVentanaPrincipal_glade.cc
./src/informacionmalla.cpp ./src/informar.cpp ./src/malla.cpp ./src/nodo.cpp
./src/nodos.cpp ./src/polinomio.cpp ./src/punto.cpp
./src/segmenttriangleintersection.cpp ./src/segtriint.cpp
./src/SimpleGLScene.cc ./src/treegrowthsimulator.cc ./src/vect.cpp
./src/generafromoff.cpp ./src/generafromarchivosxyzu.cpp
./src/almacenatooff.cpp ./src/almacenatoxyzu.cpp ./src/mejdelaunay.cpp
./src/mejorar.cpp ./src/refinar.cpp ./src/GUIDialogRefinar.cc
./src/GUIDialogRefinar_glade.cc ./src/lepp.cpp ./src/leppdelaunay.cpp
./src/delaunaylongestedgebisection.cpp ./src/angulominimo.cpp
./src/areacara.cpp ./src/largoarcomaximo.cpp

OUTPUT = salida

default:
        $(CC) $(SOURCES) $(LFLAGS) $(PKG_CONFIG) -o $(EXEC) 2> $(OUTPUT)
clean:
        rm -rf $(EXEC)
```

Figura 6.3: Archivo *makefile* correspondiente al ejemplo de generador genérico.

Una vez efectuada la compilación de forma manual a través de este archivo *makefile*, se obtiene el archivo ejecutable `meshgenerator`, el cual corresponde al producto generado. Al ser ejecutado, despliega la interfaz mostrada en la Figura 6.4.

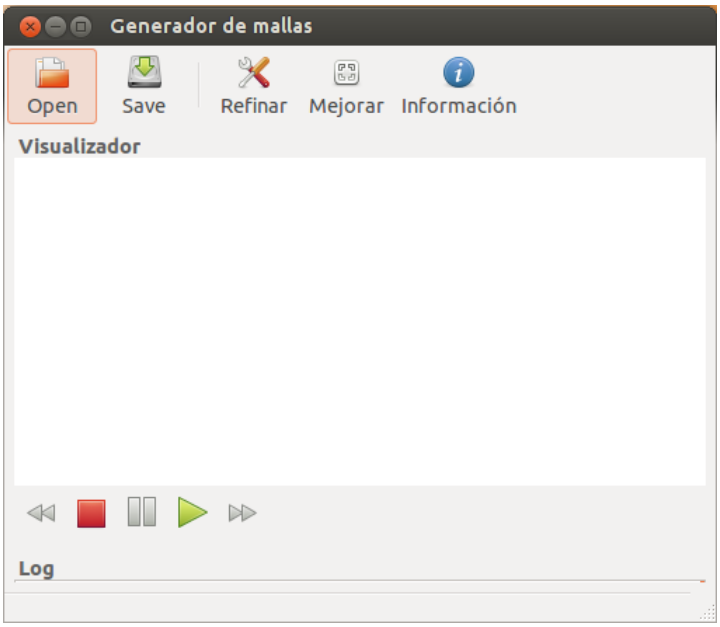

Figura 6.4: Inicio del generador genérico de mallas.

Aquí se ve que sólo los botones correspondientes a las funcionalidades escogidas son desplegados en la interfaz principal. Al escoger cargar una malla, se despliega la ventana mostrada en la Figura 6.5.

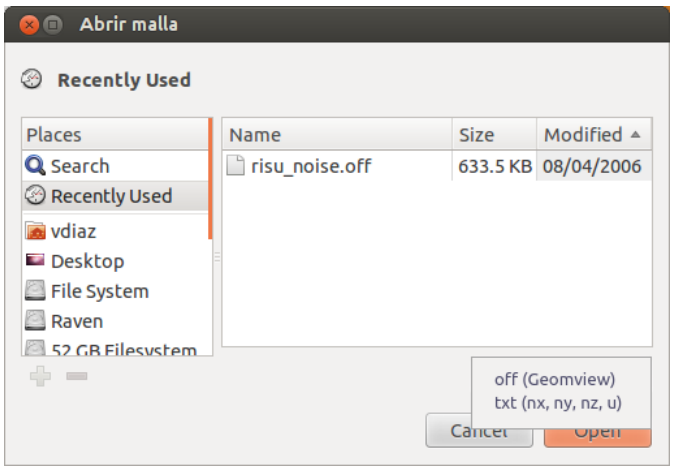

Figura 6.5: Ventana de diálogo para cargar mallas.

En esta figura se muestra que, efectivamente, sólo se pueden cargar mallas de formato *Off* y *Xyzu*. Al escoger cargar una malla de formato *Off* (en este caso, `moai_noise.off`), dicha malla es desplegada en el Visualizador, como se muestra en la Figura 6.6.

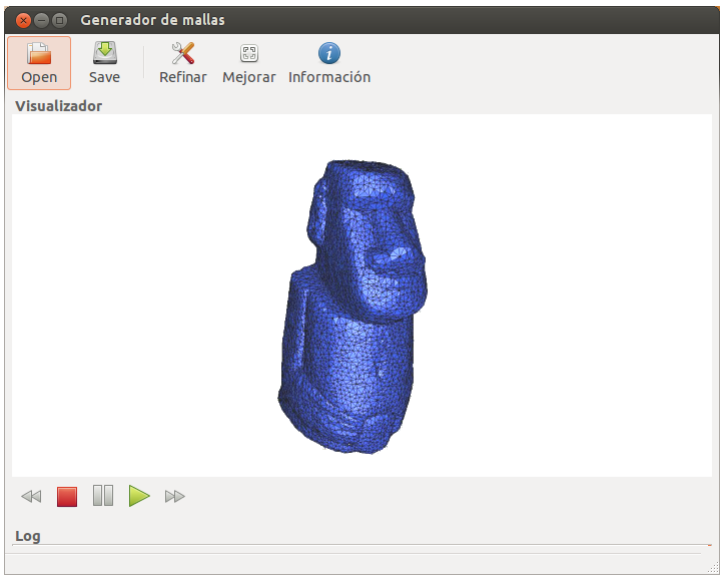

Figura 6.6: Malla *moai.off* desplegada en el Visualizador.

Al escoger guardar esta malla, se abre la ventana mostrada en la Figura 6.7, donde se ve que, tal como se eligió en la aplicación *Meshing Tool Generator*, sólo se pueden guardar mallas a los formatos *Off* y *Xyzu*.

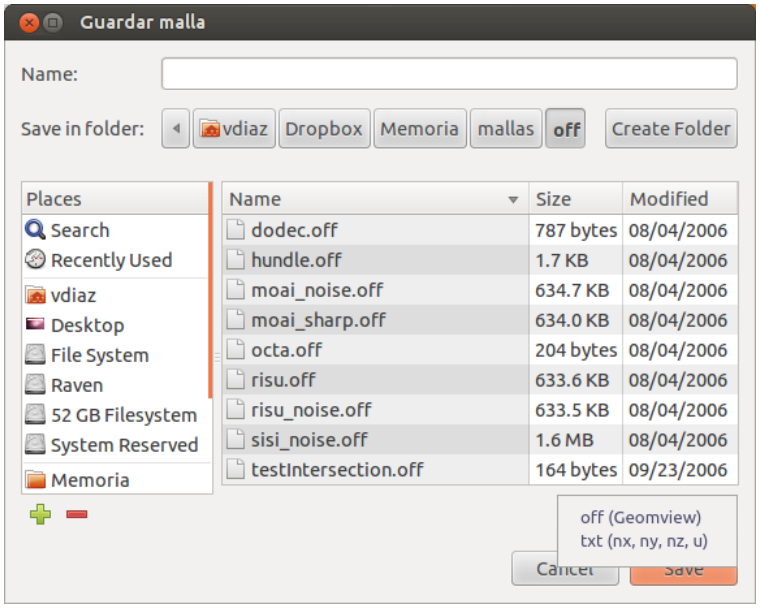

Figura 6.7: Ventana de diálogo para guardar mallas.

Finalmente, al escoger refinar la malla, se despliega la ventana mostrada en la Figura 6.8, donde se muestra que aparecen todos los algoritmos y criterios disponibles, como se seleccionó en la aplicación *Meshing Tool Generator*.

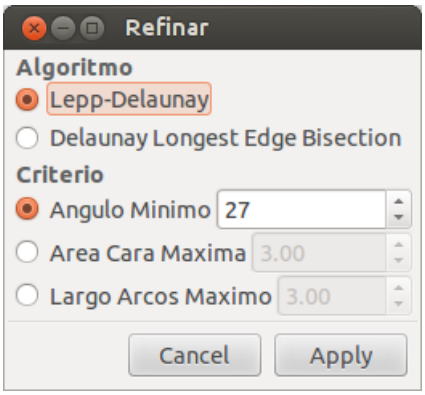

Figura 6.8: Ventana de diálogo para refinar mallas.

6.2. Configuración de un software TGS

En este ejemplo se construirá un software TGS que incluya las siguientes funcionalidades y variantes:

- **Cargar malla**: Formatos *Xyzu* y *Matlab*.
- **Guardar malla**: Formatos *Xyzu* y *Comsol*.
- **Generar malla**: A partir de un cilindro.
- **Refinar malla**: Algoritmo de Lepp-Delaunay y criterio de ángulo mínimo.
- **Distribuir hormona**
- **Deformar malla**: Algoritmos de Verificación de Nodos vecinos y Consistencia local.

Después de ejecutar el software *Meshing Tool Generator*, seleccionando las alternativas mencionadas en cada paso, se generó el archivo de configuración que se muestra en la Figura 6.9, junto con el archivo *header* que se muestra en la Figura 6.10 y el archivo *makefile* que se muestra en la Figura 6.11.

```
TreeGrowthSimulator = true;
LoadMesh :
{
  Comsol = false;
  Matlab = true;
  Off = false;
  Xyzu = true;
};
SaveMesh :
{
  Comsol = true;
  Off = false;
  Xyzu = true;
};
GenerateMesh :
{
  Cylinder = true;
  Medulla = false;
};
ImproveMesh = false;
RefineMesh :
{
  Algorithms :
  {
    LeppDelaunay = true;
    LongestEdge = false;
  };
  Criteria :
  {
    MaxArea = false;
    MaxLength = false;
    MinAngle = true;
  };
};
DerefineMesh = false;
DistributeHormone = true;
DeformMesh :
{
  LocalConsistency = true;
  NeighborNodes = true;
  NoVerification = false;
};
```

Figura 6.9: Archivo de configuración correspondiente al ejemplo de software TGS.

```
#ifndef CONFIG_H_INCLUDED
#define CONFIG_H_INCLUDED

#define TREE_GROWTH
#define LOAD_MATLAB
#define LOAD_XYZU
#define SAVE_COMSOL
#define SAVE_XYZU
#define GENERATE_CYLINDER
#define REFINE_ALG_LEPPDELAUNAY
#define REFINE_CRITERION_MINANGLE
#define DISTRIB_HORMONE
#define DEFORM_ALG_LOCALCONSISTENCY
#define DEFORM_ALG_NEIGHBORNODES

#endif // CONFIG_H_INCLUDED
```

Figura 6.10: Archivo *header* correspondiente al ejemplo de software TGS.

```
EXEC = treegrowthsimulator
CC = g++
LFLAGS = -O2 -lgsl -lgslcblas -lconfig++
PKG_CONFIG = `pkg-config gtkglextmm-1.2 --cflags --libs sigc++-2.0`

SOURCES = ./src/arco.cpp ./src/arcos.cpp ./src/cara.cpp ./src/caras.cpp
./src/comando.cpp ./src/configreader.cpp ./src/generar.cpp ./src/guardar.cpp
./src/GUIDialogInformacion.cc ./src/GUIDialogInformacion_glade.cc
./src/GUIVentanaPrincipal.cc ./src/GUIVentanaPrincipal_glade.cc
./src/informacionmalla.cpp ./src/informar.cpp ./src/malla.cpp ./src/nodo.cpp
./src/nodos.cpp ./src/polinomio.cpp ./src/punto.cpp
./src/segmenttriangleintersection.cpp ./src/segtriint.cpp
./src/SimpleGLScene.cc ./src/treegrowthsimulator.cc ./src/vect.cpp
./src/generafrommatlab.cpp ./src/GUIDialogAbrirMatlab.cc
./src/GUIDialogAbrirMatlab_glade.cc ./src/generafromarchivosxyzu.cpp
./src/almacenatocomsol_1.cpp ./src/almacenatoxyzu.cpp
./src/GUIDialogNuevaMalla.cc ./src/GUIDialogNuevaMalla_glade.cc
./src/generacilindro.cpp ./src/refinar.cpp ./src/GUIDialogRefinar.cc
./src/GUIDialogRefinar_glade.cc ./src/lepp.cpp ./src/leppdelaunay.cpp
./src/angulominimo.cpp ./src/distribuirhormona.cpp
./src/GUIDialogDistribuirHormona.cc ./src/GUIDialogDistribuirHormona_glade.cc
./src/deformar.cpp ./src/GUIDialogCambios.cc ./src/GUIDialogCambios_glade.cc
./src/verificacionlocal.cpp ./src/verificacionnodosvecinos.cpp

OUTPUT = salida

default:
        $(CC) $(SOURCES) $(LFLAGS) $(PKG_CONFIG) -o $(EXEC) 2> $(OUTPUT)
clean:
        rm -rf $(EXEC)
```

Figura 6.11: Archivo *makefile* correspondiente al ejemplo de software TGS.

Una vez efectuada la compilación de forma manual a través de este archivo *makefile*, se obtiene el archivo ejecutable `treegrowthsimulator`, el cual corresponde al producto generado. Al ser ejecutado, despliega la interfaz mostrada en la Figura 6.12.

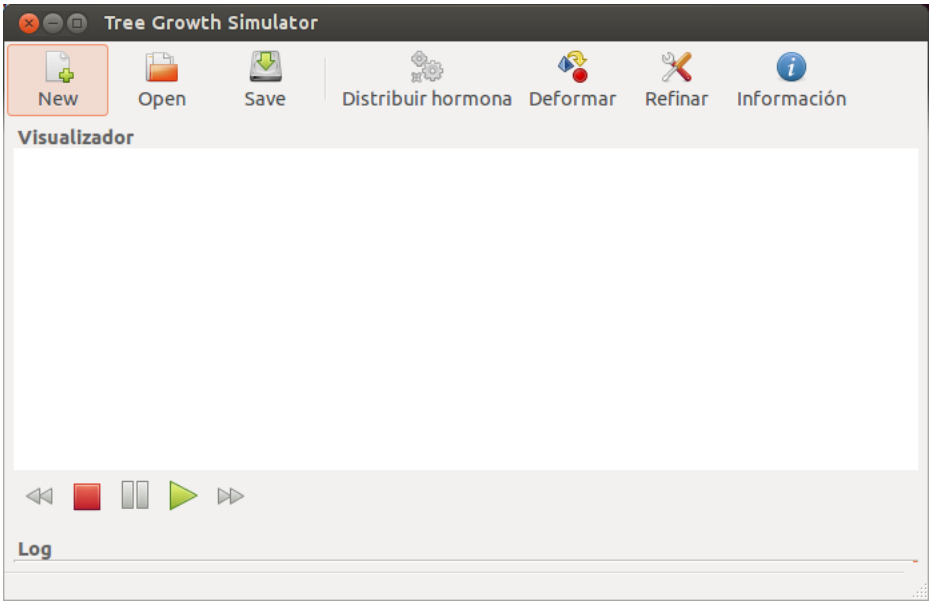

Figura 6.12: Inicio del software TGS.

Aquí se ve que sólo los botones correspondientes a las funcionalidades escogidas son desplegados en la interfaz principal. Al escoger cargar una malla, se despliega la ventana mostrada en la Figura 6.13.

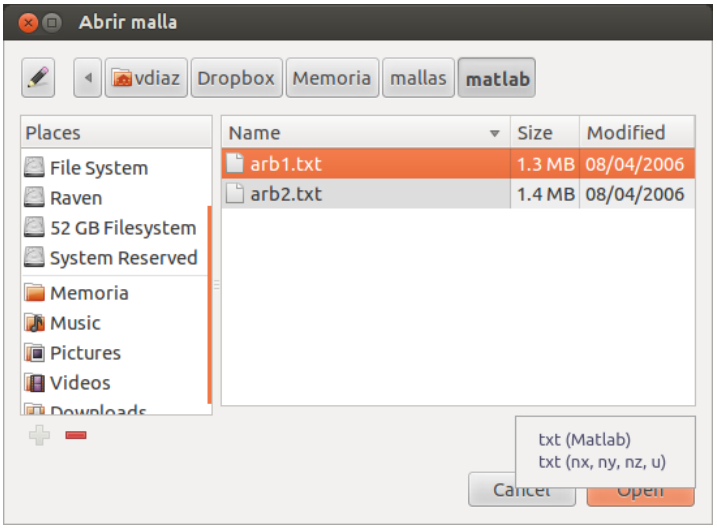

Figura 6.13: Ventana de diálogo para cargar mallas.

En esta figura se muestra que, efectivamente, sólo se pueden cargar mallas de formato *Xyzu* y *Matlab*. Al escoger cargar una malla de formato *Xyzu* (en este caso, `data`), dicha malla es desplegada en el Visualizador, como se muestra en la Figura 6.14.

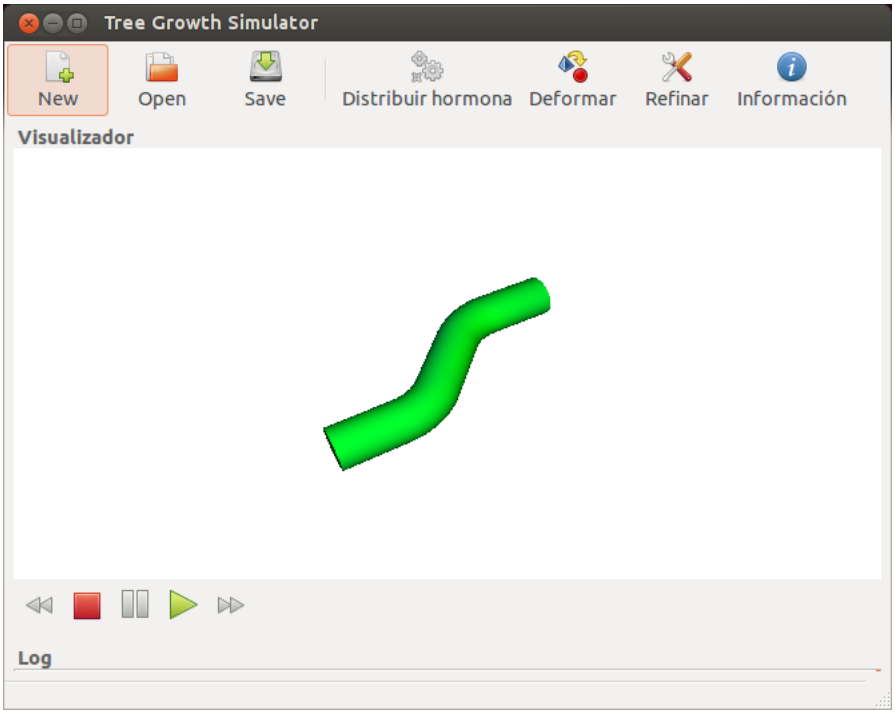

Figura 6.14: Malla *data* desplegada en el Visualizador.

Al escoger guardar esta malla, se abre la ventana mostrada en la Figura 6.15, donde se ve que, tal como se eligió en la aplicación *Meshing Tool Generator*, sólo se pueden guardar mallas a los formatos *Xyzu* y *Comsol*.

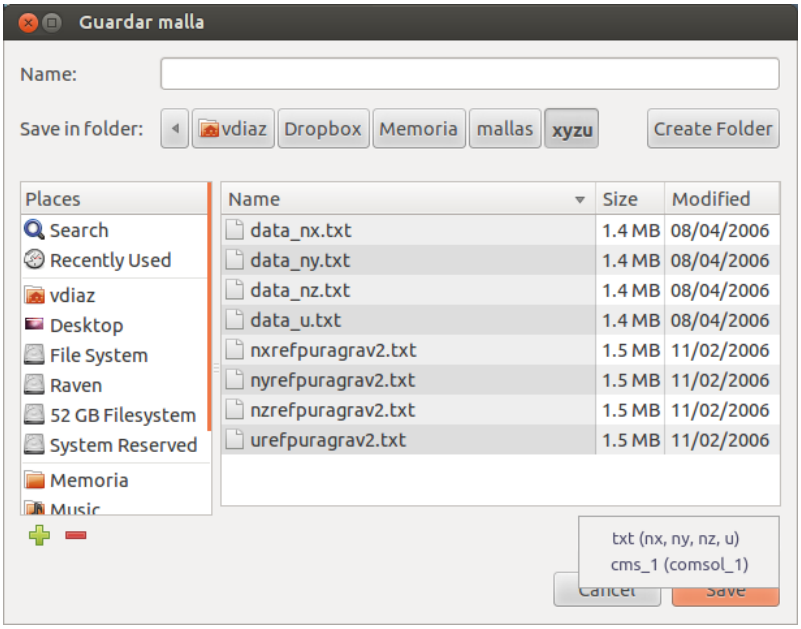

Figura 6.15: Ventana de diálogo para guardar mallas.

Al escoger generar una malla nueva, se abre la ventana mostrada en la Figura 6.16, donde se ve que, tal como se eligió en la aplicación *Meshing Tool Generator*, sólo se puede generar una malla nueva con forma de cilindro.

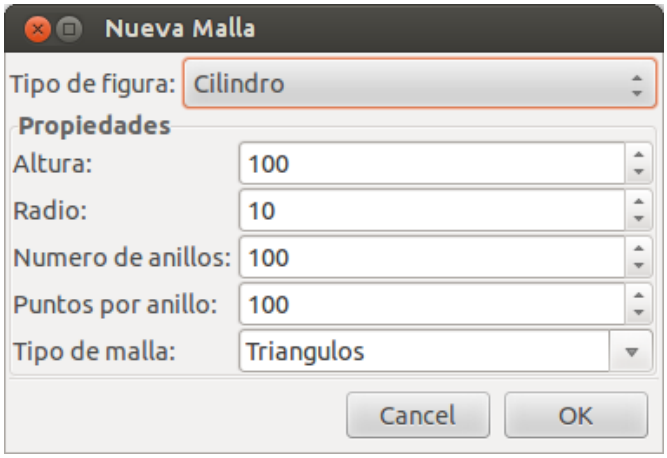

Figura 6.16: Ventana de diálogo para generar una malla nueva.

Al escoger refinar la malla, se despliega la ventana mostrada en la Figura 6.17, donde se muestra que aparecen sólo el algoritmo y criterio seleccionados en la aplicación *Meshing Tool Generator*.

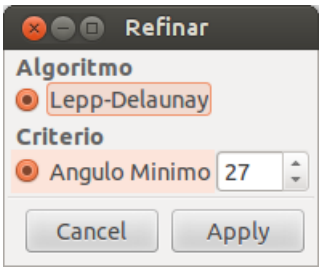

Figura 6.17: Ventana de diálogo para refinar mallas.

Finalmente, al escoger deformar la malla, se despliega la ventana mostrada en la Figura 6.18, donde se muestra que aparecen sólo los algoritmos seleccionados en la aplicación *Meshing Tool Generator*.

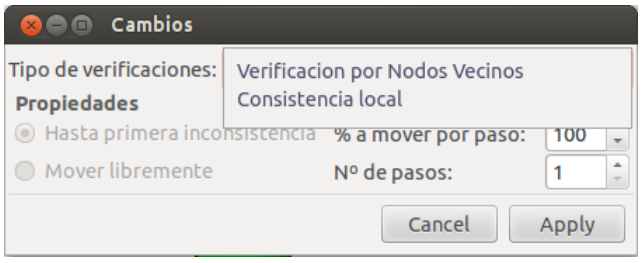

Figura 6.18: Ventana de diálogo para deformar mallas.

6.3. Comparación de los productos obtenidos

El desarrollo del presente trabajo fue realizado en un computador con las siguientes características:

- Procesador: Intel Core i3-370M 2.4 GHz
- Memoria RAM: 4 GB
- Sistema Operativo: Ubuntu 12.04 LTS, 64-bit

Para verificar que las configuraciones generadas efectivamente utilizan sólo las clases requeridas, se ejecutó la aplicación *Meshing Tool Generator* para construir los siguientes productos de ejemplo (además de los ya construidos en las secciones anteriores):

- **Generador genérico de mallas minimal**: Es capaz sólo de mostrar información, cargar, guardar y mejorar mallas. Se escogió sólo el formato *Off* para las funcionalidades de cargar y guardar.
- **Software TGS minimal**: Es capaz sólo de mostrar información, cargar, guardar, generar mallas a partir de un cilindro y distribuir hormona. Se escogió sólo el formato *Off* para las funcionalidades de cargar y guardar.
- **Generador genérico de mallas maximal**: Incluye todas las funcionalidades, formatos, algoritmos y criterios posibles.
- **Software TGS maximal**: Incluye todas las funcionalidades, formatos, algoritmos y criterios posibles.

La Tabla 6.1 compara el tamaño del código fuente, el tamaño del ejecutable producido y el tiempo de compilación para los ejemplos expuestos en las secciones 6.1 y 6.2, junto con estos productos de ejemplo. Dado que no hay variación entre el tiempo de ejecución del software base (que incluye todas las funcionalidades y variantes) y el tiempo de ejecución de los productos generados, esta variable no fue considerada en la comparación.

Producto	**Tamaño del código fuente (kB)**	**Tamaño del ejecutable (kB)**	**Tiempo de compilación (s)**
Generador genérico minimal	274.4	508.1	26.775
TGS minimal	322.2	614.1	38.489
Ejemplo de Generador genérico (sección 6.1)	316.5	585	33.569
Ejemplo de TGS (sección 6.2)	408.3	781.8	55.346
Generador genérico maximal	370.2	700.1	44.763
TGS maximal	449.7	864.3	61.412

Tabla 6.1: Tabla comparativa de los resultados obtenidos.

Se observa que, a medida que se incluyen más funcionalidades en el producto, el tamaño del ejecutable correspondiente y el tiempo de compilación aumentan, lo cual es lógico, dado que la inclusión de funcionalidades implica la inclusión de clases en la compilación.

Capítulo 7
Conclusiones

El resultado del trabajo desarrollado es una interfaz gráfica, llamada *Meshing Tool Generator*, que permite al usuario configurar y crear distintos productos generadores de mallas geométricas, incluyendo solamente las funcionalidades y variantes especificadas por él. El producto generado puede cargar, almacenar o generar mallas, y también aplicar algoritmos de refinamiento, desrefinamiento, mejoramiento, deformación y distribución de hormona. Se obtuvo además una nueva versión del software TGS, llamada *Mesh Generator*, la cual permite la eliminación de las dependencias innecesarias al momento de configurar el sistema.

Esta memoria valida el modelo propuesto en [8] para el diseño e implementación de generadores de mallas; en definitiva, se logró generar una línea de productos de software generadores de mallas geométricas, compuesta por el software TGS y el Generador genérico de mallas. Esto resulta muy beneficioso, puesto que conlleva una disminución en el esfuerzo requerido para producir productos nuevos y un aumento en la calidad de éstos, pues las componentes desarrolladas han sido testeadas; además, es más fácil determinar el precio de un producto desarrollado, pues sus componentes son conocidas.

El objetivo de la memoria está cumplido, pues los productos generados efectivamente varían en tamaño dependiendo de las elecciones del usuario, e incluyen las clases estrictamente necesarias. Sin embargo, tras un examen más exhaustivo del código del software TGS, se observó que no fue construido pensando en una posible generalización, pues sus clases poseen un alto nivel de acoplamiento y por lo tanto se deben revisar y separar. De lograrse esta separación, sería posible generar productos que utilicen exactamente lo que se necesita.

Como trabajo futuro se propone la revisión de las clases de mayor tamaño del software TGS; en particular, la clase `Malla` merece atención, ya que es la clase de mayor tamaño y complejidad, además de ser común a todos los productos. Esta clase incurre en un error conceptual, pues contiene métodos que deberían ser parte del proceso específico que los usa; por ejemplo, se hace cargo de comportamientos correspondientes a los nodos, arcos y caras. Por lo tanto, es importante que encapsule solo con las funcionalidades que le correspondan, y que son utilizadas por todos los productos.

Además, se observó el siguiente comportamiento dentro de todas las clases que corresponden a algoritmos o funcionalidades dentro del software TGS: el constructor de la clase recibe uno o más valores numéricos (que denotan las opciones elegidas por el usuario) por parte de la

clase `GUIVentanaPrincipal`, y dependiendo de estos valores llama a los constructores de sus variantes asociadas. Esto se considera erróneo, pues limita la extensibilidad de la aplicación. En un principio se pensó en trasladar los constructores de las variantes fuera de la clase, pasando los objetos ya construidos en vez de valores numéricos, pero aquello aumentaría el acoplamiento para la clase `GUIVentanaPrincipal`, pues necesitaría acceder a los constructores de todas las variantes. El manejo de esta situación queda propuesto como trabajo futuro.

Por otra parte, en el software *Meshing Tool Generator* existen detalles que se pueden mejorar; por ejemplo, en la clase que genera el archivo *makefile*, los nombres de los archivos a incluir en la compilación son ingresados en duro, lo cual dificulta la extensibilidad de la aplicación; esto podría solucionarse estableciendo una convención para los nombres de los archivos correspondientes a las funcionalidades o características variables, lo cual facilitaría la generación automática del archivo *makefile*, o también creando un repositorio o base de datos para los componentes.

Finalmente, se espera la adición de nuevos sistemas generadores de mallas geométricas a la línea de productos ya construida, pues *Meshing Tool Generator* puede ser fácilmente extendido para crear nuevos productos. Por ejemplo, es posible la configuración y creación del software *FaceAnimator*, el cual permite modelar rostros humanos como mallas geométricas con el fin de simular su comportamiento en diferentes contextos, como por ejemplo videoconferencias. Esta aplicación también provee algunas de las funcionalidades descritas en esta memoria, como las de refinar, desrefinar y mejorar la malla [11].

Capítulo 8
Bibliografía

[1] María Cecilia Bastarrica, Nancy Hitschfeld, "Designing a product family of meshing tools", Advances in Engineering Software 37(2006): 1-10.

[2] Paul Clements, Linda Northrop, "Software Product Lines: Practices and Patterns", SEI Series in Software Engineering, Addison-Wesley, 2002.

[3] Kwanwoo Lee, Kyo C. Kang, Jaejoon Lee, "Concepts and Guidelines of Feature Modeling for Product Line Software Engineering", Department of Computer Science and Engineering, Pohang University of Science and Technology, Korea, 2002.

[4] Ricardo Medina, "Modelador de cambios en la geometría de objetos utilizando mallas geométricas", Memoria para optar al título de Ingeniero Civil en Computación, 2005.

[5] Cristina Melo, "Desarrollo de una herramienta que genera mallas de superficie compuestas de cuadriláteros para modelar el crecimiento de árboles", Memoria para optar al título de Ingeniero Civil en Computación, 2008.

[6] Klaus Pohl, Günter Böckle, Frank van der Linden, "Software Product Line Engineering. Foundations, Principles and Techniques", Springer, 2005.

[7] Roger S. Pressman, "Ingeniería del software: un enfoque práctico", McGraw-Hill, 2002.

[8] Pedro O. Rossel, María Cecilia Bastarrica, Nancy Hitschfeld, "A Systematic Process for Defining Meshing Tool Software Product Line Domain Model", Proceedings of the WER'09: 12th Workshop on Requirements Engineering, pp. 103-114, Valparaíso, Chile, Julio 2009.

[9] Nicolás Silva, "Modelamiento del crecimiento de árboles usando mallas de superficie", Memoria para optar al título de Ingeniero Civil en Computación, 2007.

[10] Gonzalo Urroz, "Adaptación de software de aplicación al paradigma de la ingeniería de línea de productos de software", Memoria para optar al título de Ingeniero Civil en Computación, 2011.

[11] Renato Valenzuela, "Creación de una herramienta para la visualización de animaciones de rostros", Memoria para optar al título de Ingeniero Civil en Computación, 2009.

Anexo A
Código

A.1. Clase MeshingToolGenerator

```
#define FILE_STEP1 "./Step1_eng.glade"
#define FILE_STEP2 "./Step2_eng.glade"
#define FILE_STEP3 "./Step3_eng.glade"

#include "ConfigGenerator.hh"
#include "HeaderGenerator.hh"
#include "MakefileGenerator.hh"
#include "MeshingToolGenerator.hh"

MeshingToolGenerator::MeshingToolGenerator()
{
        set_title("Meshing Tool Generator");
        set_border_width(10);
        set_resizable(false);
        set_position(Gtk::WIN_POS_CENTER_ALWAYS);

    builder1 = Gtk::Builder::create_from_file(FILE_STEP1);
    builder1->get_widget("boxStep1", step1);
    builder1->get_widget("seleccionarTGS", seleccionarTGS);

    builder2 = Gtk::Builder::create_from_file(FILE_STEP2);
    builder2->get_widget("boxStep2", step2);
    builder2->get_widget("cargarBox", cargarBox);
    builder2->get_widget("guardarBox", guardarBox);
    builder2->get_widget("generarBox", generarBox);

    builder2->get_widget("generarSi", generarSi);
    generarSi->signal_clicked().connect(sigc::mem_fun( *this, &
       MeshingToolGenerator::on_generarSi_changed ));
    builder2->get_widget("generarNo", generarNo);
    generarNo->signal_clicked().connect(sigc::mem_fun( *this, &
       MeshingToolGenerator::on_generarNo_changed ));

    builder2->get_widget("figuraGenerar", figuraGenerar);
```

```
builder2->get_widget("generarMedula", generarMedula);
generarMedula->signal_toggled().connect(sigc::mem_fun( *this, &
   MeshingToolGenerator::on_step2_toggled ));
builder2->get_widget("generarCilindro", generarCilindro);
generarCilindro->signal_toggled().connect(sigc::mem_fun( *this,
   &MeshingToolGenerator::on_step2_toggled ));

builder2->get_widget("guardarOff", guardarOff);
guardarOff->signal_toggled().connect(sigc::mem_fun( *this, &
   MeshingToolGenerator::on_step2_toggled ));
builder2->get_widget("guardarXyzu", guardarXyzu);
guardarXyzu->signal_toggled().connect(sigc::mem_fun( *this, &
   MeshingToolGenerator::on_step2_toggled ));
builder2->get_widget("guardarComsol", guardarComsol);
guardarComsol->signal_toggled().connect(sigc::mem_fun( *this, &
   MeshingToolGenerator::on_step2_toggled ));

builder2->get_widget("cargarOff", cargarOff);
cargarOff->signal_toggled().connect(sigc::mem_fun( *this, &
   MeshingToolGenerator::on_step2_toggled ));
builder2->get_widget("cargarXyzu", cargarXyzu);
cargarXyzu->signal_toggled().connect(sigc::mem_fun( *this, &
   MeshingToolGenerator::on_step2_toggled ));
builder2->get_widget("cargarMatlab", cargarMatlab);
cargarMatlab->signal_toggled().connect(sigc::mem_fun( *this, &
   MeshingToolGenerator::on_step2_toggled ));
builder2->get_widget("cargarComsol", cargarComsol);
cargarComsol->signal_toggled().connect(sigc::mem_fun( *this, &
   MeshingToolGenerator::on_step2_toggled ));

builder3 = Gtk::Builder::create_from_file(FILE_STEP3);
builder3->get_widget("boxStep3", step3);
builder3->get_widget("distribBox", distribBox);

builder3->get_widget("mejorarSi", mejorarSi);
mejorarSi->signal_clicked().connect(sigc::mem_fun( *this, &
   MeshingToolGenerator::on_step3_toggled ));
builder3->get_widget("mejorarNo", mejorarNo);
mejorarNo->signal_clicked().connect(sigc::mem_fun( *this, &
   MeshingToolGenerator::on_step3_toggled ));

builder3->get_widget("refinarSi", refinarSi);
refinarSi->signal_clicked().connect(sigc::mem_fun( *this, &
   MeshingToolGenerator::on_refinarSi_changed ));
builder3->get_widget("refinarNo", refinarNo);
refinarNo->signal_clicked().connect(sigc::mem_fun( *this, &
   MeshingToolGenerator::on_refinarNo_changed ));

builder3->get_widget("algoritmoRefinar", algoritmoRefinar);
builder3->get_widget("leppDelaunay", leppDelaunay);
```

```
leppDelaunay->signal_toggled().connect(sigc::mem_fun( *this, &
   MeshingToolGenerator::on_step3_toggled ));
builder3->get_widget("longestEdge", longestEdge);
longestEdge->signal_toggled().connect(sigc::mem_fun( *this, &
   MeshingToolGenerator::on_step3_toggled ));

builder3->get_widget("criterioRefinar", criterioRefinar);
builder3->get_widget("anguloMin", anguloMin);
anguloMin->signal_toggled().connect(sigc::mem_fun( *this, &
   MeshingToolGenerator::on_step3_toggled ));
builder3->get_widget("areaMax", areaMax);
areaMax->signal_toggled().connect(sigc::mem_fun( *this, &
   MeshingToolGenerator::on_step3_toggled ));
builder3->get_widget("largoMax", largoMax);
largoMax->signal_toggled().connect(sigc::mem_fun( *this, &
   MeshingToolGenerator::on_step3_toggled ));

builder3->get_widget("desrefinarSi", desrefinarSi);
desrefinarSi->signal_clicked().connect(sigc::mem_fun( *this, &
   MeshingToolGenerator::on_desrefinarSi_changed ));
builder3->get_widget("desrefinarNo", desrefinarNo);
desrefinarNo->signal_clicked().connect(sigc::mem_fun( *this, &
   MeshingToolGenerator::on_desrefinarNo_changed ));

builder3->get_widget("criterioDesrefinar", criterioDesrefinar);
builder3->get_widget("areaMin", areaMin);
areaMin->signal_toggled().connect(sigc::mem_fun( *this, &
   MeshingToolGenerator::on_step3_toggled ));
builder3->get_widget("largoMin", largoMin);
largoMin->signal_toggled().connect(sigc::mem_fun( *this, &
   MeshingToolGenerator::on_step3_toggled ));

builder3->get_widget("distribSi", distribSi);
distribSi->signal_clicked().connect(sigc::mem_fun( *this, &
   MeshingToolGenerator::on_step3_toggled ));
builder3->get_widget("distribNo", distribNo);
distribNo->signal_clicked().connect(sigc::mem_fun( *this, &
   MeshingToolGenerator::on_step3_toggled ));

builder3->get_widget("deformarSi", deformarSi);
deformarSi->signal_clicked().connect(sigc::mem_fun( *this, &
   MeshingToolGenerator::on_deformarSi_changed ));
builder3->get_widget("deformarNo", deformarNo);
deformarNo->signal_clicked().connect(sigc::mem_fun( *this, &
   MeshingToolGenerator::on_deformarNo_changed ));

builder3->get_widget("algoritmoDeformar", algoritmoDeformar);
builder3->get_widget("nodosVecinos", nodosVecinos);
nodosVecinos->signal_toggled().connect(sigc::mem_fun( *this, &
   MeshingToolGenerator::on_step3_toggled ));
```

```
    builder3->get_widget("consLocal", consLocal);
    consLocal->signal_toggled().connect(sigc::mem_fun( *this, &
       MeshingToolGenerator::on_step3_toggled ));
    builder3->get_widget("sinVerificacion", sinVerificacion);
    sinVerificacion->signal_toggled().connect(sigc::mem_fun( *this,
       &MeshingToolGenerator::on_step3_toggled ));

    append_page(*step1);
    append_page(*step2);
    append_page(*step3);

    set_page_title(*get_nth_page(0), "Generator Type");
    set_page_title(*get_nth_page(1), "File Handling");
    set_page_title(*get_nth_page(2), "Mesh Processing Algorithms");

    set_page_type(*step1, Gtk::ASSISTANT_PAGE_INTRO);
    set_page_type(*step2, Gtk::ASSISTANT_PAGE_CONTENT);
    set_page_type(*step3, Gtk::ASSISTANT_PAGE_CONFIRM);

    set_page_complete(*step1, true);
}

void MeshingToolGenerator::on_generarSi_changed()
{
    figuraGenerar->set_sensitive(true);
    generarMedula->set_sensitive(true);
    generarCilindro->set_sensitive(true);
    set_page_complete(*step2, (validaCargar() && validaGuardar() &&
       validaGenerar()));
}

void MeshingToolGenerator::on_generarNo_changed()
{
    figuraGenerar->set_sensitive(false);
    generarMedula->set_active(false);
    generarMedula->set_sensitive(false);
    generarCilindro->set_active(false);
    generarCilindro->set_sensitive(false);
    set_page_complete(*step2, (validaCargar() && validaGuardar() &&
       validaGenerar()));
}

void MeshingToolGenerator::on_step2_toggled()
{
    set_page_complete(*step2, (validaCargar() && validaGuardar() &&
       validaGenerar()));
}

bool MeshingToolGenerator::validaCargar()
{
```

```
        return cargarOff->get_active() || cargarXyzu->get_active()
           || cargarMatlab->get_active() || cargarComsol->get_active
           ();
}

bool MeshingToolGenerator::validaGuardar()
{
        return guardarOff->get_active() || guardarXyzu->get_active()
            || guardarComsol->get_active();
}

bool MeshingToolGenerator::validaGenerar()
{
        return generarNo->get_active() || (generarSi->get_active()
           && (generarMedula->get_active() || generarCilindro->
           get_active()));
}

void MeshingToolGenerator::on_refinarSi_changed() {
    criterioRefinar->set_sensitive(true);
    anguloMin->set_sensitive(true);
    areaMax->set_sensitive(true);
    largoMax->set_sensitive(true);

    algoritmoRefinar->set_sensitive(true);
    leppDelaunay->set_sensitive(true);
    longestEdge->set_sensitive(true);
    set_page_complete(*step3, validaStep3());
}

void MeshingToolGenerator::on_refinarNo_changed() {
    criterioRefinar->set_sensitive(false);
    anguloMin->set_active(false);
    anguloMin->set_sensitive(false);
    areaMax->set_active(false);
    areaMax->set_sensitive(false);
    largoMax->set_active(false);
    largoMax->set_sensitive(false);

    algoritmoRefinar->set_sensitive(false);
    leppDelaunay->set_active(false);
    leppDelaunay->set_sensitive(false);
    longestEdge->set_active(false);
    longestEdge->set_sensitive(false);
    set_page_complete(*step3, validaStep3());
}

void MeshingToolGenerator::on_desrefinarSi_changed() {
    criterioDesrefinar->set_sensitive(true);
    areaMin->set_sensitive(true);
```

```
    largoMin->set_sensitive(true);
    set_page_complete(*step3, validaStep3());
}

void MeshingToolGenerator::on_desrefinarNo_changed() {
    criterioDesrefinar->set_sensitive(false);
    areaMin->set_active(false);
    areaMin->set_sensitive(false);
    largoMin->set_active(false);
    largoMin->set_sensitive(false);
    set_page_complete(*step3, validaStep3());
}

void MeshingToolGenerator::on_deformarSi_changed() {
    algoritmoDeformar->set_sensitive(true);
    nodosVecinos->set_sensitive(true);
    consLocal->set_sensitive(true);
    sinVerificacion->set_sensitive(true);
    set_page_complete(*step3, validaStep3());
}

void MeshingToolGenerator::on_deformarNo_changed() {
    algoritmoDeformar->set_sensitive(false);
    nodosVecinos->set_active(false);
    nodosVecinos->set_sensitive(false);
    consLocal->set_active(false);
    consLocal->set_sensitive(false);
    sinVerificacion->set_active(false);
    sinVerificacion->set_sensitive(false);
    set_page_complete(*step3, validaStep3());
}

void MeshingToolGenerator::on_step3_toggled()
{
    set_page_complete(*step3, validaStep3());
}

bool MeshingToolGenerator::validaRefinar()
{
        return refinarSi->get_active() && (leppDelaunay->get_active
           () || longestEdge->get_active()) && (anguloMin->
           get_active() || areaMax->get_active() || largoMax->
           get_active());
}

bool MeshingToolGenerator::validaDesrefinar()
{
        return desrefinarSi->get_active() && (areaMin->get_active()
           || largoMin->get_active());
}
```

```
bool MeshingToolGenerator::validaDeformar()
{
        return deformarSi->get_active() && (nodosVecinos->get_active
           () || consLocal->get_active() || sinVerificacion->
           get_active());
}

bool MeshingToolGenerator::validaStep3()
{
        return mejorarSi->get_active() || validaRefinar() ||
           validaDesrefinar() || distribSi->get_active() ||
           validaDeformar();
}

void MeshingToolGenerator::on_apply()
{
    using std::map;
    using std::string;

    ConfigGenerator* config_generator = new ConfigGenerator();
    config_generator->GenerarConfigStep1(seleccionarTGS->get_active
       ());

    map<string, bool> formatosCargar;
    formatosCargar.insert(make_pair("Off", cargarOff->get_active()))
       ;
    formatosCargar.insert(make_pair("Xyzu", cargarXyzu->get_active()
       ));
    formatosCargar.insert(make_pair("Matlab", cargarMatlab->
       get_active()));
    formatosCargar.insert(make_pair("Comsol", cargarComsol->
       get_active()));

    map<string, bool> formatosGuardar;
    formatosGuardar.insert(make_pair("Off", guardarOff->get_active()
       ));
    formatosGuardar.insert(make_pair("Xyzu", guardarXyzu->get_active
       ()));
    formatosGuardar.insert(make_pair("Comsol", guardarComsol->
       get_active()));

    map<string, bool> formatosGenerar;
    if (generarSi->get_active()) {
        formatosGenerar.insert(make_pair("Medulla", generarMedula->
           get_active()));
        formatosGenerar.insert(make_pair("Cylinder", generarCilindro
           ->get_active()));
    }
```

```
config_generator->GenerarConfigStep2(formatosCargar,
   formatosGuardar, formatosGenerar);

map<string, bool> algoritmosRefinar, criteriosRefinar;
if (refinarSi->get_active()) {
    algoritmosRefinar.insert(make_pair("LeppDelaunay",
       leppDelaunay->get_active()));
    algoritmosRefinar.insert(make_pair("LongestEdge",
       longestEdge->get_active()));

    criteriosRefinar.insert(make_pair("MinAngle", anguloMin->
       get_active()));
    criteriosRefinar.insert(make_pair("MaxArea", areaMax->
       get_active()));
    criteriosRefinar.insert(make_pair("MaxLength", largoMax->
       get_active()));
}

map<string, bool> criteriosDesrefinar;
if (desrefinarSi->get_active()) {
    criteriosDesrefinar.insert(make_pair("MinArea", areaMin->
       get_active()));
    criteriosDesrefinar.insert(make_pair("MinLength", largoMin->
       get_active()));;
}

map<string, bool> algoritmosDeformar;
if (deformarSi->get_active()) {
    algoritmosDeformar.insert(make_pair("NeighborNodes",
       nodosVecinos->get_active()));
    algoritmosDeformar.insert(make_pair("LocalConsistency",
       consLocal->get_active()));
    algoritmosDeformar.insert(make_pair("NoVerification",
       sinVerificacion->get_active()));
}

config_generator->GenerarConfigStep3(mejorarSi->get_active(),
   algoritmosRefinar, criteriosRefinar, criteriosDesrefinar,
   distribSi->get_active(), algoritmosDeformar);

cout << "Configuration file successfully generated." << endl;

config_generator->CopyFile();

MakefileGenerator* makefile_generator = new MakefileGenerator();
makefile_generator->Generate();
makefile_generator->CopyFile();

cout << "Makefile successfully generated." << endl;
```

```
    HeaderGenerator* header_generator = new HeaderGenerator();
    header_generator->Generate();
    header_generator->CopyFile();

    cout << "Header successfully generated." << endl;
    hide();
}

void MeshingToolGenerator::on_cancel()
{
    hide();
}

void MeshingToolGenerator::on_close()
{
    hide();
}

void MeshingToolGenerator::on_prepare(Gtk::Widget* page)
{
    if (get_current_page() == 1)
    {
        cargarMatlab->set_visible(seleccionarTGS->get_active());
        generarBox->set_visible(seleccionarTGS->get_active());
        set_page_complete(*step2, (validaCargar() && validaGuardar()
            && validaGenerar()));
    }

    if (get_current_page() == 2)
    {
        distribBox->set_visible(seleccionarTGS->get_active());
        set_page_complete(*step3, validaStep3());
    }
}

int main (int argc, char **argv)
{
        Glib::RefPtr<Gtk::Application> app = Gtk::Application::
           create(argc, argv, "org.gtkmm.example");

        MeshingToolGenerator* mtGenerator = new MeshingToolGenerator
           ();
    app->run(*mtGenerator);

        return 0;
}
```

A.2. Clase ConfigGenerator

```
#include "ConfigGenerator.hh"
#define CONFIG_FILENAME "./config.cfg"
#define FILE_HEADER "./config.h"
#define MG_DIRECTORY "../MeshGenerator/"

void ConfigGenerator::GenerarConfigStep1 (bool treegrowth)
{
    Config cfg;
        Setting &root = cfg.getRoot();
        root.add("TreeGrowthSimulator", Setting::TypeBoolean) =
           treegrowth;
        cfg.writeFile(CONFIG_FILENAME);
}

void ConfigGenerator::GenerarConfigStep2 (map<string, bool>
   formatosCargar, map<string, bool> formatosGuardar, map<string,
   bool> formatosGenerar)
{
    Config cfg;
    cfg.readFile(CONFIG_FILENAME);
        Setting &root = cfg.getRoot();
    map<string, bool>::iterator it;

    Setting &cargarMalla = root.add("LoadMesh", Setting::TypeGroup);
    for(it = formatosCargar.begin(); it != formatosCargar.end(); it
       ++) {
        cargarMalla.add(it->first, Setting::TypeBoolean) = it->
           second;
    }

    Setting &guardarMalla = root.add("SaveMesh", Setting::TypeGroup)
       ;
    for(it = formatosGuardar.begin(); it != formatosGuardar.end();
       it++) {
        guardarMalla.add(it->first, Setting::TypeBoolean) = it->
           second;
    }

    if (formatosGenerar.size() == 0) {
        root.add("GenerateMesh", Setting::TypeBoolean) = false;
    }
    else {
        Setting &generarMalla = root.add("GenerateMesh", Setting::
           TypeGroup);

        for(it = formatosGenerar.begin(); it != formatosGenerar.end
           (); it++) {
            generarMalla.add(it->first, Setting::TypeBoolean) = it->
               second;
        }
```

```
    }

    cfg.writeFile(CONFIG_FILENAME);
}

void ConfigGenerator::GenerarConfigStep3 (bool mejorar, map<string,
   bool> algoritmosRefinar, map<string, bool> criteriosRefinar, map<
   string, bool> criteriosDesrefinar, bool distribHormona, map<
   string, bool> algoritmosDeformar)
{
    Config cfg;
    cfg.readFile(CONFIG_FILENAME);
        Setting &root = cfg.getRoot();
    map<string, bool>::iterator it;

    root.add("ImproveMesh", Setting::TypeBoolean) = mejorar;

    if (algoritmosRefinar.size() == 0 && criteriosRefinar.size() ==
       0) {
        root.add("RefineMesh", Setting::TypeBoolean) = false;
    }

    else {
        Setting &refinarMalla = root.add("RefineMesh", Setting::
           TypeGroup);

        Setting &algoritmos = refinarMalla.add("Algorithms", Setting
           ::TypeGroup);
        for(it = algoritmosRefinar.begin(); it != algoritmosRefinar.
           end(); it++) {
            algoritmos.add(it->first, Setting::TypeBoolean) = it->
               second;
        }

        Setting &criterios = refinarMalla.add("Criteria", Setting::
           TypeGroup);
        for(it = criteriosRefinar.begin(); it != criteriosRefinar.
           end(); it++) {
            criterios.add(it->first, Setting::TypeBoolean) = it->
               second;
        }
    }

    if (criteriosDesrefinar.size() == 0) {
        root.add("DerefineMesh", Setting::TypeBoolean) = false;
    }

    else {
        Setting &desrefinarMalla = root.add("DerefineMesh", Setting
           ::TypeGroup);
```

```
        for(it = criteriosDesrefinar.begin(); it !=
           criteriosDesrefinar.end(); it++) {
            desrefinarMalla.add(it->first, Setting::TypeBoolean) =
               it->second;
        }
    }

    root.add("DistributeHormone", Setting::TypeBoolean) =
       distribHormona;

    if (algoritmosDeformar.size() == 0) {
        root.add("DeformMesh", Setting::TypeBoolean) = false;
    }

    else {
        Setting &deformarMalla = root.add("DeformMesh", Setting::
           TypeGroup);

        for(it = algoritmosDeformar.begin(); it !=
           algoritmosDeformar.end(); it++) {
            deformarMalla.add(it->first, Setting::TypeBoolean) = it
               ->second;
        }
    }

    cfg.writeFile(CONFIG_FILENAME);
}

void ConfigGenerator::CopyFile()
{
    char command[40];
    strcpy(command, "cp ");
    strcat(command, CONFIG_FILENAME);
    strcat(command, " ");
    strcat(command, MG_DIRECTORY);

    system(command);
}
```

A.3. Clase HeaderGenerator

```
#include "HeaderGenerator.hh"
#define CONFIG_FILENAME "./config.cfg"
#define HEADER_FILENAME "./config.h"
#define MG_DIRECTORY "../MeshGenerator/"

HeaderGenerator::HeaderGenerator()
{
```

```
        cfg.readFile(CONFIG_FILENAME);
}

const char* HeaderGenerator::StringToUpper(string str)
{
    for (string::iterator p = str.begin(); str.end() != p; ++p)
        *p = toupper(*p);

    return str.c_str();
}

void HeaderGenerator::Generate()
{
    Setting &root = cfg.getRoot();
    ofstream headerfile (HEADER_FILENAME);

    headerfile << "#ifndef CONFIG_H_INCLUDED" << endl;
    headerfile << "#define CONFIG_H_INCLUDED" << endl << endl;

    Setting &treeGrowth = root["TreeGrowthSimulator"];
    if (treeGrowth.getType() == Setting::TypeBoolean && treeGrowth)
        headerfile << "#define TREE_GROWTH" << endl;

    bool value;
    string name;

    Setting &cargarMalla = root["LoadMesh"];
    for (int i = 0; i < cargarMalla.getLength(); i++) {
        name = cargarMalla[i].getName();
        cargarMalla.lookupValue(name, value);

        if (value)
            headerfile << "#define LOAD_" << StringToUpper(name) <<
               endl;
    }

    Setting &guardarMalla = root["SaveMesh"];
    for (int i = 0; i < guardarMalla.getLength(); i++) {
        name = guardarMalla[i].getName();
        guardarMalla.lookupValue(name, value);

        if (value)
            headerfile << "#define SAVE_" << StringToUpper(name) <<
               endl;
    }

    Setting &generarMalla = root["GenerateMesh"];
    if (generarMalla.getType() == Setting::TypeGroup)
    {
        for (int i = 0; i < generarMalla.getLength(); i++) {
```

```
        name = generarMalla[i].getName();
        generarMalla.lookupValue(name, value);

        if (value)
            headerfile << "#define GENERATE_" << StringToUpper(
               name) << endl;
    }
}

Setting &mejorarMalla = root["ImproveMesh"];
if (mejorarMalla.getType() == Setting::TypeBoolean &&
   mejorarMalla)
    headerfile << "#define IMPROVE" << endl;

Setting &refinarMalla = root["RefineMesh"];
if (refinarMalla.getType() == Setting::TypeGroup)
{
    Setting &algoritmosRefinar = refinarMalla["Algorithms"];
    for (int i = 0; i < algoritmosRefinar.getLength(); i++) {
        name = algoritmosRefinar[i].getName();
        algoritmosRefinar.lookupValue(name, value);

        if (value)
            headerfile << "#define REFINE_ALG_" << StringToUpper
               (name) << endl;
    }

    Setting &criteriosRefinar = refinarMalla["Criteria"];
    for (int i = 0; i < criteriosRefinar.getLength(); i++) {
        name = criteriosRefinar[i].getName();
        criteriosRefinar.lookupValue(name, value);

        if (value)
            headerfile << "#define REFINE_CRITERION_" <<
               StringToUpper(name) << endl;
    }
}

Setting &desrefinarMalla = root["DerefineMesh"];
if (desrefinarMalla.getType() == Setting::TypeGroup)
{
    for (int i = 0; i < desrefinarMalla.getLength(); i++) {
        name = desrefinarMalla[i].getName();
        desrefinarMalla.lookupValue(name, value);

        if (value)
            headerfile << "#define DEREFINE_CRITERION_" <<
               StringToUpper(name) << endl;
    }
}
```

```
    Setting &distribHormona = root["DistributeHormone"];
    if (distribHormona.getType() == Setting::TypeBoolean &&
       distribHormona)
        headerfile << "#define DISTRIB_HORMONE" << endl;

    Setting &deformarMalla = root["DeformMesh"];
    if (deformarMalla.getType() == Setting::TypeGroup)
    {
        for (int i = 0; i < deformarMalla.getLength(); i++) {
            name = deformarMalla[i].getName();
            deformarMalla.lookupValue(name, value);

            if (value)
                headerfile << "#define DEFORM_ALG_" << StringToUpper
                   (name) << endl;
        }
    }

    headerfile << endl << "#endif // CONFIG_H_INCLUDED" << endl;
    headerfile.close();
}

void HeaderGenerator::CopyFile()
{
    char command[40];

    strcpy(command, "cp ");
    strcat(command, HEADER_FILENAME);
    strcat(command, " ");
    strcat(command, MG_DIRECTORY);
    strcat(command, "/src");

    system(command);
}
```

A.4. Clase MakefileGenerator

```
#include "MakefileGenerator.hh"
#define CONFIG_FILENAME "./config.cfg"
#define MAKEFILE_NAME "./makefiletgs"
#define MG_MAKEFILE "../MeshGenerator/makefile"

MakefileGenerator::MakefileGenerator()
{
        cfg.readFile(CONFIG_FILENAME);
}

void MakefileGenerator::Generate()
```

```
{
    Setting &root = cfg.getRoot();
    ofstream mkfile (MAKEFILE_NAME);

    Setting &treeGrowth = root["TreeGrowthSimulator"];

    if (treeGrowth.getType() == Setting::TypeBoolean && treeGrowth)
        mkfile << "EXEC = treegrowthsimulator" << endl;

    else
        mkfile << "EXEC = meshgenerator" << endl;

    mkfile << "CC = g++" << endl;
    mkfile << "LFLAGS = -O2 -lgsl -lgslcblas -lconfig++" << endl;
    mkfile << "PKG_CONFIG = `pkg-config gtkglextmm-1.2 --cflags --
       libs sigc++-2.0`" << endl << endl;

    mkfile << "SOURCES = ";
    mkfile << "./src/arco.cpp ./src/arcos.cpp ";
    mkfile << "./src/cara.cpp ./src/caras.cpp ";
    mkfile << "./src/comando.cpp ./src/configreader.cpp ";
    mkfile << "./src/generar.cpp ./src/guardar.cpp ";
    mkfile << "./src/GUIDialogInformacion.cc ./src/
       GUIDialogInformacion_glade.cc ";
    mkfile << "./src/GUIVentanaPrincipal.cc ./src/
       GUIVentanaPrincipal_glade.cc ";
    mkfile << "./src/informacionmalla.cpp ./src/informar.cpp ";
    mkfile << "./src/malla.cpp ./src/nodo.cpp ./src/nodos.cpp ";
    mkfile << "./src/polinomio.cpp ./src/punto.cpp ";
    mkfile << "./src/segmenttriangleintersection.cpp ./src/segtriint
       .cpp ";
    mkfile << "./src/SimpleGLScene.cc ./src/treegrowthsimulator.cc
       ./src/vect.cpp ";

    bool value;
    Setting &cargarMalla = root["LoadMesh"];
    cargarMalla.lookupValue("Comsol", value);
    if (value)
        mkfile << "./src/generafromcomsol_1.cpp ";

    cargarMalla.lookupValue("Matlab", value);
    if (value) {
        mkfile << "./src/generafrommatlab.cpp ";
        mkfile << "./src/GUIDialogAbrirMatlab.cc ";
        mkfile << "./src/GUIDialogAbrirMatlab_glade.cc ";
    }

    cargarMalla.lookupValue("Off", value);
    if (value)
        mkfile << "./src/generafromoff.cpp ";
```

```
cargarMalla.lookupValue("Xyzu", value);
if (value)
    mkfile << "./src/generafromarchivosxyzu.cpp ";


Setting &guardarMalla = root["SaveMesh"];
guardarMalla.lookupValue("Comsol", value);
if (value)
    mkfile << "./src/almacenatocomsol_1.cpp ";

guardarMalla.lookupValue("Off", value);
if (value)
    mkfile << "./src/almacenatooff.cpp ";

guardarMalla.lookupValue("Xyzu", value);
if (value)
    mkfile << "./src/almacenatoxyzu.cpp ";


Setting &generarMalla = root["GenerateMesh"];
if (generarMalla.getType() == Setting::TypeGroup)
{
    mkfile << "./src/GUIDialogNuevaMalla.cc ";
    mkfile << "./src/GUIDialogNuevaMalla_glade.cc ";

    generarMalla.lookupValue("Cylinder", value);
    if (value)
        mkfile << "./src/generacilindro.cpp ";

    generarMalla.lookupValue("Medulla", value);
    if (value)
        mkfile << "./src/generafrommedula.cpp ";
}


Setting &mejorarMalla = root["ImproveMesh"];
if (mejorarMalla.getType() == Setting::TypeBoolean &&
   mejorarMalla)
{
    mkfile << "./src/mejdelaunay.cpp ";
    mkfile << "./src/mejorar.cpp ";
}


Setting &refinarMalla = root["RefineMesh"];
bool areaMaxima = false;

if (refinarMalla.getType() == Setting::TypeGroup)
{
```

```
    mkfile << "./src/refinar.cpp ";
    mkfile << "./src/GUIDialogRefinar.cc ";
    mkfile << "./src/GUIDialogRefinar_glade.cc ";

    Setting &algoritmosRefinar = refinarMalla["Algorithms"];

    algoritmosRefinar.lookupValue("LeppDelaunay", value);
    if (value) {
        mkfile << "./src/lepp.cpp ";
        mkfile << "./src/leppdelaunay.cpp ";
    }

    algoritmosRefinar.lookupValue("LongestEdge", value);
    if (value)
        mkfile << "./src/delaunaylongestedgebisection.cpp ";

    Setting &criteriosRefinar = refinarMalla["Criteria"];

    criteriosRefinar.lookupValue("MinAngle", value);
    if (value)
        mkfile << "./src/angulominimo.cpp ";

    criteriosRefinar.lookupValue("MaxArea", areaMaxima);
    if (areaMaxima)
        mkfile << "./src/areacara.cpp ";

    criteriosRefinar.lookupValue("MaxLength", value);
    if (value)
        mkfile << "./src/largoarcomaximo.cpp ";
}


Setting &desrefinarMalla = root["DerefineMesh"];
if (desrefinarMalla.getType() == Setting::TypeGroup)
{
    mkfile << "./src/desrefinar.cpp ";
    mkfile << "./src/desrefinamientoedgecollapse.cpp ";
    mkfile << "./src/GUIDialogDesrefinar.cc ";
    mkfile << "./src/GUIDialogDesrefinar_glade.cc ";

    desrefinarMalla.lookupValue("MinArea", value);
    if (value && !areaMaxima)
        mkfile << "./src/areacara.cpp ";

    desrefinarMalla.lookupValue("MinLength", value);
    if (value)
        mkfile << "./src/largoarcominimo.cpp ";
}
```

```
    Setting &distribHormona = root["DistributeHormone"];
    if (distribHormona.getType() == Setting::TypeBoolean &&
       distribHormona)
    {
        mkfile << "./src/distribuirhormona.cpp ";
        mkfile << "./src/GUIDialogDistribuirHormona.cc ";
        mkfile << "./src/GUIDialogDistribuirHormona_glade.cc ";
    }


    Setting &deformarMalla = root["DeformMesh"];
    if (deformarMalla.getType() == Setting::TypeGroup)
    {
        mkfile << "./src/deformar.cpp ";
        mkfile << "./src/GUIDialogCambios.cc ";
        mkfile << "./src/GUIDialogCambios_glade.cc ";

        deformarMalla.lookupValue("LocalConsistency", value);
        if (value)
            mkfile << "./src/verificacionlocal.cpp ";

        deformarMalla.lookupValue("NeighborNodes", value);
        if (value)
            mkfile << "./src/verificacionnodosvecinos.cpp ";

        deformarMalla.lookupValue("NoVerification", value);
        if (value)
            mkfile << "./src/noverificacion.cpp ";
    }

    mkfile << endl << "OUTPUT = salida" << endl << endl;

    mkfile << "default:" << endl;
    mkfile << "\t$(CC) $(SOURCES) $(LFLAGS) $(PKG_CONFIG) -o $(EXEC)
        2> $(OUTPUT)" << endl;

    mkfile << "clean:" << endl;
    mkfile << "\trm -rf $(EXEC)" << endl;

    mkfile.close();

}

void MakefileGenerator::CopyFile()
{
    char command[50];

    strcpy(command, "cp ");
    strcat(command, MAKEFILE_NAME);
    strcat(command, " ");
```

```
    strcat(command, MG_MAKEFILE);

    system(command);
}
```

A.5. Clase ConfigReader

```
#include "configreader.h"
#define CONFIG_FILENAME "./config.cfg"

ConfigReader::ConfigReader() {
        cfg.readFile(CONFIG_FILENAME);
}

Setting & ConfigReader::getTreeGrowth() {
        Setting &root = cfg.getRoot();
        return root["TreeGrowthSimulator"];
}

Setting & ConfigReader::getGenerarMalla() {
        Setting &root = cfg.getRoot();
        return root["GenerateMesh"];
}

Setting & ConfigReader::getGuardarMalla() {
        Setting &root = cfg.getRoot();
        return root["SaveMesh"];
}

Setting & ConfigReader::getCargarMalla() {
        Setting &root = cfg.getRoot();
        return root["LoadMesh"];
}

Setting & ConfigReader::getMejorarMalla() {
        Setting &root = cfg.getRoot();
        return root["ImproveMesh"];
}

Setting & ConfigReader::getRefinarMalla() {
        Setting &root = cfg.getRoot();
        return root["RefineMesh"];
}

Setting & ConfigReader::getDesrefinarMalla() {
        Setting &root = cfg.getRoot();
        return root["DerefineMesh"];
}
```

```
Setting & ConfigReader::getDistribuirHormona() {
        Setting &root = cfg.getRoot();
        return root["DistributeHormone"];
}

Setting & ConfigReader::getDeformarMalla() {
        Setting &root = cfg.getRoot();
        return root["DeformMesh"];
}
```