Universidad de las Ciencias Informáticas

Facultad 4

Aplicación para visualizar entidades geométricas, topologías y modelos de componentes mecánicos (CAD2D)

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autor:

Angel Luis Ortega Amador

Tutores:

Dr.C Augusto Cesar Rodríguez Medina

Ing. Wendy Reyes Jiménez

La Habana, junio de 2015

"Año 57 de la Revolución"

Agradecimientos:

A mis padres y a toda mi familia por todo su apoyo durante estos cinco años.

A mi tutor Augusto, por ser el mejor tutor que se puede pedir.

A Sandy por toda su ayuda durante la tesis.

A mi cotutora Wendy y mi compañero de proyecto Abel.

A mis compañeros de grupo, del 1 y del 3.

A todos, un millón de gracias.

Declaración de autoría

Declaro que soy el único autor de este trabajo y autorizo a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los ____ días del mes de _____ del año 2015.

_____

Angel Luis Ortega Amador

Firma del Autor

_____ _____

Dr.C Augusto Cesar Rodríguez Medina

Firma del Tutor

Ing. Wendy Reyes Jiménez

Firma del tutor

"Si he visto más lejos que otros, ha sido parándome sobre hombros de gigantes"

Isaac Newton.

Resumen

El presente trabajo posee como problemática la falta de un visor de prototipos virtuales de entidades mecánicas en el marco de un proyecto productivo existente en la facultad 4 de la Universidad de las Ciencias Informáticas, y contempla como objetivo la creación del mismo, a partir de la técnica de reutilización de código presente en soluciones similares existentes.

Contenido

Introducción

El proceso de investigación y desarrollo, cuyos resultados se exponen en los capítulos de este documento, tuvo su origen en las necesidades de un proyecto de la Facultad 4 de la Universidad de las Ciencias Informáticas, cuyo propósito es el desarrollo de una aplicación para el diseño asistido por computadoras, destinado a sectores de la industria nacional vinculados a la actividad de diseño. En ese contexto, la principal tarea fue crear un visor para mostrar entidades modeladas utilizando la tecnología OpenCASCADE y el framework Qt, utilizando como tecnología de visualización Open Inventor.

El documento está estructurado en tres capítulos: el primero contiene la fundamentación teórica del proceso de investigación y desarrollo, en el que se expone el resultado del análisis realizado a los sistemas existentes, identificándose las tecnologías necesarias para obtener la solución; en el segundo se describen los principales conceptos asociados al uso de la metodología SCRUM con XP, así como la planificación del proceso y los artefactos generados; en el tercer capítulo se relaciona el proceso de ejecución de las iteraciones efectuadas para la implementación además del análisis de los resultados mediante pruebas de integración y unitarias guiadas por el principio de desarrollo establecido en la metodología XP.

1 Fundamentación teórica

Este capítulo contiene los resultados de la investigación requerida, para encontrar la solución a uno de los problemas generados en el contexto de un proyecto de investigación, destinado a obtener un sistema informático para el dibujo técnico mecánico en dos dimensiones (2D), este tipo de aplicación es ampliamente utilizada en las etapas de diseño por parte de los ingenieros en diversos perfiles. Actualmente se utilizan sistemas propietarios para estas labores, los más conocidos y difundidos en nuestro país son AutoCAD e Inventor, ambos de AutoDesk, que como muchos otros, poseen una elevada calidad, pero nuestro país no los puede adquirir a causa de las restricciones que impone el bloqueo de Estados Unidos contra Cuba.

Al profundizar en la problemática que dio origen a esta investigación, se detectaron dificultades que conducen a sostener el criterio de que es necesario sustituir el uso de los mencionados sistemas para evitar litigios por el uso no autorizado de los mismos; la lógica llevó a definir como línea estratégica el remplazo del AutoCAD por ser el más empleado en la industria nacional. En el siguiente apartado se exponen los aspectos fundamentales relacionados con el problema.

1.1 Aspectos generales relacionados con el problema

En este trabajo, el problema concreto está asociado con un requerimiento de proyecto, la necesidad de una aplicación base capaz de visualizar entidades geométricas modeladas con la tecnología OpenCASCADE.

El **objetivo general** para dar solución al problema planteado es:

Obtener una aplicación para el proyecto “Sistema para el dibujo técnico mecánico en dos dimensiones, asistido por computadora” con la capacidad de visualizar entidades geométricas modeladas con la tecnología OpenCASCADE y funcionalidades para importar y exportar información gráfica en los formatos utilizados por las aplicaciones que existen actualmente en el mercado, principalmente .step, .brep y .iges.

El **objeto de estudio** es la visualización de prototipos virtuales de objetos mecánicos y el **campo de acción**, la gráfica computacional.

Los **objetivos específicos**, derivados del general son:

- Elaborar el marco teórico conceptual que sustenta el trabajo.
- Diseñar el visor de la aplicación CAD2D.
- Implementar el visor de la aplicación CAD2D.
- Integrar los componentes “aceleradores de diseño” a la aplicación.

Las **tareas** a desarrollar para cumplir los objetivos planteados son:

- Elaboración del estado del arte de la tesis a partir de los resultados del proceso de investigación.Descripción de los diferentes aspectos que conforman la aplicación.
- Diseño de la interfaz gráfica del visor.
- Elaboración de la estructura de clases que gestionarán el visor.
- Implementación del visor de modelos geométricos de entidades mecánicas.
- Integración de los aceleradores de diseño a la aplicación.
- Comprobación de las funcionalidades del visor.

Para lograr lo anterior es preciso profundizar en el estudio de las principales tecnologías, que pueden ser utilizadas y que estén disponibles para realizar este tipo de aplicaciones, así como entender la forma de visualizar los modelos virtuales por computadora. A continuación se expone la forma convencional de dibujar en una computadora.

1.2 Proceso de visualización en 3D

“Un proceso de visualización 3D, es la unión de una serie de operaciones con el fin de realizar el despliegue de un objeto tridimensional en un dispositivo gráfico de salida. Por lo general este objeto es generado mediante una serie de datos procesados por un motor gráfico 3D” (1)

1.2.1 Frameworks y bibliotecas de diseño gráfico

El desarrollo de software para la visualización de objetos en tres dimensiones se ha diversificado a una velocidad increíble, cada vez es mayor el número de aplicaciones y bibliotecas creadas con este objetivo.

Hace unos años el mercado estaba dominado por algunos pocos productos de este tipo, pero actualmente existen múltiples opciones de frameworks y bibliotecas gráficas, tanto privativas como en el mundo del software libre, y cada una con ventajas y desventajas propias, pero la mayoría son muy prometedoras en cuanto a la cantidad de funcionalidades que poseen y las facilidades que brindan a sus usuarios, incluso, algunas de las de software libre pueden igualar o incluso superar a sus similares privativas en algunas de sus características.

Las bibliotecas gráficas son un conjunto de clases con miles de líneas de código, que generalmente poseen compatibilidad con varios tipos de hardware, permitiendo así realizar la misma función de la misma forma en diferentes tarjetas de video, estandarizando la forma de hacer una misma función, o representar una misma figura, o sea, son independientes del hardware utilizado por el ordenador, permitiendo la implementación una sola vez de funciones creadas por el programador.

1.2.2 OpenGL

OpenGL (Open Graphics Library) es una de las alternativas libres más utilizadas. Es una biblioteca multilenguaje y multiplataforma de funciones primitivas, comandos o rutinas utilizada para situar objetos en el espacio en una computadora y dibujar escenas tridimensionales complejas a partir de primitivas geométricas simples, tales como puntos, líneas y triángulos. Está orientada al desarrollo de aplicaciones interactivas, para diseñar interfaces gráficas para el manejo de 2D y 3D. Es ampliamente utilizada en realidad virtual, diseño asistido por computadora, representación científica, visualización de información y simulación de vuelo, también es muy utilizada en la producción de videojuegos. Aunque no se considera una biblioteca de alto nivel, es innegable que se ha hecho un lugar importante en la industria de las aplicaciones de gráficos por computadora. (2)

1.2.3 VTK

El kit de herramientas de visualización o *Visualization Toolkit*, en inglés, es un sistema orientado a objetos para el procesamiento y la visualización de imágenes en tres dimensiones. VTK incluye un cuadro de texto,

una biblioteca de clases en C++, y varias capas de interfaz interpretadas, incluyendo Tcl/Tk, Java y Python. VTK soporta una amplia variedad de algoritmos de visualización incluyendo el escalar, vector, tensor, textura, y métodos volumétricos. También soporta técnicas de modelado avanzado como el modelado implícito, reducción de polígonos, suavizado de mallas, cortes y contorneo, entre otros. Por otra parte, docenas de algoritmos de imágenes han sido integrados en el sistema. Esto permite la mezcla de algoritmos gráficos de imágenes 2D/3D y datos. (3)

1.2.4 Open Inventor y Coin3D

Coin3D es una biblioteca libre y de código abierto de la “Open Inventor API”. Al igual que “Open Inventor”, es una API de gráficos 3D orientada a objetos en C++ utilizada para proveer una capa de alto nivel de programación en OpenGL. Coin3D es completamente compatible con la versión de la API de Open Inventor 2.1. FreeCAD adoptó Coin3D como interfaz para desarrollar programas de Open Inventor. Está escrita en C++, pero puede ser accedida desde otros lenguajes de programación usando 'bindings' disponibles, tales como Pivy (para Python) o Nickel (para .NET). (2)

1.3 Estudio de metodologías y estándares de desarrollo de software

Atendiendo a las características del equipo de desarrollo, y habiendo hecho un análisis de las metodologías existentes se ha decidido utilizar las que serán descritas en los siguientes apartados.

1.3.1 eXtreme Programming (XP)

Es la más destacada de los procesos ágiles de desarrollo de software formulada por Kent Beck. La programación extrema se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad. Los defensores de XP consideran que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos. Creen que ser capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto

es una aproximación mejor y más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos (4)

Posee varias características, las cuales son (5)

- *Desarrollo iterativo e incremental:* pequeñas mejoras, unas tras otras.
- *Pruebas unitarias continuas*: frecuentemente repetidas y automatizadas, incluyendo pruebas de regresión. Se aconseja escribir el código de la prueba antes de la codificación.
- *Programación por parejas:* se recomienda que las tareas de desarrollo se lleven a cabo por dos personas en un mismo puesto. Se supone que la mayor calidad del código escrito de esta manera -el código es revisado y discutido mientras se escribe- es más importante que la posible pérdida de productividad inmediata.
- *Frecuente interacción* del equipo de programación con el cliente o usuario. Se recomienda que un representante del cliente trabaje junto al equipo de desarrollo.
- *Corrección* de todos los errores antes de añadir nueva funcionalidad. Hacer entregas frecuentes.
- *Refactorización* del código, es decir, reescribir ciertas partes del código para aumentar su legibilidad y facilidad de mantención pero sin modificar su comportamiento. Las pruebas han de garantizar que en la refactorización no se ha introducido ningún fallo.
- *Propiedad del código compartida:* en vez de dividir la responsabilidad en el desarrollo de cada módulo en grupos de trabajo distintos, este método promueve el que todo el personal pueda corregir y extender cualquier parte del proyecto. Las frecuentes pruebas de regresión garantizan que los posibles errores serán detectados.
- *Simplicidad en el código:* es la mejor manera de que las cosas funcionen. Cuando todo funcione se podrá añadir funcionalidad si es necesario. La programación extrema apuesta que en más sencillo hacer algo simple y tener un poco de trabajo extra para cambiarlo si se requiere, que realizar algo complicado y quizás nunca utilizarlo.

Como ventajas y desventajas se encuentran (5):

Ventajas:

- Apropiado para entornos volátiles.

- Estar preparados para el cambio, significa reducir su coste.
- Planificación más transparente para nuestros clientes, conocen las fechas de entrega de funcionalidades. Vital para su negocio.
- Permitirá definir en cada iteración cuales son los objetivos de la siguiente
- Permite tener realimentación de los usuarios muy útil.
- La presión está a lo largo de todo el proyecto y no en una entrega final.

Desventajas:

- Delimitar el alcance del proyecto con nuestro cliente.

1.3.2 SCRUM

SCRUM es un proceso en el que se aplican de manera regular un conjunto de mejores prácticas para trabajar colaborativamente, en equipo, y obtener el mejor resultado posible de un proyecto. Estas prácticas se apoyan unas a otras y su selección tiene origen en un estudio de la manera de trabajar de equipos altamente productivos (5)

Scrum es un proceso ágil y liviano que sirve para administrar y controlar el desarrollo de software. El desarrollo se realiza en forma iterativa e incremental (una iteración es un ciclo corto de construcción repetitivo). Cada ciclo o iteración termina con una pieza de software ejecutable que incorpora nueva funcionalidad. Las iteraciones en general tienen una duración entre 2 y 4 semanas. Scrum se utiliza como marco para otras prácticas de ingeniería de software como RUP o Extreme Programming (4).

Scrum se focaliza en priorizar el trabajo en función del valor que tenga para el negocio, maximizando la utilidad de lo que se construye y el retorno de inversión. Está diseñado especialmente para adaptarse a los cambios en los requerimientos, por ejemplo en un mercado de alta competitividad. Los requerimientos y las prioridades se revisan y ajustan durante el proyecto en intervalos muy cortos y regulares. De esta forma se puede adaptar en tiempo real el producto que se está construyendo a las necesidades del cliente. Se busca entregar software que realmente resuelva las necesidades, aumentando la satisfacción del cliente (4).

En Scrum se realizan entregas parciales y regulares del producto final, priorizadas por el beneficio que aportan al receptor del proyecto. Por ello, Scrum está especialmente indicado para proyectos en entornos complejos, donde se necesita obtener resultados pronto, donde los requisitos son cambiantes o poco definidos, donde la innovación, la competitividad, la flexibilidad y la productividad son fundamentales (5).

Como ventajas se encuentran (5):

- **Entrega mensual (o quincenal) de resultados** (los requisitos más prioritarios en ese momento, ya completados) lo cual proporciona las siguientes ventajas:
    - Gestión regular de las expectativas del cliente y basadas en resultados tangibles.
    - Resultados anticipados (time to market).
    - Flexibilidad y adaptación respecto a las necesidades del cliente, cambios en el mercado, etc.
    - Gestión sistemática del Retorno de Inversión (ROI).
    - Mitigación sistemática de los riesgos del proyecto.
- Productividad y calidad.
- Alineamiento entre el cliente y el equipo de desarrollo.
- Equipo motivado.

Se escogió Scrum junto a XP pues en esta, el equipo se focaliza en una única cosa: construir software de calidad. Por el otro lado, la gestión de un proyecto Scrum se focaliza en definir cuáles son las características que debe tener el producto a construir (qué construir, qué no y en qué orden) y en remover cualquier obstáculo que pudiera entorpecer la tarea del equipo de desarrollo.

1.3.3 Lenguaje Unificado UML

UML: *"…*es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software." (6).

El Lenguaje Unificado de Modelado proporciona a los desarrolladores un vocabulario que incluye tres categorías: elementos, relaciones y diagramas (6).

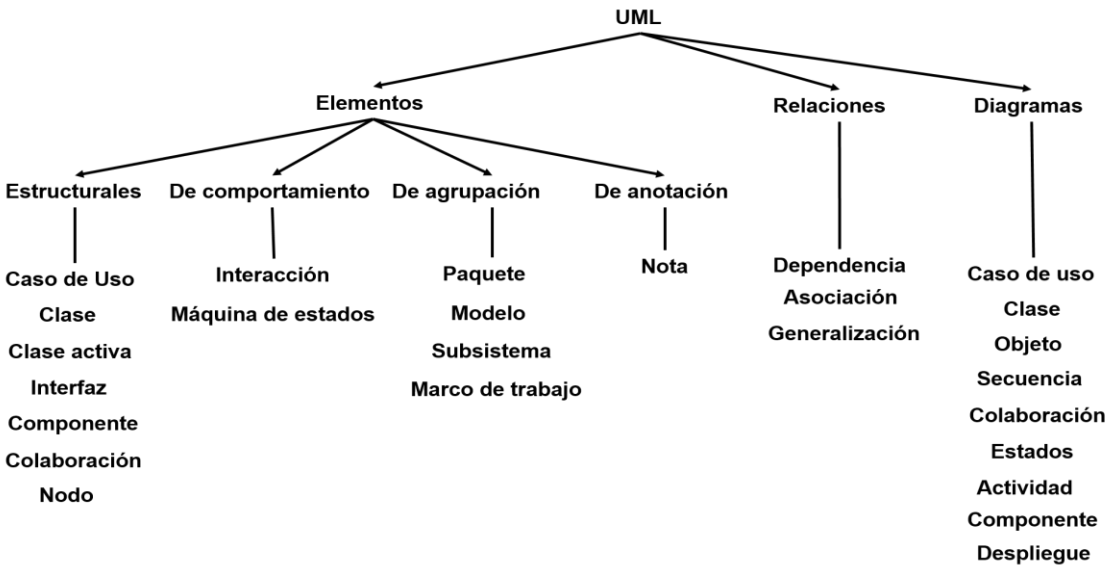


Figura 1: Vocabulario de UML *(6)***.**

UML está pensado para ser empleado en herramientas interactivas de modelado visual que tengan generadores de código y/o generadores de informes. La especificación de UML no define un proceso estándar pero está ideado para ser útil en un proceso de desarrollo iterativo. Pretende dar apoyo a la mayoría de los procesos de desarrollo orientados a objetos (6).

La capacidad de UML para modelar cualquier sistema de *software* y el hecho de que haya sido adoptado por el Grupo de Gestión de Objetos como un estándar desde Noviembre de 1997, permitieron determinarlo como el lenguaje de modelado para desarrollar la propuesta de solución (6).

Se escogió como lenguaje de modelado UML ("Unified Modeling Language") porque es posible establecer una serie de requerimientos y estructuras necesarias para modelar un sistema de software previo al proceso intensivo de escribir código. UML es un lenguaje que posee más características visuales que programáticas, las que facilitan a integrantes de un equipo participar e intercomunicarse fácilmente, siendo algunos de estos integrantes los especialistas de área y desde luego los programadores. Una de las características más importantes que hacen que se escoja UML como lenguaje de modelado es que está diseñado para uso con software orientado a objetos, y tiene un uso limitado en otro tipo de cuestiones de programación.

1.4 Patrones de diseño

Un patrón es una forma de dar solución a un problema, con un nombre y que es aplicable a otros contextos, cuenta con una sugerencia sobre la manera de usarlo en situaciones nuevas. (7)

Los patrones de diseño son el esqueleto de las soluciones a problemas comunes en el desarrollo de software. Brindan una solución ya probada y documentada a problemas que están sujetos a contextos similares. (8)

Creador

El patrón creador ayuda a identificar quién debe ser el responsable de la creación (o instanciación) de nuevos objetos o clases.

La nueva instancia deberá ser creada por la clase que (9):

- Tiene la información necesaria para realizar la creación del objeto.
- Usa directamente las instancias creadas del objeto.
- Almacena o maneja varias instancias de la clase.
- Contiene o agrega la clase.

La clase MainWindow se encarga del control general de la aplicación, creando y almacenando las instancias de la mayoría de los objetos utilizados en la aplicación.

Experto

Permite asignar una responsabilidad al experto en información, es decir, la clase que cuenta con la información necesaria para cumplir la responsabilidad (7)

La clase ViewerShape contiene las instancias de las figuras, tanto en OpenCASCADE como en Open Inventor, generando las segundas a partir de las primeras.

Controlador

La mayor parte de los sistemas reciben eventos de entrada externa. En estos casos hay que elegir controladores que manejen esos eventos de entrada. Este patrón ofrece una guía para tomar decisiones apropiadas en la elección de los controladores de eventos. Su utilización propicia que las operaciones del sistema se manejen en la capa de dominio de los objetos y no en la de presentación (7)

La clase MainWindow también hace función de controlador, al igual que la clase ViewerWidget, la cual gestiona las acciones sobre el visor.

Singleton

El patrón de diseño singleton (instancia única) está diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto.

Su intención consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella.

El patrón singleton provee una única instancia global gracias a que:

- La propia clase es responsable de crear la única instancia.
- Permite el acceso global a dicha instancia mediante un método de clase.
- Declara el constructor de clase como privado para que no sea instanciable directamente.

La clase CADApplication inicializa la aplicación, así como el motor de Open Inventor, la implementación de la misma sigue el patrón singleton, con el objetivo de tener una instancia única.

1.5 Tecnologías actuales. Selección de herramientas y lenguaje de desarrollo.

Atendiendo a las características del proyecto y las soluciones disponibles en la actualidad se han tomado las siguientes decisiones con respecto a las tecnologías que serán utilizadas.

1.5.1 Lenguaje C++

C++ es un lenguaje de programación, diseñado a mediados de los años 1980, por Bjarne Stroustrup, como extensión del lenguaje de programación C, abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos por lo que se considera un lenguaje híbrido multiparadigma. C++ es un lenguaje de programación maduro y de gran velocidad de compilación y presenta las siguientes características (10):

- Lenguaje versátil, potente y general.
- Lenguaje rico en operadores y expresiones.
- Flexible, conciso y eficiente.
- Presenta programación modular.
- Introduce nuevas palabras claves y operadores para manejo de clases.

C++ está considerado por muchos como el lenguaje más potente, debido a que permite trabajar tanto a alto como a bajo nivel. Además, se trata de un lenguaje de programación estandarizado (ISO/IEC 14882:1998), ampliamente difundido, y con una biblioteca estándar C++ que lo ha convertido en un lenguaje universal, de propósito general, y ampliamente utilizado tanto en el ámbito profesional como en el educativo. Posee una serie de propiedades difíciles de encontrar en otros lenguajes de alto nivel (10):

- Posibilidad de redefinir los operadores (sobrecarga de operadores).
- Identificación de tipos en tiempo de ejecución (RTTI).

C++ fue el lenguaje seleccionado porque además de que lo utiliza el framework seleccionado (QT), posee características que lo realzan de los demás lenguajes. Existen varias bibliotecas con algoritmos desarrollados en C++, simplemente es tomarlos y adaptarlos a la solución. También se seleccionó por las ventajas que brinda la programación orientada a objetos y por el conocimiento y la experiencia adquiridos con este lenguaje.

1.5.2 Framework Qt

Qt es un framework de desarrollo para aplicaciones multiplataforma que simplifica mucho el desarrollo de aplicaciones en C++ de forma nativa, también puede ser utilizado en otros lenguajes, funciona en las principales plataformas y tiene un amplio apoyo. Es una biblioteca para desarrollar interfaces gráficas de usuario y también para el desarrollo de programas sin interfaz gráfica como herramientas de consola y servidores (Pérez, 2011).

Este framework está compuesto por bibliotecas Qt (clases en C++), además se pueden crear formularios visualmente con Qt Designer y presenta un acceso rápido a la documentación a través de Qt Assistant. Permite una traducción con su Qt Linguist y simplifica el proceso de construcción de proyectos en las diferentes plataformas soportadas. La Interfaz de Programación de Aplicaciones (API) de la biblioteca cuenta con métodos para acceder a bases de datos mediante SQL, así como uso de XML; cuenta con una API multiplataforma unificada para la manipulación de archivos y otras para el manejo de ficheros, además de estructuras de datos tradicionales (10).

Otras Características:

- QT es una biblioteca para la creación de interfaces gráficas. Se distribuye bajo una licencia libre GPL (o QPL), además de la LGPL, que nos permite incorporar las QT en nuestras aplicaciones open-source.
- Se encuentra disponible para una gran número de plataformas: Linux, Mac OS X, Solaris, HP-UX, UNIX con X11, Windows.
- Es orientado a objetos y el lenguaje para el que se encuentra disponible es C++.
- Es una biblioteca que se basa en los conceptos de widgets (objetos), Señales-Slots y Eventos (Ej.: clic del ratón).
- Las señales y los slots es el mecanismo para que unos widgets se comuniquen con otros.
- Los widgets pueden contener cualquier número de hijos. El widget superior (también conocido como top-level), puede ser cualquiera, sea ventana, botón, etc.
- Compatibilidad multiplataforma con un sólo código fuente.
- Fácil de internacionalizar.
- Arquitectura lista para plugins.

Con todo lo anterior expuesto se puede decir que Qt dispone de tres grandes ventajas ante otras bibliotecas rivales (Pérez, 2011):

- Qt es completamente gratuito para aplicaciones de código abierto, y una de las licencias mencionadas anteriormente, la LGPL, permite que se utilice con fines comerciales sin costo alguno.
- Las herramientas, bibliotecas y clases están disponibles para casi todas las plataformas Unix y sus derivados (como Linux, Mac OS X, Solaris, etc.) como también para la familia Windows, por lo que una aplicación puede ser compilada y utilizada en cualquier plataforma sin necesidad de cambiar el código y la aplicación se verá y actuará mejor que una aplicación nativa.
- Qt tiene una extensa biblioteca con clases y herramientas para la creación de aplicaciones con interfaz de usuario enriquecidas. Estas bibliotecas y clases están bien documentadas, son muy fáciles de usar y tienen una gran herencia de programación orientada a objetos lo cual hace de la programación de interfaces gráficas un trabajo más ameno.

A continuación otras de las características más importantes de este framework que ayudan a la selección del mismo (10):

- Compatibilidad multiplataforma con un sólo código fuente.
- Rendimiento en C++.
- Disponibilidad del código fuente.
- Excelente documentación.
- Arquitectura lista para plugins.

Con este framework se pueden crear de forma rápida y sencilla interfaces gráficas de usuario; tiene disponible como lenguaje C++, siendo este el lenguaje seleccionado para la implementación del componente. También el uso de esta biblioteca ahorra una enorme cantidad de tiempo y esfuerzo, pues de lo contrario se tendrían que desarrollar algunos componentes desde cero. Estas son las principales características que debe tener la aplicación que se desarrolla, lo cual dio paso a la selección del mismo para la realización de la aplicación de este trabajo.

1.5.3 OpenCASCADE

Dentro de las variantes que han surgido para la implementación y trabajo con las tecnologías CAD/CAE/CAM en código libre, está la tecnología OpenCASCADE (Estrada, 2012).

OpenCASCADE (abreviatura de las siglas en inglés ”Computer-Aided Software for Computer-Aided Design and Engineering”), desarrollada inicialmente a principios de los años 90 por “Matra Datavision”, es una aplicación orientada a usuarios que se desenvuelven dentro del área del diseño, como ingenieros, arquitectos y diseñadores. Gracias a sus bondades es posible diseñar y modificar cualquier modelo bidimensional o tridimensional deseado (Estrada, 2012).

Cuenta con una licencia de funcionamiento GPL, lo que permite utilizarla de forma ilimitada en cualquier ordenador. Se comporta como un entorno de desarrollo integrado, orientado a la edición y el diseño de superficies y objetos bidimensionales y tridimensionales, utilizando para su diseño una gran cantidad de opciones de personalización que facilitan enormemente el modelado de estas figuras u objetos (11).

Cuenta con la capacidad de realizar análisis gráficos de cualquier superficie, lo que se traduce en estimar y simular los datos generados por las curvas de los gráficos de forma numérica y rápida, facilitando los estudios de ingeniería. Por si fuera poco, ofrece la posibilidad de importar y exportar figuras. Además viene con un paquete completo de efectos especiales y texturas que se pueden añadir a los diseños de modo que se puedan personalizar por completo (11)

1.5.4 OCAF (OpenCASCADE Application Framework)

La manera más sencilla de trabajar con OpenCASCADE es a través de su framework OCAF, que por sus siglas en inglés significa “OpenCASCADE Application Framework”. OCAF es una plataforma fácil de usar y que permite el desarrollo rápido de aplicaciones sofisticadas de diseño. Una aplicación típica desarrollada con OCAF puede tener un modelador geométrico en dos o tres dimensiones, herramientas de fabricación o de análisis, y aplicaciones de simulación o herramientas de ilustración. Cuenta con un sinnúmero de características que la convierten en una opción atractiva a la hora de hacer una selección de tecnología para el trabajo con CAD/CAE/CAM, y que justifican su selección en este caso, como son (11):

- Facilidad de diseño de la arquitectura de la aplicación.
- Definición de los componentes y la forma en que cooperan.

- Definición del modelo de datos capaz de soportar las funcionalidades requeridas.
- Estructuración del software en relación a:
    - Sincronizar la visualización con los datos. Los comandos de visualización de objetos tienen que actualizar la vista una vez que estos son llamados.
    - Soportar comandos para las operaciones “deshacer” y “rehacer” (Ctrl+Z). Esta función debe ser tomada en cuenta muy tempranamente en el proceso de diseño, y hecho así, funciona muy eficazmente.
- Permite la implementación de las funciones para salvar los datos. Si la aplicación tiene un ciclo de vida muy largo, la compatibilidad de los datos entre las versiones tiene que ser tratada.
- Facilidad para crear una interfaz de usuario.

Gracias a tener implementadas todas estas funciones, permite desarrollar aplicaciones en buen tiempo, permitiendo que a la hora de desarrollar una aplicación, solo sea necesario concentrarse en las funcionalidades específicas, y no en la esencia del código que se maneja, factor decisivo para su selección final.

1.5.5 Gestor de documentación Doxygen

Doxygen es un generador de documentación para C++, C, Java, Python, entre muchos otros. Funciona en Linux, Windows y Mac OS X. El mismo es utilizado para generar la documentación de los métodos o clases que puedan surgir a partir del desarrollo de la aplicación. Genera un buscador de documentación en línea y/o un manual de referencia de un conjunto de archivos fuente documentados. También tiene soporte para archivos HTML, PDF con hipervínculos, archivos RTF de Word y otros. La documentación es extraída directamente del código fuente, lo cual hace mucho más fácil mantener la documentación actualizada. Doxygen es desarrollada bajo Mac OS X y Linux, pero está hecho para ser altamente portátil.

1.5.6 Contador de líneas de código CCCC

En el marco de la aplicación se pretende seguir con especial atención la cantidad de líneas de código procesadas, por lo que se hace necesario una aplicación capaz de automatizar este trabajo. C and C++ Code Counter (CCCC por sus siglas en inglés) fue desarrollado como una base de prueba para varias ideas sobre métricas, siguiendo la metodología de cantidad de líneas de código escritas, la cual es similar a la que se pretende utilizar en nuestro proyecto para llevar un seguimiento del trabajo realizado por los desarrolladores.

1.6 Aplicaciones existentes para el diseño asistido por computadora

Actualmente en el mundo existen varias herramientas para el diseño asistido por computadora, el dominio lo tienen los de carácter comercial, que además son de código cerrado; el desarrollo alcanzado por ellas es notable y constituyen una importante y productiva herramienta en manos de ingenieros y diseñadores. El gran problema para nuestro país es la falta de acceso a estas tecnologías a causa del bloqueo, pero aun cuando este no exista, los costos por licencia de operación serían difíciles de asumir.

La única alternativa a esta situación de monopolio está en herramientas de código abierto con funcionalidades que pueden ser aprovechadas para desarrollar sistemas para el diseño asistido por computadora; se trata de sistemas en desarrollo no recomendados para su uso profesional, al menos en su estado actual.

En los próximos apartados se expone información general de estas herramientas, seis de las propietarias y dos de código abierto.

1.6.1 CATIA

Catia (Computer-Aided Three Dimensional Interactive Application) es un programa informático de diseño, fabricación e ingeniería asistida por computadora comercial realizado por Dassault Systèmes. El programa está desarrollado para proporcionar apoyo desde la concepción del diseño hasta la producción y el análisis

de productos. Está disponible para Microsoft Windows, Solaris, IRIX y HP-UX. Provee una arquitectura abierta para el desarrollo de aplicaciones o para personalizar el programa. Las interfaces de programación de aplicaciones, CAA2 (o CAAV5), se pueden programar en Visual Basic y C++. Fue inicialmente desarrollado para servir en la industria aeronáutica, se ha hecho un gran hincapié en el manejo de superficies complejas (12).

CATIA también es ampliamente utilizado en la industria del automóvil para el diseño y desarrollo de componentes de carrocería. Concretamente empresas como el Grupo VW (Volkswagen, Audi, SEAT y Škoda), BMW, Renault, Peugeot, Daimler AG, Chrysler, Smart y Porsche hacen un amplio uso del programa. La industria de la construcción también ha incorporado el uso del software para desarrollar edificios de gran complejidad formal; el Museo Guggenheim Bilbao, en España, es un hito arquitectónico que ejemplifica el uso de esta tecnología (12).

1.6.2 ArchiCAD

ArchiCAD desarrollado por la empresa húngara Graphisoft que comenzó en 1982 originalmente para Macintosh. Es un software DAC de modelado de información de construcción (BIM, Building Information Modeling) disponible para sistemas operativos Macintosh y Windows. BIM es un paradigma del dibujo asistido por computadora que permite un diseño basado en objetos inteligentes y en tercera dimensión. De este modo facilita el trabajo en tres dimensiones y dos dimensiones a la vez, pudiendo ahorrar tiempos en presentaciones y etapas del proyecto arquitectónico y permitiendo actualizar automáticamente toda la documentación de manera instantánea, sin la intervención del usuario para cambiar manualmente todas las vistas, planos y metrados (13).

Fundamentalmente ArchiCAD, es un software por el cual se puede construir edificaciones, inicialmente en un ámbito de dos dimensiones, basándose en parámetros básicos, tales como altura, largo, espesor y elevación, en el caso de muros, altura, ancho, largo y elevación en el caso de objetos, además el uso de este conlleva un conocimiento básico de términos usados en la arquitectura, puesto que los ámbitos más específicos (puertas, ventanas, escaleras, forjados y techos) se usan parámetros tales como: alfeizar, telar, paso, contrapaso, entre otros. La vista en plantas es la principal diferencia operacional respecto a otros programas de DAC analíticos como AutoCAD o Microstation, no es tomada desde una cámara en el infinito,

sino que se radica en un corte paralelo al horizonte, este mismo tiene su inicio en una determinada altura y finaliza donde se ubica el forjado o base de piso (13).

ArchiCAD permite trabajar al usuario con representaciones dos dimensiones o tres dimensiones en pantalla. Los diseños en d*os* dimensiones pueden ser exportados en cualquier momento, incluso en el modelo; la base de datos siempre almacena los datos en tres dimensiones. Planos, alzados y secciones son generados desde el modelo del edificio virtual de tres dimensiones y son constantemente actualizados. Los diseños detallados están basados en porciones alargadas del modelo, con detalles en 2D añadidos (13).

Desarrolladores externos y algunos fabricantes para arquitectura han desarrollado bibliotecas de componentes arquitectónicos para usar en ArchiCAD, gracias a que el programa incluye un lenguaje de descripción geométrica (GDL) usado para crear nuevos componentes. La última apuesta de la compañía se centra en este aspecto, poniendo a disposición de los usuarios una web que funcionará como almacén online de objetos. Esto permite el intercambio de objetos desarrollados por distintos usuarios, así como por empresas de mobiliario, que se encarguen de elaborar versiones virtuales de sus productos (13).

1.6.3 AutoDesk Inventor

Autodesk Inventor es un paquete que se basa en técnicas de modelado paramétrico de sólidos en tres dimensiones producido por la empresa de software Autodesk. Compite con otros programas de diseño asistido por computadora como SolidWorks, CATIA y Solid Edge. Entró en el mercado en 1999, muchos años después que los antes mencionados y se agregó a las Series de Diseño Mecánico de Autodesk como una respuesta de la empresa a la creciente migración de su base de clientes de diseño mecánico en dos dimensiones hacia la competencia, permitiendo que los computadoras personales ordinarias puedan construir y probar montajes de modelos extensos y complejos (14).

Los usuarios comienzan diseñando piezas que se pueden combinar en ensamblajes, y corrigiendo las mismas, pueden obtenerse diversas variantes. Como modelador paramétrico, no debe ser confundido con los programas tradicionales de DAC; Inventor se utiliza en diseño de ingeniería para producir y perfeccionar productos nuevos, mientras que en programas como AutoCAD se conducen solo las dimensiones. Un modelador paramétrico permite modelar la geometría, dimensión y material de manera que si se alteran las dimensiones, la geometría se actualiza automáticamente basándose en las nuevas dimensiones. Esto

permite que el diseñador almacene sus conocimientos de cálculo dentro del modelo, a diferencia del modelado no paramétrico, que está más relacionado con un “tablero de bocetos digitales”. Inventor también tiene herramientas para la creación de piezas metálicas (14).

Los bloques de construcción cruciales de Inventor son las piezas, se crean definiendo las características, que a su vez se basan en bocetos (dibujos en dos dimensiones). Este sistema de modelado es mucho más intuitivo que en ambientes antiguos de modelado, en los que para cambiar dimensiones básicas era necesario generalmente suprimir el archivo entero y comenzar de cero (14).

Como parte final del proceso, las partes se conectan para hacer ensamblajes, los ensamblajes pueden consistir en piezas u otros ensamblajes. Las piezas son ensambladas agregando restricciones entre las superficies, bordes, planos, puntos y ejes. Por ejemplo, si uno coloca un piñón sobre un eje, una restricción insertada podría agregarse al eje y el piñón, haciendo que el centro del eje sea el centro del piñón. La distancia entre la superficie del piñón y del extremo del eje se puede también especificar con la restricción insertada. Otras restricciones incluyen Coincidencia, Nivelación, inserción, ángulo, tangente, transicional, movimiento, sistema de coordenadas de usuario (14).

Las últimas versiones de Inventor incluyen funcionalidades que poseían muchos modeladores de tres dimensiones de mediano y alto nivel, utiliza el Gestor de Formas (Shape Manager) como su kernel de modelaje geométrico, el cual pertenece a Autodesk y fue derivado del kernel de modelaje ACIS; además, incluye, en la versión profesional, las herramientas necesarias para crear piezas de plástico y sus respectivos moldes de inyección. Cuenta también con análisis de tensiones por elementos finitos y análisis dinámicos, creación y análisis de estructuras, piping y cableado. Su combinación con Autodesk Vault y Autodesk 360 la hacen líder en el mercado del diseño mecánico (14).

1.6.4 SolidWorks

SolidWorks es un software DAC para modelado mecánico en tres dimensiones, desarrollado en la actualidad por SolidWorks Corp., una filial de Dassault Systèmes, S.A. (Suresnes, Francia), para el sistema operativo Microsoft Windows. Su primera versión fue lanzada al mercado en 1995 con el propósito de hacer la tecnología DAC más accesible (15).

El programa permite modelar piezas y conjuntos y extraer de ellos tanto planos técnicos como otro tipo de información necesaria para la producción. Es un programa que funciona con base en las nuevas técnicas de modelado con sistemas DAC. El proceso consiste en trasvasar la idea mental del diseñador al sistema DAC, "construyendo virtualmente" la pieza o conjunto. Posteriormente todas las extracciones (planos y ficheros de intercambio) se realizan de manera bastante automatizada (15).

1.6.5 Solid Edge

Solid Edge es un programa parametrizado de diseño asistido por computadora de piezas tridimensionales, Presentado en 1996, inicialmente fue desarrollado por Intergraph como uno de los primeros entornos basados en DAC para Windows NT, ahora pertenece y es desarrollado por Siemens AG. Su kernel de modelado geométrico era originalmente ACIS, pero fue cambiado a Parasolid, el núcleo Parasolid es desarrollado actualmente por Siemens PLM software y es usado ampliamente como el motor geométrico de otras herramientas DAC (SolidWorks, IronCAD y MoldFlow). Recientemente adquirido por Siemens AG está empezando a formar parte de todas sus plantas de producción e ingeniería. Solid Edge permite el modelado de piezas de distintos materiales, doblado de chapas, ensamblaje de conjuntos, soldadura, funciones de dibujo en plano para ingenieros (16).

Este es uno de los paquetes instados a enterrar el uso masivo del DAC 2D dando paso al DAC 3D, con las consiguientes ventajas a todos los niveles del trabajo. A través de software de terceras partes, es compatible con otras tecnologías PLM. También trae "Insight", escrito en PDM y con funcionalidades CPD basadas en tecnología Microsoft (16).

1.6.6 AutoCAD de AutoDesk

AutoCAD es un software DAC utilizado para dibujo en dos dimensiones y modelado en tres dimensiones. Actualmente es desarrollado y comercializado por la empresa Autodesk (17).

El nombre AutoCAD surge como creación de la compañía Autodesk, en que Auto hace referencia a la empresa creadora del software y CAD a Diseño Asistido por Computadora (por sus siglas en inglés

"*Computer-Aided Design*"), teniendo su primera aparición en 1982. AutoCAD es un software reconocido a nivel internacional por sus amplias capacidades de edición, que hacen posible el dibujo digital de planos de edificios o la recreación de imágenes en tres dimensiones; es uno de los programas más usados por arquitectos, ingenieros, diseñadores industriales y otros (17).

1.6.7 FreeCAD

FreeCAD es una aplicación de código abierto para el diseño asistido por computadora en tres dimensiones, ingeniería asistida por computadora, para la asistencia en ingeniería mecánica y el diseño de elementos mecánicos. Está basado en OpenCASCADE y programado en los lenguajes C++ y Python (18).

FreeCAD presenta un entorno de trabajo similar a CATIA, SolidWorks, Solid Edge, ArchiCAD o Autodesk Revit. Utiliza técnicas de modelado paramétrico y está provisto de una arquitectura de software modular, pudiendo añadir de forma sencilla funcionalidades sin tener que cambiar el núcleo del sistema (18).

A diferencia de los sistemas DAC analíticos tradicionales, como pueden ser AutoCAD o Microstation, FreeCAD se basa en un croquizador paramétrico con características similares a los de las aplicaciones comerciales modernas. En el diseño paramétrico cada elemento del dibujo (muros, puertas, ventanas, etc.) es tratado como un objeto, el cual no es definido únicamente por sus coordenadas espaciales (x, y, z), sino también por sus parámetros, ya sean estos gráficos o funcionales. Las bases de datos relacionadas con el objeto hacen que este software, y especialmente su banco de trabajo de arquitectura, esté muy relacionado con el enfoque BIM, en el que un modelo BIM contiene el ciclo de vida completo de la construcción, desde el concepto hasta la edificación (18).

Como muchos modeladores 3D actuales, tiene un componente para dos dimensiones para extraer un diseño detallado de un modelo en tres dimensiones y con ello producir dibujos en dos dimensiones, pero el diseño directo en dos dimensiones (como el de AutoCAD LT) no es la meta, ni tampoco la animación ni formas orgánicas (como las creadas por Maya, 3ds Max o Cinema 4D). FreeCAD posee elementos que se reutilizarán, tales como la poli línea, el arco de circunferencia, el círculo, el punto y la recta, además de módulos de parametrización y de mayado de entidades.

1.6.8 LibreCAD

LibreCAD es una aplicación informática de código libre de diseño asistido por computadora (DAC) para diseño en dos dimensiones. Funciona en los sistemas operativos GNU/Linux, Mac OS X, Solaris y Microsoft Windows (19).

LibreCAD fue desarrollado a partir de un fork de *QCad Community Edition*. El desarrollo de LibreCAD está basado en las bibliotecas Qt4, pudiendo ser ejecutado en varias plataformas de manera idéntica (19).

Buena parte de la interfaz y de los conceptos sobre su uso son similares a los de AutoCAD, haciendo el uso de este más cómodo para usuarios con experiencia en ese tipo de programas DAC comerciales (19).

LibreCAD utiliza el formato del archivo de AutoCAD DXF internamente y para guardar e importar archivos, así como permite la exportación de estos en varios formatos. Al igual que en el software de diseño asistido por computadora FreeCAD, el LibreCAD posee funcionalidades como la poli línea, la recta, el circulo y el punto, las cuales han sido reutilizadas en el Proyecto “Sistema CAD 2D”.

1.7 Conclusiones del capítulo

En el presente capítulo se presentó la fundamentación teórica que sustenta este trabajo de investigación. En el mismo se abordaron temas como la necesidad de su realización, un estudio de las tecnologías a utilizar. También se abordaron las decisiones tomadas en el marco del proyecto respecto a la metodología y posible arquitectura de la aplicación, así como un estudio de las principales soluciones de su tipo existentes, con el objetivo de tomar características y funcionalidades de estas, o incluso segmentos de código de las que su licencia así lo permita.

2 Descripción de la solución

Este capítulo contiene la descripción de la solución y los elementos fundamentales de la planificación y el diseño; se abordan además, las ideas obtenidas durante el estudio del código fuente de FreeCAD en cuanto a su arquitectura

2.1 Arquitectura a utilizar

En el marco de proyecto al que pertenece la aplicación se definió como arquitectura a utilizar la Arquitectura por capas, debido al flujo de datos, en el cual una capa le envía datos a la superior para su procesamiento.

2.2 Estructura de FreeCAD

FreeCAD es la integración de varias bibliotecas: OpenCASCADE, que es utilizada para la construcción y manejo de entidades geométricas, es decir, todo lo que abarca modelación; Coin3D para visualizar las mencionadas entidades y Qt para el trabajo asociado a las interfaces de usuario (diseño e implementación de las mismas).

Las entidades geométricas que se muestran en el visor de FreeCAD son renderizadas con la biblioteca Coin3D. La aplicación OpenCASCADE también ofrece esta funcionalidad, pero los desarrolladores de FreeCAD decidieron no utilizar su visor, ya que consideraron que el de Coin3D tiene un mejor rendimiento, no obstante existen aplicaciones como Salome MECA que utiliza dos tipos de visores, el de OpenCASCADE y el de VTK.

Open Inventor es un lenguaje de descripción de escenas 3D, las cuales son renderizadas en la pantalla con OpenGL, de esto se encarga Coin3D, así el programador no necesita lidiar con llamadas complejas de OpenGL, solo tiene que proveer un código Open Inventor válido.

Una de las funcionalidades que posee FreeCAD es traducir información de geometrías de OpenCASCADE a Open Inventor. (11)

FreeCAD cuenta con 1228 archivos de código fuente (.cpp), 1253 archivos cabecera (.h) y 156 archivos de interfaz de usuario (.ui). Distribuidos en las siguientes carpetas, de las cuales serán explicadas las más

Fig 1. Estructura de carpetas de FreeCAD.

importantes:

- App:
  Clases principales de la aplicación, de integración con las otras bibliotecas, documentos, comandos de la consola, etc. Principalmente utilizada en el modo Consola del FreeCAD.
- Base:
  Clases base de FreeCAD, tales como excepciones, consola, archivos, etc.
- Gui:
  Parecido a App, pero con interfaz gráfica, esta carpeta tiene todas las clases que intervienen en la interfaz gráfica, incluyendo los archivos de interfaz gráfica (*.ui) y la integración con Quarter, y la modelación de entidades en FreeCAD.
- Main:
  Posee las clases que son ejecutadas, que se encargan de inicializar la aplicación y comprobar que todas las bibliotecas necesitadas por FreeCAD para ejecutarse están presentes en el equipo.
- Mod:

Carpeta que contiene los módulos del FreeCAD, tales como el Sketcher, Part Design, etc.

Entre las clases analizadas para la aplicación se encuentran las de la biblioteca Quarter, ubicadas en la carpeta /src/Gui/Quarter. Quarter es una biblioteca ligera que provee integración entre el visor 3D anteriormente mencionado y el IDE Qt. Con esta, se crea un objeto de tipo QuarterWidget, subclase de QGLWidget que consiste en un visor de Coin que se utiliza como área de trabajo.

La clase /src/App/Application.cpp, encargada de inicializar la aplicación e Inventor.

La clase /src/App/Document.cpp, encargada de gestionar los archivos abiertos en el FreeCAD.

En la carpeta Mod/Part se encuentran las clases encargadas de convertir las figuras creadas con OpenCASADE a Open Inventor, las cuales son mostradas en el visor.

2.3 Requerimientos del componente

Un requerimiento de software constituye una restricción que se determina con alta precisión, y debe ser satisfecha por el software final. Existen dos tipos de requisitos de software, los funcionales y los no funcionales.

Los aspectos funcionales permiten expresar una especificación detallada de las responsabilidades del sistema que se propone, además determinan de una manera clara, el comportamiento del mismo.

Las características del sistema permiten definir las cualidades que el software debe tener con el objetivo de brindar a los usuarios una mayor usabilidad, disponibilidad y rendimiento.

2.3.1 Aspectos funcionales

1: El sistema debe tener la capacidad de cargar archivos de proyecto realizados en otras aplicaciones DAC, tales como Inventor o AUTOCAD de Autodesk, o la misma FreeCAD.

2: El visor es capaz de interpretar los archivos importados y visualizar correctamente las entidades modeladas en los mismos.

3: El sistema es capaz de utilizar módulos realizados utilizando las tecnologías: OpenCASCADE y Open Inventor, e integrarse con estos satisfactoriamente.

2.3.2 Características del sistema

-Software

1: El sistema puede ser ejecutado en cualquier distribución de GNU/Linux basada en Ubuntu.

2: Tener instaladas en el equipo las bibliotecas: Oce (“OpenCASCADE Community Edition”), boost, Open Inventor, Coin3D.

-Hardware

3: El sistema necesita una tarjeta de acelerador gráfico compatible con OpenGL 1.1 o superior.

4: La estación de trabajo deberá tener como mínimo 1 GB dedicado a video.

-Interfaz

5: La interfaz de usuario debe ser lo más sencilla posible, siguiendo pautas puestas por otras aplicaciones DAC conocidas, con el objetivo de amortiguar la resistencia al cambio de los futuros usuarios del software, preferentemente pareciéndose a la del AutoCAD de AutoDesk.

-Usabilidad

6: Se pueden realizar todas las operaciones de trazado utilizando solo el mouse y el teclado, sin la necesidad de digitalizadores.

7: El sistema podrá ser utilizado por cualquier persona que posea conocimientos básicos de mecánica.

8: Se podrá cancelar la operación desde la interfaz principal.

2.4 Planificación

La planificación se basó en un análisis detallado del trabajo a realizar y su descomposición en tareas. Parte por tanto de un proyecto de obra, o de unos requisitos detallados de lo que se quiere hacer. Sobre esa información se desarrolla un plan adecuado a los recursos y tiempos disponibles; y durante la construcción se sigue de cerca la ejecución para detectar posibles desviaciones y tomar medidas para mantener el plan, o determinar qué cambios va a experimentar. Se trata por tanto de una gestión “predictiva”, que vaticina a través del plan inicial cuáles van a ser la secuencia de operaciones de todo el proyecto, su coste y tiempos. Su principal objetivo es conseguir que el producto final se obtenga según lo “previsto”; y basa el éxito del proyecto en los tres puntos apuntados: agendas, costes y calidad.

2.5 Diseño

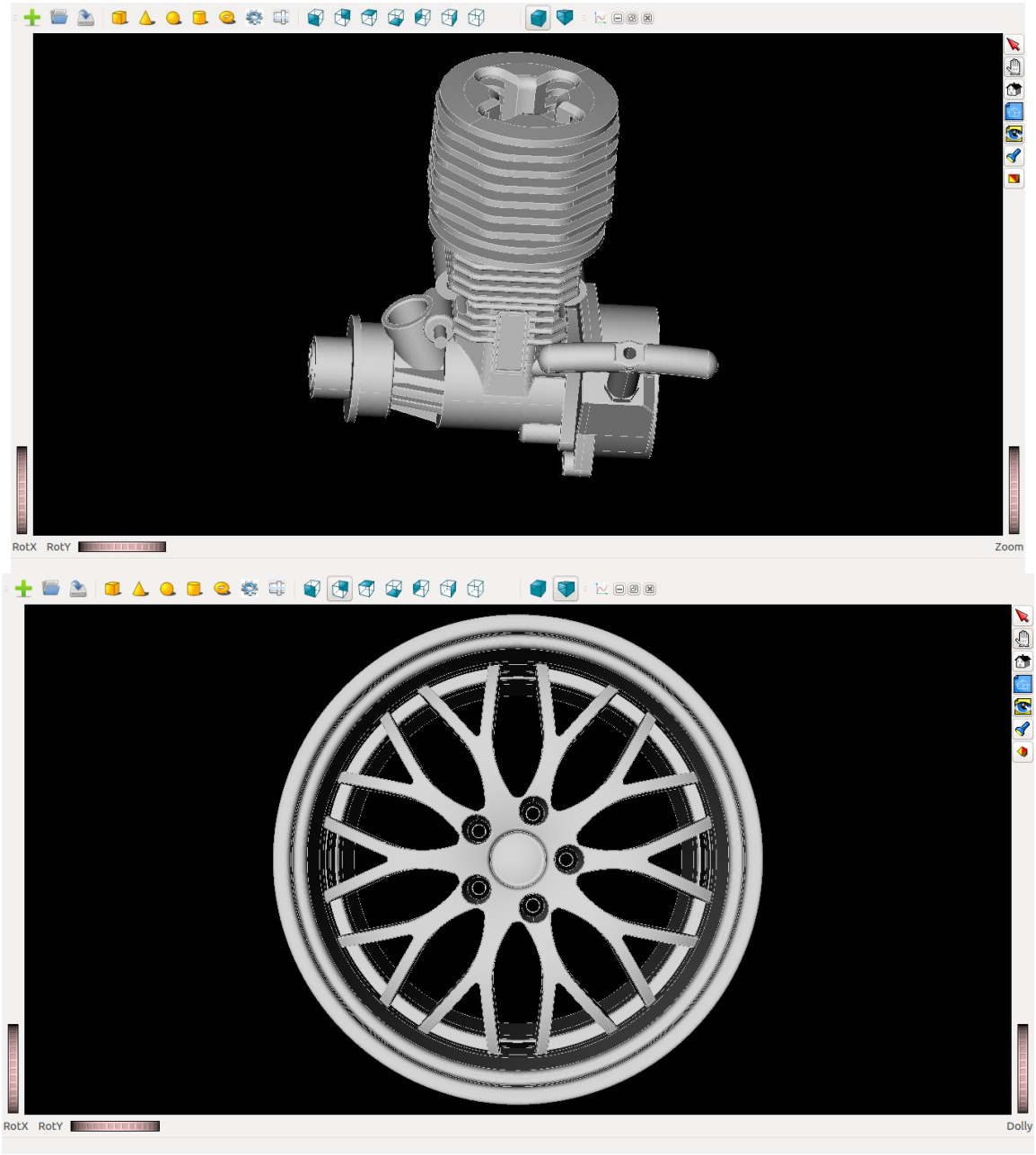
El diseño es el centro de atención al final de la fase de elaboración y el comienzo de las iteraciones de construcción. Esto contribuye a una arquitectura estable y sólida. En el diseño modelamos el sistema y encontramos su forma para que soporte todos los requerimientos, incluyendo los no funcionales y las restricciones que se le suponen.

Una entrada esencial en el diseño es el resultado de la planificación, que proporciona una comprensión detallada de los requisitos. Además, impone una estructura del sistema que debemos esforzarnos por conservar lo más fielmente posible cuando demos forma al componente. (20)

El diseño crea una entrada apropiada y un punto de partida para actividades de implementación, y se es capaz de descomponer dichas actividades en partes más manejables, que pueden ser llevadas a cabo por diferentes equipos de desarrollo, teniendo en cuenta la posible concurrencia.

2.5.1 Interfaz de usuario de la aplicación

La interfaz de usuario no es más que la parte de la aplicación con la que interactúa el usuario, las ventanas, menús, cuadros de diálogo, todo lo que el usuario ve en la aplicación. A continuación se muestra la interfaz de usuario de la aplicación.



2.6 Conclusiones del capítulo

En el presente capítulo se expusieron aspectos propios de las etapas de Diseño e Implementación de la aplicación, tales como los principales requisitos, tanto funcionales como no funcionales, con que debe contar la aplicación para garantizar el logro de los objetivos que se desean cumplir con el desarrollo de la misma. A partir de estos requisitos se planificaron las tareas de desarrollo correspondientes a cada uno de ellos. También se presentaron prototipos de la interfaz de usuario con que contará la aplicación, dando paso a la fase de Implementación de la solución.

3 Discusión de los resultados

A partir de las pautas establecidas en los capítulos anteriores, se procedió a la implementación de una solución con las características previamente mencionadas, con el objetivo de resolver el problema planteado.

3.1 Implementación

Después de haber concluido la fase de diseño, en el ciclo de vida de la aplicación viene la fase de implementación, en la cual se materializan las ideas y procedimientos descritos anteriormente. En esta fase se estableció el estándar de codificación por el que se ha regido el proyecto siguiendo las normas establecidas.

3.1.1 Estándares de codificación

Un estándar de codificación comprende todos los aspectos de la generación de código. Un código fuente completo debe quedar como si un único programador hubiera escrito todo el código de una sola vez. Al inicio de un desarrollo de software, es necesario establecer un estándar de codificación para asegurarse de que todos los programadores del proyecto trabajen de forma coordinada. Para un programador comprender bien un sistema de software, influye directamente la legibilidad del código fuente. La mantenibilidad del código es la facilidad con que el sistema de software puede modificarse para añadirle nuevas características, modificar las ya existentes, depurar errores, o mejorar el rendimiento (21).

- **Definición de Objetos, Clases, funciones y atributos**
  - ✓ Todos los nombres de las clases implementadas comenzarán con letra mayúscula. En caso de poseer un nombre compuesto se escribirán de acuerdo a la normativa CamelCase-UpperCamelCase.

Ejemplo:

```
class MakeSpurGear
{
  ...
};
```

- ✓ Siempre se declara para todas las clases implementadas su respectivo destructor de clase.

Ejemplo:

```
~QoccViewerContext();
```

- ✓ La declaración de funciones o métodos siempre comenzara en letra inicial minúscula. En caso de ser un nombre compuesto se regirá por la normativa CamelCase-lowerCamelCase.

Ejemplo:

```
vector<TopoDS_Shape> makePinnon( ...);
```

- ✓ Los atributos siempre estarán escritos con letra minúscula. En caso de ser un nombre compuesto se regirá por la normativa CamelCase-lowerCamelCase.

Ejemplo:

```
double alpha = 0.0;
```

- ➢ ***Definición de parámetros dentro de las funciones y constructores de clases***
  - ✓ Los nombres de los identificadores de los parámetros en las funciones deben estar escritos con minúscula separados a un espacio cada uno y en caso de ser compuesto utilizar normativa CamelCase-lowerCamelCase.

Ejemplo:

```
vector<TopoDS_Shape> MakeSpurGear::makePinnon(double a, double m, double z1, ...)
{
  ...
}
```

- ✓ Los identificadores de los parámetros dentro de los constructores de las clases deben estar escritos con minúscula separados a un espacio cada uno y en caso de ser compuesto utilizar normativa CamelCase-lowerCamelCase.

Ejemplo:

```
MakeSpurGear::MakeSpurGear(Handle_AIS_InteractiveContext theContext)
{
    context=theContext;
}
```

- ➢ **Definición de expresiones**
  - ✓ Para una mejor comprensión en la lectura y legibilidad del código los operadores binarios exceptuando los punteros, función de llamado a miembros, escritura de un arreglo y paréntesis de una función se escribirán con un espacio entre ellos.

Ejemplo:

```
da = d + (2.0 * m);
```

```
QString texto1 = QTime::currentTime().toString("h:mm:ss");
ui->textBrowser->setText(texto1);
```

- ➢ **Definición de estructuras de control y bucles**
  - ✓ Las estructuras de control y los bucles estarán definidos de igual manera en ambos casos siguiendo el estándar determinado por el framework de Qt.

Ejemplo:

```
while(!found && i < puntosInvoluta.size())
{
  ...
}
```

```
if(i == 1)
{
  ...
}
```

- ➢ **Comentarios en el código según el estándar de C++**
  - ✓ Comentarios pequeños.

Ejemplo:

```
/*fin de mirror*/
```

- ✓ Otros comentarios.

Ejemplo:

```
/*
 *FIN DE PRUEBA
 */
```

- ✓ Comentario de versión, descripción de clase y otras características de la clase o paquete.

Ejemplo:

```
/*
 ****************************************
 *Fin de Modelacion de dientes Gear1      *
 ****************************************
*/
```

3.1.2 Pila de Sprint o “sprint backlog”

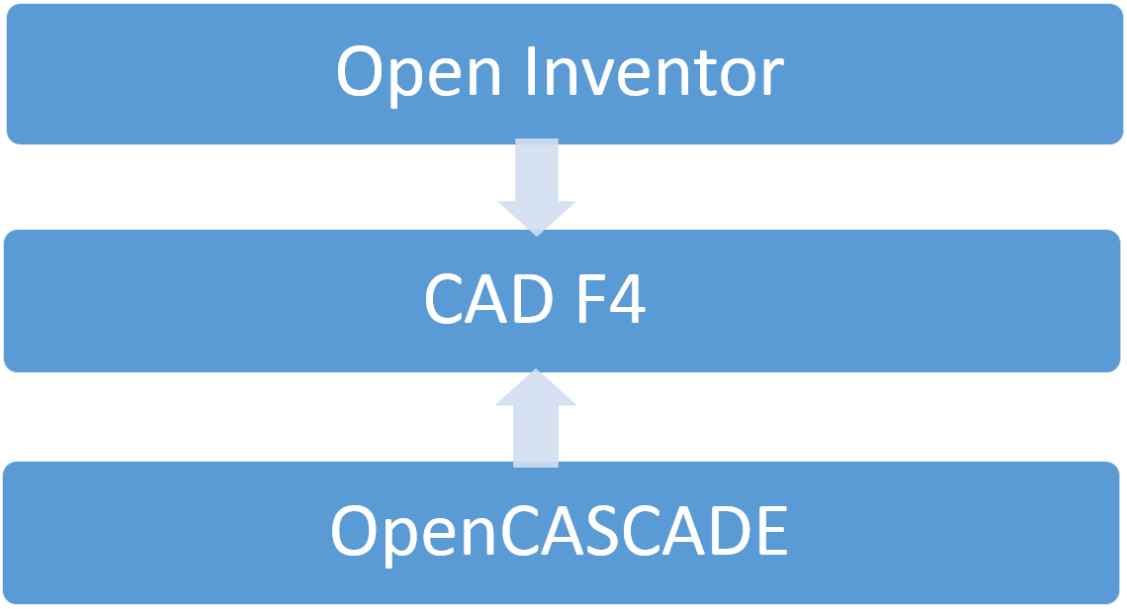
Es una lista que descompone las funcionalidades de la pila del producto en las tareas que debe realizar el equipo para construir un incremento: una parte completa y operativa del producto. Cada tarea de la pila de sprint tiene asignada una persona, y la indicación del tiempo que aún falta para terminarla. Durante el “sprint” el equipo actualiza a diario en la pila del sprint los tiempos pendientes de cada tarea. Es útil porque descompone el proyecto en unidades de tamaño adecuado para determinar el avance a diario, e identificar riesgos y problemas sin necesidad de procesos complejos de gestión.

Sprint	Inicio	Duración (días)	Backlog
3	29-03-2015	6	3

ID	Tarea	Delegado	Estado
1	Cargar archivos con datos de entidades exportados en otras aplicaciones.	Angel Luis Ortega Amador	Completado
2	Interpretar archivos importados y visualizar los datos correctamente	Angel Luis Ortega Amador	Completado
3	Integrar la aplicación con módulos realizados en OpenCASCADE.	Angel Luis Ortega Amador	Completado

3.1.3 Integración entre OpenCASCADE y Open Inventor

Open Inventor y OpenCASCADE son dos tecnologías diferentes, ambas no se reconocen entre sí, por lo que fueron creadas varias clases con el objetivo de relacionarlas. Estas también cumplen con la tarea de asegurar la independencia de la aplicación de ambas tecnologías, pues en caso de verse posteriormente en la necesidad de remplazar alguna de estas, solo habría que re implementar estas clases, evitando tener que realizar modificaciones mayores, tanto en las interfaces de usuario como en la sección de control de la aplicación.



A continuación se realiza una explicación de estas clases.

Shape:

Contiene como atributos una figura de OpenCASCADE y un nombre para la misma, el cual será un indicativo para la correspondiente figura, en el árbol que representa las figuras con que se cuenta en el visor.

ViewerShape:

Contiene un Shape, esta clase es la encargada de convertir el TopoDS_Shape a una figura legible por el visor de Open Inventor. Este proceso se logra a través de la técnica de triangulación. El proceso se basa en generar la figura de Open Inventor a partir de pequeños triángulos para modelar las partes curvas de la misma, y vértices y ejes para las regiones planas. Esta es una de las técnicas más utilizadas actualmente en la visualización de gráficos avanzados por computadora.

OIVViewer:

Esta clase hereda directamente de la clase de Open Inventor SoQtExaminerViewer, la cual es un visor, con sus funcionalidades comunes, tales como la rotación, ajustar la cámara para ver todos los elementos de la escena, etc.

ViewerWidget:

El *widget* que contiene el OIVVIewer, es el encargado de controlar el funcionamiento del visor. Lleva el registro de las figuras en el mismo, así como los métodos para los diferentes ángulos de la cámara. También se encarga de establecer los tipos de visualización: Perspectiva y Ortográfica, y posee la funcionalidad de repintar todos los objetos.

3.2 Pruebas de software

Las pruebas de *software* son un elemento crítico para la garantía de calidad del *software* y representa una revisión final de las especificaciones, del diseño y de la codificación. Estas constituyen un conjunto de herramientas, técnicas y métodos que evalúan el desempeño de un programa. Estas involucran las operaciones del sistema, evaluando los resultados bajo condiciones controladas, lo que hace que la realización de pruebas al software sea un factor de vital importancia. Las pruebas de software son las investigaciones técnicas cuyo objetivo es proporcionar información objetiva e independiente sobre la calidad del producto a la parte interesada. Son una actividad más en el proceso de control de calidad.

Las pruebas son básicamente un conjunto de actividades dentro del desarrollo de software. Dependiendo del tipo de pruebas, estas actividades podrán ser implementadas en cualquier momento de dicho proceso de desarrollo (21)

Uno de las principales características de la metodología XP es el desarrollo basado en pruebas, la cual fue adoptada en la realización de la aplicación, a continuación se muestran los resultados de las pruebas realizadas.

3.2.1 Pruebas unitarias

Las pruebas unitarias, clasificadas como pruebas de caja blanca dentro de la metodología de desarrollo **XP**, también llamadas pruebas modulares, permiten determinar si un módulo del sistema está listo y correctamente terminado. El objetivo fundamental de las pruebas unitarias es asegurar el correcto funcionamiento de las interfaces o flujo de datos entre componentes. Conociendo previamente los datos de entrada y salida de cada funcionalidad se deben comparar los datos obtenidos en la prueba con los datos esperados, si son idénticos el módulo supera la prueba.

En la aplicación se le desarrolló esta prueba a la clase ViewerShape encargada de vincular ambas tecnologías, la cual fue explicada anteriormente, obteniendo resultados satisfactorios.

Biblioteca QTest:

En la realización de dichas pruebas fue utilizada la biblioteca QTest, la cual viene integrada en el framework Qt utilizado para el desarrollo de la aplicación. A continuación los resultados obtenidos.

```
Starting /home/achel/programas/build-UnityTest-Qt_4_8_6_qt4-Debug/tst_unitytesttest...
SoQT initialized
********* Start testing of UnityTestTest *********
Config: Using QTest library 4.8.6, Qt 4.8.6
PASS   : UnityTestTest::initTestCase()
PASS   : UnityTestTest::checkgetShape()
PASS   : UnityTestTest::checkgetName()
PASS   : UnityTestTest::checkgetSoGroup()
PASS   : UnityTestTest::cleanupTestCase()
Totals: 5 passed, 0 failed, 0 skipped
********* Finished testing of UnityTestTest *********
/home/achel/programas/build-UnityTest-Qt_4_8_6_qt4-Debug/tst_unitytesttest exited with code 0
```

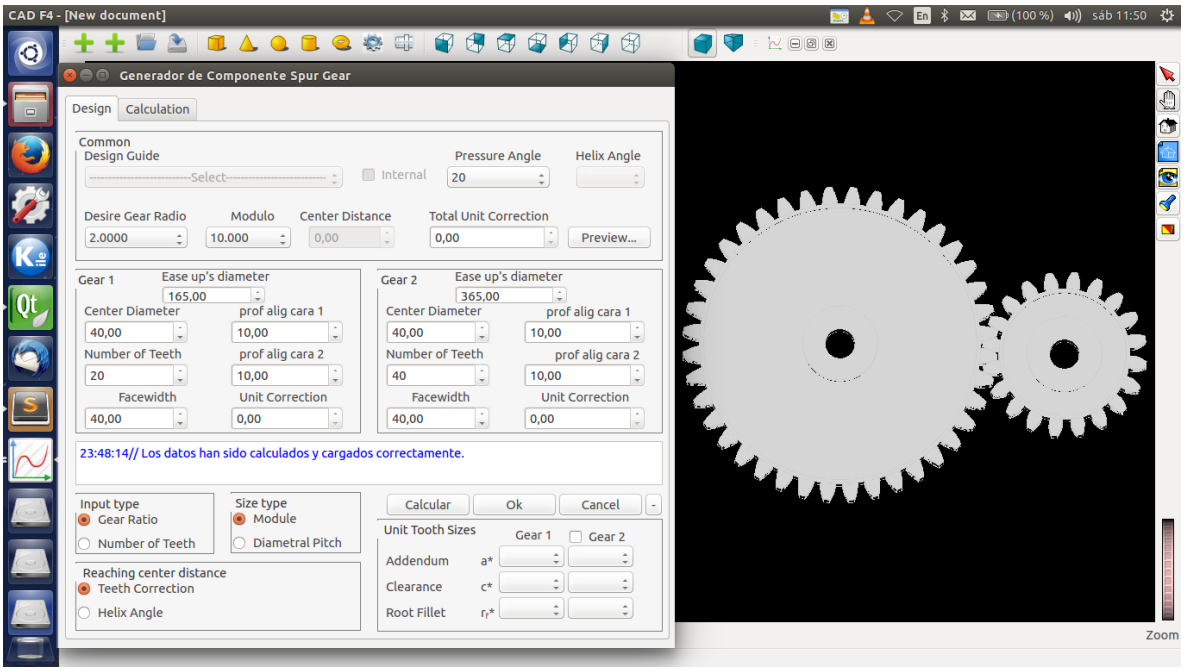
3.2.2 Pruebas de integración

Las pruebas de integración se refieren a la prueba o pruebas de todos los elementos unitarios que componen un sistema, hecha en conjunto, de una sola vez. Consiste en realizar pruebas para verificar que un gran conjunto de partes de software funcionan juntos. Las pruebas de integración es la fase de la prueba

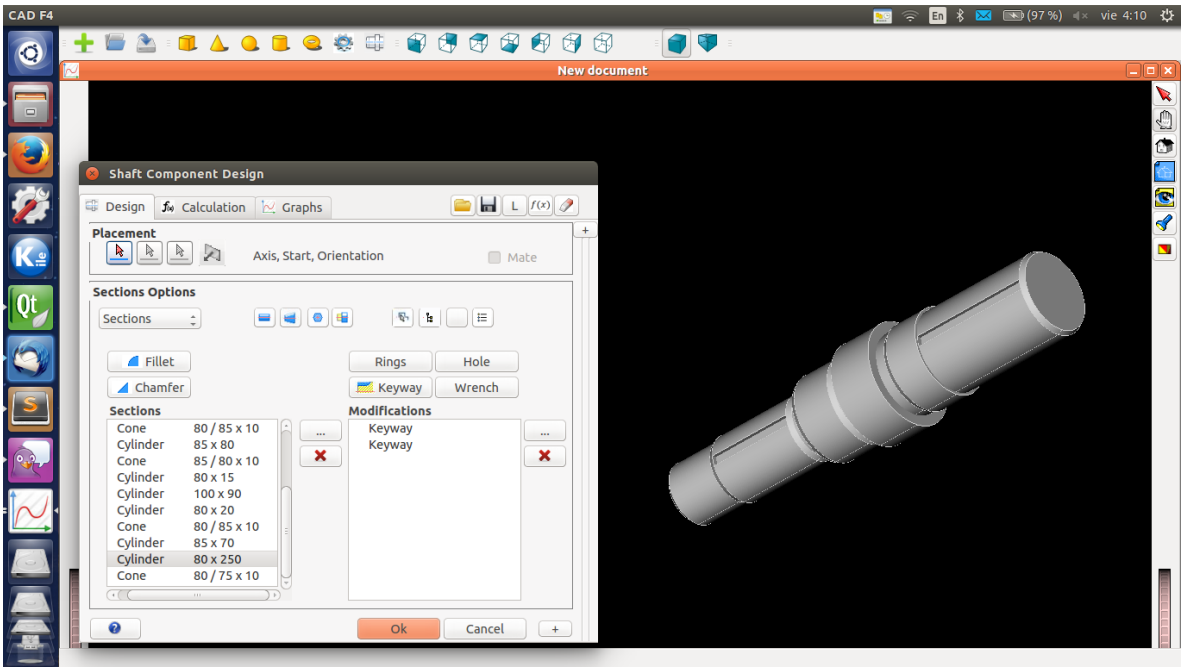
de software en la cual módulos individuales de software son combinados y probados como un grupo. Son las pruebas posteriores a las pruebas unitarias y preceden a las pruebas del sistema.

La aplicación debe tener la capacidad de integrarse con módulos que fueron desarrollados para la misma, a continuación se muestra evidencia grafica de la correcta integración de la misma con los módulos mencionados.

Integración con módulo para la creación de engranajes cilíndricos de dientes rectos-



Integración con módulo para la creación de árboles escalonados.



3.3 Descripción del sistema obtenido

El sistema obtenido cumple con el objetivo de crear una interfaz de usuario adecuada a las necesidades del proyecto CAD2D, contando con una serie de características y funcionalidades que serán explicadas a continuación:

Posee una interfaz gráfica consistente en barras de menús, barras de herramientas, barra de estado, zona de trabajo, y ventanas ancladas, consistente con el diseño de las aplicaciones DAC existentes en el mercado, con el objetivo de suavizar la resistencia al cambio para futuros usuarios del mismo.

Dispone de un componente capaz de visualizar y modelar entidades geométricas compuestas por varias primitivas básicas creadas con la tecnología OpenCASCADE, sirviendo también como plataforma para la integración con módulos implementados utilizando cualquiera de estas tecnologías.

Cuenta con la capacidad de importar archivos de proyectos guardados en los formatos más utilizados en otras aplicaciones de DAC existentes en la actualidad (por ejemplo: .step, .brep y .iges) y visualizarlos adecuadamente. También puede exportar los modelos hechos con los aceleradores de diseño integrados en el mismo, en los formatos mencionados, siendo posible que sean cargados en otros sistemas DAC, tanto propietarios como de código abierto.

Tiene la capacidad de modificar independientemente las propiedades de cada una de las primitivas básicas utilizadas en la confección de una estructura compleja.

También cuenta con un historial de actividades o historial de eventos, como también se le denomina en la literatura referida, lo que permite deshacer o rehacer acciones previamente realizadas, facilitando la manipulación de las entidades.

El sistema se basa en el software FreeCAD en su versión 0.14 con el objetivo de utilizar algunas de las clases ya implementadas en el mismo, por esto a continuación se hará referencia a mayor profundidad a FreeCAD.

3.4 Conclusiones del capítulo

Haciendo uso del análisis realizado en los capítulos anteriores, se logró implementar un visor que cumple con las expectativas planteadas. En este capítulo se discutieron los resultados del proceso de implementación, así como detalles de la aplicación, tales como el estándar de código utilizado y la Pila de Sprint adoptada. También se abordó el proceso de pruebas realizadas al software con el objetivo de garantizar la calidad del mismo.

Conclusiones Generales

- Se desarrolló una aplicación para visualizar entidades geométricas modeladas con la tecnología OpenCASCADE, basada en la biblioteca gráfica Open Inventor; con funcionalidades para exportar e importar datos de geometrías a archivos con formatos .step,.brep y .iges
- La aplicación tiene una arquitectura flexible y abierta, lo que posibilita integrar componentes basados en OpenCASCADE mediante una capa de clases que recolecta los datos de las figuras creadas.
- Es la primera solución de código abierto, en Cuba, que integra aceleradores de diseño en una aplicación para visualizar, importar y exportar la información gráfica generada por los mismos.

Recomendaciones

Agregar a la aplicación la posibilidad de que el usuario seleccione entre varios visores de diferentes tecnologías.

Implementar una estructura de datos para garantizar la persistencia de los mismos.

Glosario de términos

DAC: Diseño Asistido por Computadora.

OCC: OpenCASCADE

OIV: Open Inventor

Bibliografía

1. **Carmona, P. R.** *Introducción a la Visualización 3D.* 2008.

2. **Wikipedia. [En línea] [Citado el: 26 de 2 de 2015.] http://en.wikipedia.org/wiki/Coin3D.**

3. VTK. ***VTK.*** **[En línea] [Citado el: 6 de 2 de 2015.] http://www.vtk.org/.**

4. Solis A, Camilo J y Figueroa D, Roberth G. ***Metodologias de Desarrollo de SW.*** **2011.**

5. Pressman, Roger S. ***Software Engineering.*** **s.l. : Higher Education, 2010. 978-0-07-337597-7.**

6. Jacobson, Ivar, Rumbaugh, James y Booch, Grady. ***El lenguaje unificado de modelado. Manual de referencia.*** **2000.**

7. Larman, Craig. ***UML y Patrones. Introducción al análisis y diseño orientado a objetos.*** **1999.**

8. ¿Qué es un Patrón de Diseño? ***Microsoft.*** **[En línea] [Citado el: 16 de 4 de 2015.] https://msdn.microsoft.com/es-es/library/bb972240.aspx.**

9. ¿Qué es un Patrón de Diseño? ***Microsoft.*** **[En línea] [Citado el: 16 de febrero de 2015.] https://msdn.microsoft.com/es-es/library/bb972240.aspx.**

10. Pérez, Yarisleydis Barzaga. ***Componente de Interpretación de registros.*** **La Habana : s.n., 2011.**

11. Estrada, José Angel Lores. ***Módulo de transformación a entidades 2D.*** **La Habana : s.n., 2012.**

12. Catia_Dassault_Systèmes. Catia. ***Catia.*** **[En línea] Dassault Systèmes. [Citado el: 02 de febrero de 2015.] http://www.3ds.com/products/catia.**

13. ArchiCAD_Graphisoft. ArchiCAD. ***ArchiCAD.*** **[En línea] Graphisoft, 1982. [Citado el: 02 de febrero de 2015.] http://www.graphisoft.com/products/archicad.**

14. Inventor, Autodesk. Autodesk Inventor. ***Autodesk Inventor.*** **[En línea] Autodesk, 1999. [Citado el: 02 de febrero de 2015.] http://www.autodesk.com.**

15. SolidWorks_Corporation. SolidWorks. *SolidWorks Corporation.* [En línea] Dassault Systèmes, 1993. [Citado el: 02 de febrero de 2015.] http://www.solidworks.com.mx.

16. SolidEdge_Intergraph. SolidEdge. [En línea] Siemens PLM Software, 1996. [Citado el: 02 de febrero de 2015.] www.siemens.com/solidedge.

17. Autodesk_AutoCAD. AutoCAD. *AutoCAD.* [En línea] Autodesk, 1982. [Citado el: 02 de febrero de 2015.] http://www.autodesk.es/autocad.

18. FreeCAD. FreeCAD. *FreeCAD.* [En línea] [Citado el: 02 de febrero de 2015.] http://freecadweb.org.

19. Comunidad_LibreCAD. LibreCAD. *LibreCAD.* [En línea] [Citado el: 02 de febrero de 2015.] http://librecad.org/cms/home.html.

20. *Charla 1: Introduccion a los patrones de diseño. Algunos patrones básicos.* 18.

21. HENDRICKSON, E. *Agility for Testers.* Pacific Northwest Software Quality Conference : s.n., 2011.

22. Ditbutec. [En línea] 2015. [Citado el: 6 de 2 de 2015.] http://ditbutec.es.tl/Historia-del-dibujo-t-e2-cnico.htm.

23. Ferrer, Martín. arquitectura.com. [En línea] http://www.arquitectura.com/cad/artic/elcad.asp.

24. Doxygen. [En línea] http://www.doxygen.org.

25. SourceForge. [En línea] http://cccc.sourceforge.net/.

26. Web FreeCAD. *Web FreeCAD.* [En línea] [Citado el: 10 de 3 de 2015.] http://freecadweb.org/.

27. Open Inventor. [En línea] [Citado el: 13 de 5 de 2015.] http://www.openinventor.com.

28. Open Source Ecology. [En línea] 22 de 3 de 2015. [Citado el: 13 de 5 de 2015.] http://opensourceecology.org/wiki/CAD_tools.

29. Wernecke, Josie. *The Inventor Mentor.* s.l. : Addison-Wesley Publishing Company, 1994. ISBN 0-201-62495-8.