



UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS
SIPII, FACULTAD 4

FUNCIONALIDADES PARA EL MODELO APLICACIÓN-DOCUMENTO-ATRIBUTO BASADAS EN EL MARCO OCAF PARA LA APLICACIÓN INGENIERO

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

Autor: Eduardo González Castillo

Tutor: Dr. Augusto Cesar Rodriguez Medina

La Habana, 2019

A mis padres...

Agradecimientos

Agradezco encarecidamente a los profesionales de la Universidad de las Ciencias Informáticas por los conocimientos brindados en especial a Augusto Cesar Rodríguez Medina por el apoyo hasta el último momento.

Declaración de autoría

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales sobre esta, con carácter exclusivo.

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

Eduardo González Castillo
Autor

Dr. Augusto Cesar Rodriguez Medina
Tutor

Abstract:

OpenCASCADE application framework offers functionalities for rapidly developing of computer-aided design systems, these functionalities allows an adequate data's treatment. Its architecture is based on Application-Document-Attribute and to gain access to the advantages offered by this framework, the bases of it must be established. This paper is intended for the integration of the OCAF architecture to a CAD system for managing the information generated in the design process.

keywords: OCAF, architecture, integration, functionalities

El marco de trabajo para aplicaciones de OpenCASCADE ofrece funcionalidades para el desarrollo rápido de sistemas de diseño asistido por computadora, dichas funcionalidades permiten realizar un adecuado tratamiento de los datos. Su arquitectura está basada en Aplicación-Documento-Atributo y para tener acceso a las ventajas que ofrece este marco de trabajo, se deben establecer las bases de misma. Este documento esta destinado a la integración de la arquitectura de OCAF a un sistema CAD para gestionar la información generada en el proceso de diseño.

Palabras clave: OCAF, arquitectura, integrar, funcionalidades.

Lista de tareas pendientes	xv
Introducción	1
1 Fundamentación Teórica	4
1.1 Sistemas comerciales para el diseño asistido por computadora	4
1.1.1 Autodesk Inventor	4
1.1.2 Solid Edge	5
1.1.3 Catia	6
1.1.4 SolidWorks	7
1.2 Aspectos teóricos sobre el modelo arquitectónico Aplicación-Documento-Atributo.	8
1.3 Descripción de las funcionalidades de OCAF	9
1.3.1 Arquitectura	10
1.3.2 Modelo de llave de referencia	12
1.3.3 Estructura de datos	13
1.3.4 Atributos de formas	14
1.3.5 Visualización de los atributos	15
1.3.6 Servicios de función	16
1.4 Análisis del sistema FreeCAD	16
1.4.1 Núcleo	17
1.4.2 Documento	18
1.4.3 Aplicación e interfaz de usuario	18
1.4.4 Visualización	19
1.4.5 Uso de OCAF en FreeCAD	19
1.4.6 Método para guardar el historial de modificaciones a un modelo (History Based Feature)	20
1.4.7 Archivos de intercambio	21
1.5 Conclusiones del capítulo	24

2	Metodología y tecnologías para el proceso de desarrollo	25
2.1	Lenguaje de desarrollo	25
2.2	Framework de desarrollo Qt	25
2.3	Open Cascade Technology	26
2.4	Sistema de control de versiones (GitLab)	28
2.5	Doxygen	29
2.6	Visual Paradigm	29
2.7	CLOC	30
2.8	Metodología de desarrollo	30
2.9	Conclusiones del capítulo	31
3	Propuesta de solución	32
3.1	Descripción de las funcionalidades	32
3.2	Estándar de codificación	33
3.3	Requisitos	35
3.3.1	Requisitos Funcionales	35
3.3.2	Requisitos No Funcionales	36
3.4	Historias de usuario	36
3.5	Diseño	37
3.5.1	Estilo y patrón arquitectónico del software	37
3.5.2	Patrones de Diseño	38
3.5.3	Diagrama de clases	39
3.5.4	Detalles de implementación	40
3.5.5	Resultados de la implementación	42
3.6	Conclusiones del capítulo	45
4	Evaluación del resultado	46
4.1	Pruebas	46
4.1.1	Niveles de prueba	46
4.2	Métodos de pruebas	47
4.2.1	Niveles de caso de prueba	47
4.2.2	Pruebas unitarias	48
4.2.3	Pruebas de integración	49
4.2.4	Prueba de aceptación	49
4.2.5	Resultados de las pruebas funcionales	49
4.3	Conclusiones del capítulo	51
	Conclusiones	52

Recomendaciones	53
Referencias bibliográficas	54
Apéndices	57
A Historias de usuarios	58
B Diseño de casos de prueba	61

Índice de figuras

1.1	Árbol de Autodesk Inventor.	5
1.2	Árbol de Solid Edge.	6
1.3	Árbol de Catia.	7
1.4	Árbol de SolidWorks.	8
1.5	Modelo del Documento	11
1.6	Aplicación-Documento-Atributos	12
1.7	Manejador topológico, manejador por llave de referencia	13
1.8	Simple modelo del framework	14
1.9	Relaciones del atributo TNamingNamedShape	15
1.10	Relaciones de la presentación visual del atributo	16
1.11	Árbol de FreeCAD	21
1.12	Árbol de Inventor	21
1.13	Modelos de facetas. (a) Archivo STL. (b) Una faceta.	24
1.14	Formato STL de un cubo	24
2.1	Estructura de dato topológico	27
3.1	Estándar de codificación	33
3.2	Estándar de codificación	34
3.3	Estándar de codificación	34
3.4	Ejemplo de estándar de codificación <i>CamelCase</i>	35
3.5	Diagrama de Arquitectura de programa principal/subprograma	38
3.6	Diagrama de clases de la propuesta de solución	40
3.7	Nuevo documento OCAF	43
3.8	Menú con la opción crear nuevo documento	43
3.9	Insertar Primitivas	43
3.10	Íconos que crean las primitivas	43
3.11	Prototipo de interfaz para la caja, el cono el torus, la esfera y el cilindro respectivamente	44
3.12	Deshacer	44
3.13	Rehacer	44
3.14	Salvar documento de OCAF	44

3.15 Exportar documento de OCAF	44
3.16 Visor de Inventor	45
3.17 Visor de OpenCASCADE	45
4.1 Prueba unitaria realizada con Qt Test.	49
4.2 Iteraciones de las pruebas	51

Índice de tablas

1.1	Servicios brindados por OCAF.	9
1.2	Formatos de guardados de OCAF y sus atributos.	10
4.1	No conformidades.	50
A.1	Historia de usuario # 1	58
A.2	Historia de usuario # 2	58
A.3	Historia de usuario # 3	59
A.4	Historia de usuario # 4	59
A.5	Historia de usuario # 5	59
A.6	Historia de usuario # 6	60
B.1	Diseño de Casos de Prueba RF 1	61
B.2	Diseño de Casos de Prueba RF 2	61
B.3	Diseño de Casos de Prueba RF 3	62
B.4	Diseño de Casos de Prueba RF 3	62
B.5	Diseño de Casos de Prueba RF 5	62
B.6	Diseño de Casos de Prueba RF 6	63

Lista de algoritmos

Lista de códigos fuentes

Lista de tareas pendientes

El presente trabajo contiene los aspectos fundamentales de la investigación realizada, con el propósito de integrar el modelo arquitectónico Aplicación-Documento-Atributo del marco de trabajo para aplicaciones de OpenCASCADE (**OCAF**, por sus siglas en inglés) a la aplicación Ingeniero. Siguiendo este modelo se realizaron funcionalidades provistas por OCAF para el tratamiento de los datos, específicamente su persistencia y serialización en formatos estándares y de intercambio; y la implementación de las operaciones deshacer y rehacer cambios (**undo/redo**).

Los sistemas de diseño e ingeniería asistido por computadora (**Computer Aided Design** y **Computer Aided Engineering**, CAD/CAE por sus acrónimos) permiten automatizar tales procesos y reducir los tiempos de trabajo en el modelado; diseñar piezas o ensambles se hace de una manera más amena e intuitiva para los ingenieros, se puede conceptualizar el objeto a diseñar con más facilidad interactuando con el visor de la aplicación; en este contexto se pueden considerar diseños alternativos, o modificar con rapidez algunos rasgos o características del modelado que se esté realizando; con los sistemas de diseño asistido por computadora es posible intercambiar información gráfica de los diseño entre diferentes sistemas, y esto representa una mejora en efectividad a la hora de interpretar diseños; sin embargo, estos modelados requieren una arquitectura y un manejo adecuado de los datos para preservar su información.

Las limitaciones que tiene la industria cubana de adquirir sistemas de diseño asistido por computadora, tanto por su alto precio en el mercado, como por las restricciones que implican el bloqueo de los Estados Unidos, sugieren a que se exploren otras alternativas para ofrecer soluciones informáticas para esta esfera del diseño, fundamentalmente, porque existe la posibilidad de acceder a funcionalidades que están presente en tecnologías de código abierto.

Una de estas alternativas es un emprendimiento académico denominado Ingeniero, que pretende contener algunas soluciones para el diseño asistido por computadora. El conjunto de soluciones que están encapsulada en el programa Ingeniero, logra acelerar el diseño de algunas piezas complejas, pero la aplicación carece funcionalidades importantes como:

- El documento existente no tiene implementadas las funcionalidades de deshacer rehacer, pero el nuevo documento basado en la arquitectura de OCAF, se le debe agregar además estas operaciones.
- El árbol, que debe reflejar el historial de rasgos de la pieza, no muestra toda la información del modelo de una forma lógica, dificultando la visualización precisa del procedimiento del diseño.
- Las variantes de persistencia de los datos no están diversificadas y se requiere agregar algunos formatos. Este sistema no tienen otras opciones de guardados que son necesarios para tener información del

diseño.

Por lo anteriormente descrito llegamos a la definición del siguiente **problema de la investigación**: ¿Como integrar el modelo arquitectónico propuesto por OCAF para agregar las funcionalidades correspondiente a la organización, manipulación y procesamiento de los datos, a partir de los modelos generados por el sistema Ingeniero? El **objeto de estudio** se centra en las funcionalidades que ofrece la tecnología de Open-CASCADE y el estudio de la aplicación FreeCAD. Para solucionar el problema planteado se propone como **objetivo general** desarrollar las funcionalidades requeridas para el manejo de los datos contenidos en el documento del modelo arquitectónico. A partir del objetivo general definido, se derivan los siguientes objetivos específicos:

- Integrar la arquitectura de OCAF en forma de un nuevo documento que sea independiente a la arquitectura del sistema Ingeniero.
- Rehacer y deshacer cambios en el nuevo documento de OCAF (**undo/redo**).
- Diversificar los formatos nativos en la persistencia de los datos agregando los formatos de OCAF.
- Verificar los procesos de exportación en formatos normalizados (**STEP, IGES, STL, BREP**) en el nuevo documento de OCAF.

Para dar cumplimiento a estos objetivos específicos se proponen las siguientes **tareas de la investigación**:

Marco teórico

- Definición del perfil y diseño de la investigación.
- Búsqueda y revisión bibliográfica sobre el objeto de la investigación.

Análisis

- Estudio del **framework** de aplicaciones de OpenCASCADE.
- Análisis de los códigos fuentes de las aplicaciones FreeCAD con la finalidad de identificar y evaluar la reutilización de sus funcionalidades.
- Proceso de síntesis (obtención de conclusiones, resultados, definición de las formas de hacer y conformación de la estructura).
- Estudio de los aspectos de la metodología de desarrollo de software (AUP-UCI) que se aplicarán en la investigación.
- Obtención de requisitos a partir de: los diseños existentes de el árbol de historial de rasgos en sistemas comerciales y de código abierto, la forma de guardar e importar/exportar en diferentes formatos.

Diseño

- Definición de la arquitectura, lo que implica además definir los patrones de diseño con los que cumplirá.

- Elaboración de la estructura de clases.
- Modelado del flujo de datos.
- Diseño de la interfaz gráfica.

Implementación y pruebas

- Desarrollo de funcionalidades para integrar la arquitectura OCAF al sistema **Ingeniero**
- Desarrollo de las funciones **undo/redo**.
- Desarrollo de las funciones para la persistencia de los datos por las diferentes variantes, incluyendo los formato nativos de OCAF y los de intercambios normalizados de ficheros (STEP, STL, IGES, BREP).
- Pruebas al conjunto de funcionalidades desarrolladas.

Para resolver y dar cumplimiento a los objetivos y las tareas propuestas se emplearon como **métodos de investigación**:

Métodos teóricos:

- **Análisis y síntesis:** se empleó para la construcción y desarrollo de la teoría, para la profundización en el tema y la sistematización del conocimiento.
- **Modelado:** se utilizó para el modelado de los artefactos correspondientes al análisis, diseño e implementación de las funcionalidades requeridas.

Métodos empíricos:

- **Observación:** permitió estudiar cómo funciona en el proceso de desarrollo del árbol de historial de rasgos con la tecnología de OpenCascade. Estudio de sistemas comerciales.

Este documento esta estructurado por cuatro capítulos: En el **Primer capítulo** se lleva a cabo un estudio del estado del arte sobre la persistencia y estructura de datos en sistemas comerciales, con el objetivo de capturar requisitos necesarios, más adelante se hace un estudio del marco de aplicaciones de OpenCASCADE y las funcionalidades que este provee y por último se analiza el sistema de código abierto FreeCAD en el que se analizara su estructura. En el **Segundo capítulo** se definen las herramientas y metodologías de trabajo adoptados para el desarrollo de la propuesta de solución. En el **Tercer capítulo** se exponen los principales aspectos técnicos de la propuesta de solución: requisitos funcionales y no funcionales, su descripción; detalles de implementación y una panorámica de la arquitectura y patrones de diseño empleados. El **Cuarto capítulo** se refiere a la verificación de los resultados obtenidos a través de pruebas.

En este capítulo se expone aspectos del proceso de investigación que tributaron al análisis y entendimiento de las arquitectura de los sistemas asistido por computadora; también algunos criterios sobre los sistemas comerciales, específicamente en lo referido a la persistencia de datos, y cómo está reflejado el árbol de historial de rasgos. Más adelante se hace un análisis de las funcionalidades de OCAF para facilitar el desarrollo de aplicaciones de este tipo. Se finaliza describiendo el sistema de código abierto FreeCAD.

1.1. Sistemas comerciales para el diseño asistido por computadora

1.1.1. Autodesk Inventor

Es una herramienta de diseño asistido por computadora producida por la empresa Autodesk inc. y salió al mercado por primera vez en 1999; es un modelador paramétrico de sólidos en 3D que permite a los ingenieros o diseñadores crear prototipos digitales, gracias a un conjunto de herramientas para el diseño mecánico 3D, simulación, análisis, visualización y documentación; con esta herramienta se pueden integrar planos en 2D de AutoCAD y crear representaciones digitales del producto final que les permite validar la forma, dimensiones, precisión del ensamble, así como el comportamiento del producto antes que sea construido (Inventor, 2019), utiliza un gestor de formas (**Shape Manager**) como **kernel** de modelado geométrico el cual pertenece a Autodesk; en la versión profesional incluye herramientas necesarias para crear piezas de plástico y sus respectivos moldes de inyección, cuenta con análisis de tensión por elementos finitos y análisis dinámico. En la figura 1.1 se muestra en el panel izquierdo el árbol que se va conformando a medida que se trabaja.

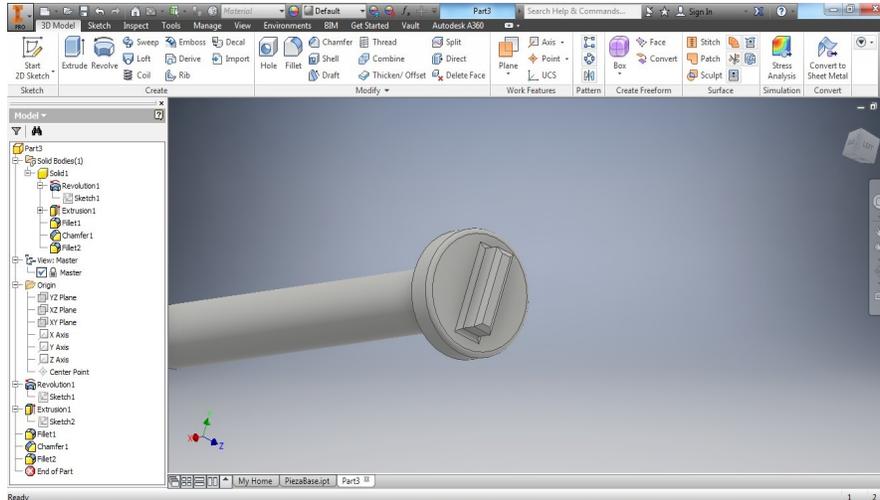


Figura 1.1. Árbol de Autodesk Inventor.

1.1.2. Solid Edge

Fue lanzado al mercado por primera vez en 1996, inicialmente desarrollado por Intergraph, pertenece y es desarrollado por Siemens AG. Es un programa de diseño paramétrico CAD de piezas tridimensionales. Permite el modelado de piezas de distintos materiales, doblado de chapas, ensamblaje de conjuntos, soldadura y funciones de dibujo en plano para ingenieros. Solid Edge es una herramienta de software fácil de usar para el proceso de desarrollo de productos diseño en 3D, simulación, fabricación, gestión del diseño ya que combina la velocidad y simplicidad del modelado directo con la flexibilidad y el control de diseño paramétrico hecho posible con **synchronous technology** (tecnología sincrónica). Este programa comercial solo se puede usar en Windows y MacOS. (Siemens, 2019)

Las herramientas más destacadas que incluye Solid Edge:

- **Diseño generativo:** Optimización topológica dentro de sus herramientas de modelado 3D, ayudando a los diseñadores a idear componentes más ligeros, minimizando el uso de material innecesario y creando diseños altamente personalizados capaces de aprovechar las nuevas tecnologías de impresión 3D.
- **Migración inteligente de archivos 3D:** Se puede importar archivos de otros programas CAD 3D sin perder las propiedades asociadas como propiedades personalizadas, materiales, configuraciones, asociaciones y características del ensamble.
- **Documentación técnica:** Permite generar instrucciones de trabajo 3D, catálogos ilustrados de los diseños, documentación de montaje y mantenimiento, y manuales de producto. Se actualiza toda la documentación generada de forma automática cuando se cambia el modelo 3D, evitando que la documentación quede obsoleta.
- **Flow Simulation:** Herramienta de análisis de dinámica de fluidos computacional (CFD) integrada e intuitiva que proporciona a los diseñadores información valiosa sobre el comportamiento del producto

desde el inicio del ciclo de diseño.

- **Fabricación en Impresoras 3D:** Ofrece una interfaz dedicada a impresión 3D que incluye una vista previa dinámica con la información topológica asociada, totalmente integrada con las instalaciones de Microsoft 3D Builder. Permite establecer tolerancias, escala o reorientar el modelo y validarlo para evitar errores antes de exportar un resultado STL o 3MF.
- **Catchbook:** Sirve para crear diseños 3D a partir de bocetos realizados en Smartphones y Tablets.

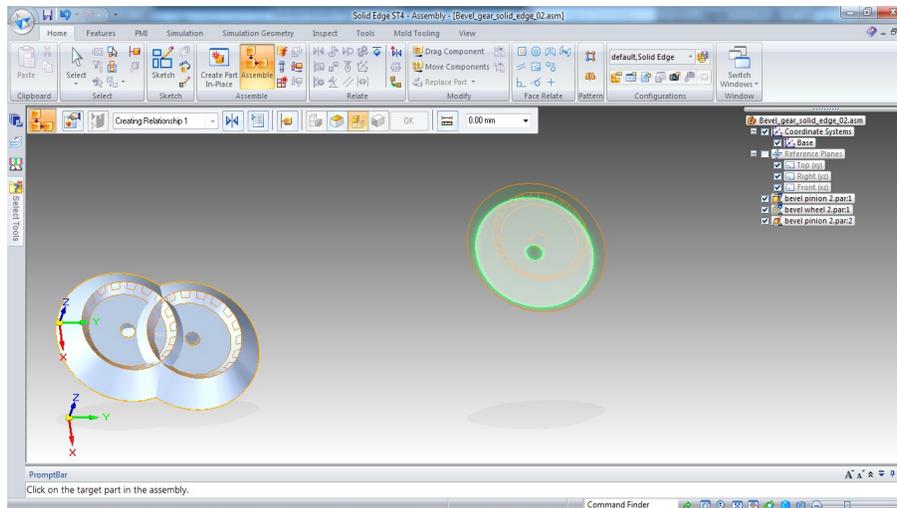


Figura 1.2. Árbol de Solid Edge.

1.1.3. 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. Fue desarrollado en 1971 de manera interna por la empresa francesa productora de aviones *Avions Marcel Dassault*, en aquel momento los usuarios utilizaban CATIA para desarrollar avión de combates *Dassault's Mirage*. Más tarde fue empleado en el sector aeroespacial, automovilístico, construcción de barcos y otras industrias. (Ecured, 2019)

Inicialmente se llamó CATI (Conception assistée tridimensionnelle interactive en Francés), fue renombrado como CATIA en 1981 cuando Dassault creó una sucursal para desarrollar y vender el software y firmar una acuerdo de distribución no exclusivo con IBM.² CATIA puede ser aplicado a una amplia variedad de industrias, desde aeronáutica y defensa, automovilística, y equipamiento industrial, hasta altas tecnologías, construcción de barcos, bienes de consumo, diseño en planta, embalaje de bienes de consumo, ciencias biológicas, arquitectura y construcción, procesos de generación de energía y petroleras. (Catia, 2019)

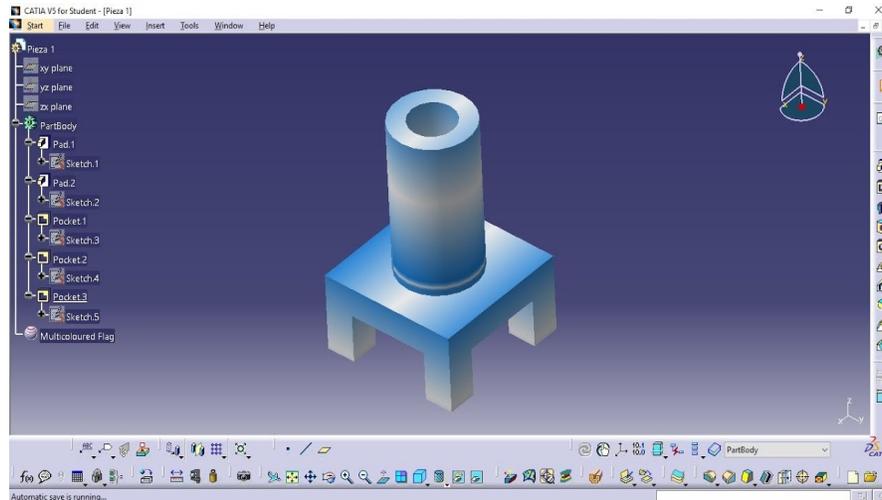


Figura 1.3. Árbol de Catia.

El árbol de Catia está bien estructurado, logra captar la intención del usuario, además es mostrado de manera que se maximice el espacio en al área de trabajo.

1.1.4. SolidWorks

La empresa SolidWorks Corporation fue fundada en 1993 por Jon Hirschtick con su sede en Concord, Massachusetts y lanzó su primer producto, SolidWorks 95, en 1995. En 1997 Dassault Systèmes, mejor conocida por su software CATIA, adquirió la compañía y actualmente posee el cien por ciento de sus acciones. SolidWorks permite modelar piezas y conjuntos, extraer de ellos tanto planos técnicos como otro tipo de información necesaria para la producción, funciona con base en las nuevas técnicas de modelado con sistemas destinados al diseño mecánico. El programa posee un grupo de productos con distintas funcionalidades:

- **Soluciones CAD 3D:** permite acelerar el desarrollo de productos, reducir los costes de fabricación, y mejorar la calidad y fiabilidad del producto en una gran variedad de sectores y aplicaciones.(Izard, 2017)
- **Visualización:** proporciona un conjunto de herramientas de software independientes que combinan las funciones de renderizado líderes del sector con unas características y flujos de trabajo orientados al diseño, las cuales facilitan la creación fácil y rápida de contenido visual a diseñadores, ingenieros, expertos de marketing y otros creadores de contenidos.(SolidWorks-Corp, 2017a)
- **Simulación:** brinda una herramienta para realizar pruebas con una amplia variedad de parámetros (durabilidad, respuesta dinámica y estática, movimiento del ensamblaje, transferencia de calor, dinámica de fluidos y moldeo de plásticos por inyección) durante el proceso de diseño.(Simulación-SolidWorks, 2017)
- **Gestión de datos de productos:** permite tener control sobre los datos de diseño y mejorar considerablemente la manera en que sus equipos administran y colaboran en el desarrollo de produc-

tos.(SolidWorks-Corp, 2017c)

- **Diseño eléctrico:** proporciona una serie de funcionalidades de diseño de sistemas eléctricos para satisfacer las necesidades de los profesionales. (SolidWorks-Corp, 2017b)

En este caso la estructura del árbol 1.4 no contiene una información tan explícita como Inventor o Catia, lo cual es un requerimiento necesario en las aplicaciones de diseño asistido por computadora para que el usuario pueda tener la mayor referencia al proceso del diseño.

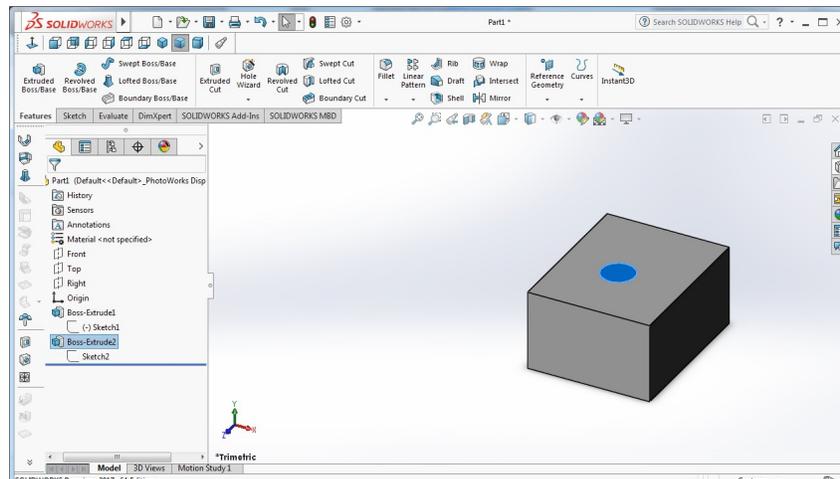


Figura 1.4. Árbol de SolidWorks.

Una vez hecho el estudio de los sistemas comerciales daremos paso al estudio de las tecnologías de OpenCASACDE, veremos las posibilidades que nos brinda para establecer la arquitectura así como algunas de sus funcionalidades. Analizaremos el sistemas FreeCAD en varios aspectos.

1.2. Aspectos teóricos sobre el modelo arquitectónico Aplicación-Documento-Atributo.

El desarrollo de las aplicaciones para el diseño y la ingeniería asistido por computadora (CAD/CAE) en las últimas cuatro décadas, ha permitido establecer que el modelo arquitectónico fundamental empleado en estas, es el denominado Aplicación-Documento-Atributos. Para una adecuada integración de este **framework** a una aplicación de diseño asitido por computadora es necesario tomar en cuenta los siguientes aspectos: (Tomado del sitio oficial de OpenCASCADE (OpenCASCADE, 2019a))

- Definición de los componentes de software y la manera en que estos serán relacionados. (diseñar la arquitectura de la aplicación).
- Definición del modelo de datos que sea capaz de soportar las funcionalidades requeridas por el usuario.

- Estructurar el software con el objetivo de lograr:
 - La sincronización entre las modificaciones que se realizan a un objeto y la visualización.
 - Establecer las bases para que las aplicación soporte las operaciones de rehacer/deshacer.
- Desarrollar la función para el salvado de datos.
- Construir la aplicación de interfaz de usuario.

1.3. Descripción de las funcionalidades de OCAF

OCAF es una plataforma para el desarrollo rápido y específico de aplicaciones de diseño, soporta el modelado geométrico en dos o tres dimensiones. OCAF ofrece las siguientes facilidades:

- Los mecanismos para soportar la aplicación ya son administrados.
- La aplicación puede ser prototipada gracias al acoplamiento con otros módulos de OpenCASCADE.
- Al ser una plataforma de código abierto garantiza el término de largo uso para el desarrollo.

La siguiente tabla muestra una comparativa entre usar solamente las librerías de OpenCASCADE y usar OCAF para el desarrollo de una aplicación.

Tareas de desarrollo	Comentarios	Sin OCAF	Con OCAF
Creación de la geometría	Algoritmo que llama a las bibliotecas de modelado	Tiene que ser creada por el desarrollador	Tiene que ser creada por el desarrollador
Organización del dato	Incluyendo atributos específicos y proceso de modelado	Tiene que ser creada por el desarrollador	Simplificado
Salvar el dato en un archivo.	Noción del documento	Tiene que ser creada por el desarrollador	Provisto
Manejo de la vista del documento	Funciones para el trato de la vista	Tiene que ser creada por el desarrollador	Provisto
Infraestructura de la aplicación.	Nuevo, Abrir, Cerrar, Guardar y Guardar como archivo.	Tiene que ser creada por el desarrollador.	Provisto
Deshacer y Rehacer	Robusto, multi-nivel	Tiene que ser creada por el desarrollador	Provisto
Cuadros de diálogos específicos de la Aplicación.	Donde se muestran mensajes o se introducen valores.	Tiene que ser creada por el desarrollador	Tiene que ser creada por el desarrollador.

Tabla 1.1. Servicios brindados por OCAF.

1.3.1. Arquitectura

OCAF proporciona un modelo orientado a objetos basado en Aplicación-Documento-Atributo, que consiste en una biblioteca de clases escritas en C++.

La clase principal, **Aplicación**, es una clase abstracta que tiene la responsabilidad de manejar los documentos durante la sesión de trabajo. Los servicios por esta clase incluyen:

- Abrir y salvar documentos.
- Inicializar las vistas de los documentos.

OCAF proporciona varios formatos estándar, cada uno de los cuales cubre algunos conjuntos de atributos de OCAF:

Formato	Kit de herramientas de persistencia	Atributos cubierto por OCAF
Formatos heredados (solo lectura)		
OCC-StdLite	TKStdL	TKLCAF
MDTV-Standard	TKStd	TKLCAF + TKCAF
Formatos Binarios		
BinLOcaf	TKBinL	TKLCAF
BinOcaf	TKBin	TKLCAF + TKCAF
BinXCAF	TKBinXCAF	TKLCAF + TKCAF + TKXCAF
TObjBin	TKBinTObj	TKLCAF + TKTObj
Formatos XML		
XmlLOcaf	TKXmlL	TKLCAF
XmlOcaf	TKXml	TKLCAF + TKCAF
XmlXCAF	TKXmlXCAF	TKLCAF + TKCAF + TKXCAF
TObjXml	TKXmlTObj	TKLCAF + TKTObj

Tabla 1.2. Formatos de guardados de OCAF y sus atributos.

El **documento** es el contenedor de los datos de la aplicación, su principal objetivo es centralizar las notificaciones de edición de datos a fin de proporcionar mecanismos para deshacer y rehacer cambios. Cada documento se guarda en un único archivo ASCII definido por su formato y extensión.

De manera general el documento es el encargado de:

- Controlar la notificación de cambios.
- Actualizar las referencias externas.
- Controlar el salvado y cargado de los datos, la transacción de comandos y las opciones de deshacer rehacer.

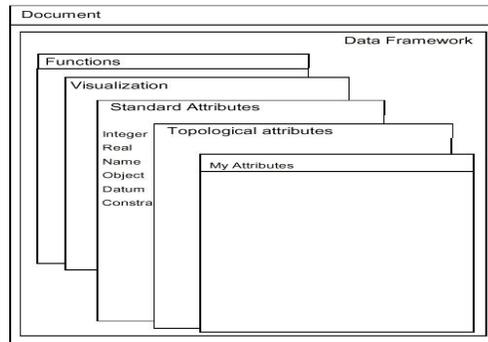


Figura 1.5. Modelo del Documento

Los datos de la aplicación son descrito por **atributos**, en OCAF los atributos definidos son:

- **Atributos estándares:** Permiten operar con datos comunes y simples en la estructura de datos, realizar funciones auxiliares, crear dependencias.
- **Atributos Shape:** Estos contienen la geometría de todo el modelo con sus elementos respectivos, incluyendo referencia a las formas y su evolución.
- **Atributos de visualización:** Permite posicionar el visor hacia el **framework** de datos, la representación visual de objetos y otras informaciones visuales, las cuales son necesarias para la representación gráfica.
- **Servicios de funciones:** El propósito de dichos atributos es la de reconstruir objetos después que han sido modificadas. Mientras que el documento maneja la notificación de cambios, una función maneja la propagación de estos cambios. El mecanismo función provee el enlace entre funciones y la llamado a varios algoritmos. Las únicas funciones que hay implementar son:
 - Copiar el atributo.
 - Convertirlo desde y hacia el almacenado persistente del dato.

La figura 1.6 muestra como está enfocado la arquitectura del **framework**: una aplicación puede contener de cero a muchos documentos, cada documento va a contener atributos representados por **labels** que pueden ser una forma, un visor y una función.

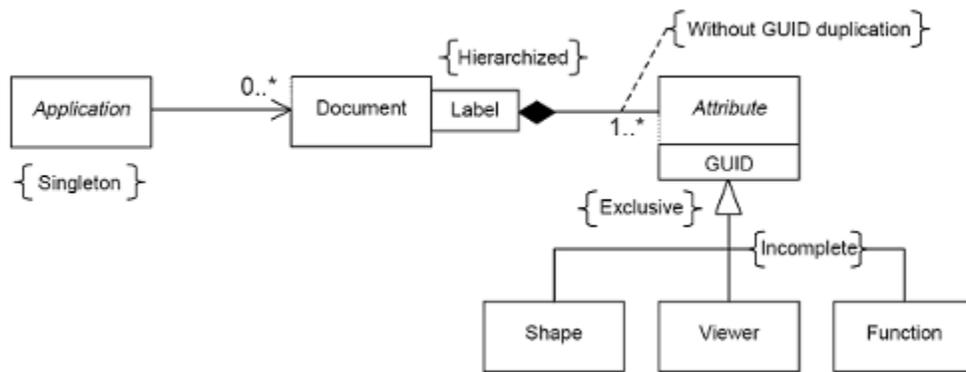


Figura 1.6. Aplicación-Documento-Atributos

Podemos observar el uso de patrón **singleton**, esto lo debemos usar cuando:

- Debe haber una instancia de la clase, y tiene que ser accesible para clientes desde un punto de acceso bien conocido.
- La sola instancia debes ser extensible por subclase, y clientes deben ser capaz de usar una instancia extendida sin modificar el código.

El patrón **singleton** aporta lo siguiente:

- **Acceso controlado a instancia exclusiva:** encapsula su instancia exclusiva, puede tener control estricto sobre cómo y cuándo los clientes acceden a este.
- **Nombre de espacio reducido:** mejora sobre las variables globales. Evita la contaminación de espacio de nombre con variables globales que almacenan instancia exclusivas.
- **Permite el refinamiento de operaciones y representaciones.** puede ser una subclase, y es fácil para configurar una aplicación con una instancia de la clase extendida.
- **Permite diferentes números de instancia:** El patrón lo hace más fácil al permitir más de una instancia de la clase **singleton**. Más aún se puede usar el mismo acercamiento para controlar el número de instancias que la aplicación usa. Solo la operación que le da acceso a la instancia **singleton** es la que se necesita cambiar.
- **Más flexible que las operaciones de las clases:** Otra manera de empaquetar una funcionalidad **singleton** es al usar operaciones de clases (esto es, funciones miembros estáticas en C++). Esta técnica de este lenguaje hace difícil de cambiar un diseño para permitir más de una instancia de una clase. (Gamma; Helm; Johnson y Vlissides, 1994)

1.3.2. Modelo de llave de referencia

Un mecanismo importante que ofrece a tecnología de OpenCASCADE es el denominado llave de referencia; en la mayoría de los sistemas de modelado existente, los datos son manejadores topológicos. Usualmente usan representación de fronteras (Brep), donde los modelos geométricos están definidos por una

colección de caras, aristas y vértices, a las cuales los datos de la aplicación están adjuntados. En OCAF son manejadores de una llave de referencia, esto es un modelo uniforme en la cual son la identificación persistente del dato. Estos pueden incluir: (OpenCASCADE, 2019a)

- un color.
- un material.
- información de una determinada arista.

Cuando un modelo geométrico está parametrizado es posible cambiar el valor del parámetro usado para construir el modelo, la geometría es dependiente de cualquier cambio. Para mantener el vínculo del dato de la aplicación, la geometría tiene que ser diferenciada de otros datos.

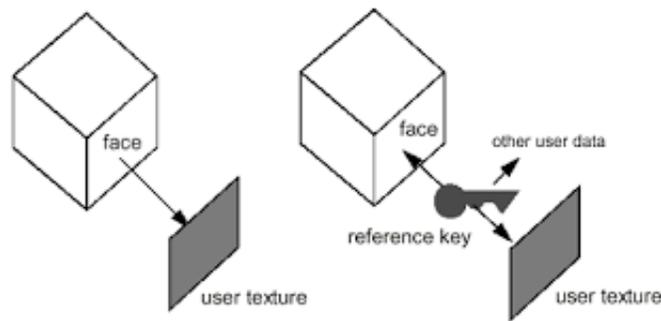


Figura 1.7. Manejador topológico, manejador por llave de referencia

Se generan llaves de referencia con el objetivo de dar semántica al dato, estas están de acuerdo a la figura creada; es decir, si se crea un prisma entonces deberemos crear una llave de referencia para la base, otra para las caras laterales y una última para la cara superior.

1.3.3. Estructura de datos

Los programas CAD son interactivos, orientados a gráficos, y exigen un gran uso de memoria. Esto hace que sea necesaria una buena organización y estructura de los datos, para garantizar que los programas se ejecuten sin problemas y de manera eficiente.

El **framework** de datos de OpenCASCADE es la creación de una llave de referencia en un modelo de estructura de árbol. Los bloques de construcción de este enfoque son:

- Tag
- Label
- Attribute

La primera etiqueta (label) en el **framework** es la etiqueta raíz del árbol, cada una de ellas tiene un identificador (tag) expresada como un valor entero, y son definidas de forma única por la entrada como una lista

de identificadores desde la raíz.

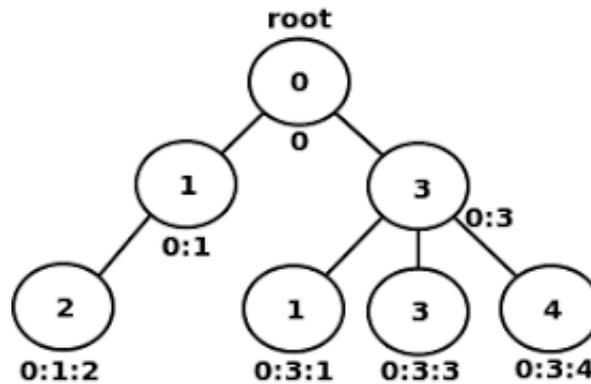


Figura 1.8. Simple modelo del framework

En esta imagen los círculos contienen identificadores de las etiquetas, la etiqueta raíz siempre tiene identificador 0. Las hijas de la etiqueta raíz con identificador 1 y 3, son hermanas, la lista de identificadores de la etiqueta de la parte inferior derecha es 0: 3: 4: esta última tiene identificador 4, su padre (con la entrada 0: 3) tiene identificador 3 y su ancestro 0.

1.3.4. Atributos de formas

Un atributo topológico puede ser visto como un gancho hacia la estructura topológica, es posible adjuntar datos para definir referencias hacia esto. Los atributos de formas en OCAF son usados por objetos topológicos junto con su evolución de acceso, estos objetos son almacenados en el atributo a través de la librería **TNamingUsedShapes** en el nivel de raíz del **framework** de datos; donde se tiene un mapa con todas las formas topológicas usadas en un documento dado.

Por otra parte, la librería **TNamingNamedShape** permite obtener formas a partir de las etiquetas, actualizar correctamente estas en caso de que alguna tenga algún cambio; esta librería contiene un conjunto de pares de relaciones de la forma anterior (**Old Shape**) con la nueva (**New Shape**), ver figura 1.9, además de tener referencias hacia formas y evolución de las mismas, a partir de la librería **TNamingUsedShapes**.

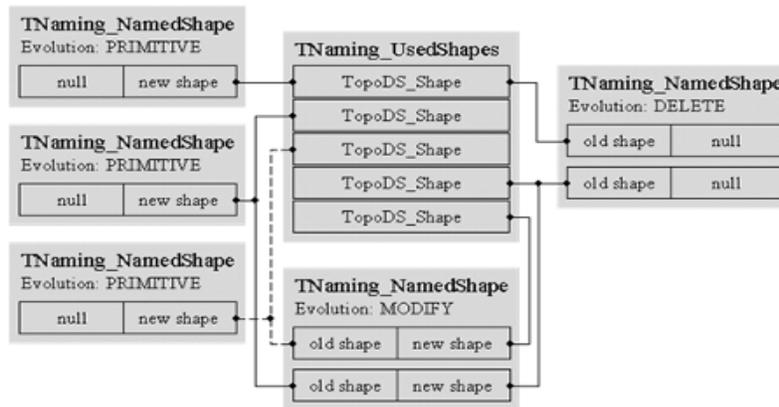


Figura 1.9. Relaciones del atributo `TNamingNamedShape`

El nombrado topológico permite que una forma mantenga su nombre interno después que se haya realizado una operación de modelado (corte, unión, etc).

1.3.5. Visualización de los atributos

Implementan los servicios interactivos de la aplicación en el contexto OCAF a partir de las librerías **AIS-Viewer** y **Presentation** perteneciente **TPrsStd**. La clase **TPrsStd-AISViewer** permite definir un atributo de visor interactivo; puede haber solo un atributo de este tipo para un marco de datos y siempre se coloca en la etiqueta raíz, por lo que puede ser establecido o encontrado por cualquier etiqueta (etiqueta de acceso) del marco de datos. Sin embargo, la arquitectura predeterminada se puede ampliar fácilmente y administrar varios visores por cada marco.

Definición un atributo de presentación

La clase **TPrsStd-AISPresentation** permite definir la presentación visual del contenido de las etiquetas de los documentos, además de varios campos visuales (color, material, transparencia, se muestra, etc.), este atributo contiene su GUID del controlador. Este GUID define la funcionalidad, que actualizará la presentación cada vez que sea necesario. La clase abstracta **TPrsStd-Driver** permite definir clases de controladores, definiendo el método de actualización en la nueva clase que reconstruirá la presentación. Si el controlador se coloca en la tabla de controladores con el GUID del controlador único, cada vez que el visor actualice las presentaciones con un GUID idéntico al GUID de el controlador, deberá llamar al método de actualización del controlador para estas presentaciones:

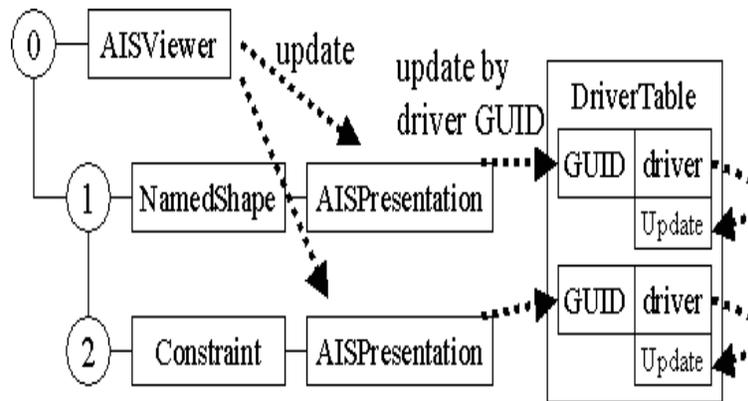


Figura 1.10. Relaciones de la presentación visual del atributo

1.3.6. Servicios de función

Agregan datos necesarios para la regeneración de un modelo. El mecanismo de funciones, disponible en el paquete **TFunction**, proporciona enlaces entre las funciones y cualquier algoritmo de ejecución, que toman sus argumentos del marco de datos y escriben sus resultados dentro del mismo. Cuando edita cualquier modelo de aplicación, tiene que volver a generar el modelo propagando las modificaciones, cada paso de propagación llama a varios algoritmos, para hacer estos algoritmos independiente del modelo de la aplicación es necesario el uso de servicios de funciones.

1.4. Análisis del sistema FreeCAD

FreeCAD es un modelador de diseño asistido por computadora en 3D. El desarrollo es completamente de código abierto (Licencia Pública General Reducida (LGPL, por sus siglas en inglés)). FreeCAD está dirigido directamente a la ingeniería mecánica y diseño de productos, pero también se emplea en una amplia gama de usos en la ingeniería, como la arquitectura (FreeCAD, 2017a), cuenta con herramientas similares a CATIA, SolidWorks o Solid Edge, y por lo tanto también cae en la categoría de Diseño Asistido por computadora Orientado a la Mecánica (MCAD, por sus siglas en inglés) y CAE; es un modelador paramétrico con una arquitectura de software modular que hace fácil proporcionar funcionalidades adicionales sin modificar el sistema central. Al igual que muchos modeladores modernos de diseño asistido por computadora en 3D tiene componentes 2D con el fin de esbozar formas en dos dimensiones o extraer los detalles del diseño del modelo de 3D para crear dibujos de producción en 2D. FreeCAD hace un uso intensivo de todas las grandes bibliotecas de código abierto que existen en el campo de la Computación Científica. Entre ellas se encuentran OpenCASCADE, un núcleo de gran alcance, Coin3D, una encarnación de Inventor abierto, Qt y Python. FreeCAD también es totalmente multiplataforma, y actualmente se ejecuta sin problemas en los sistemas Windows, Linux / Unix y Mac OSX, con el mismo aspecto y las funcionalidades exactas en todas

las plataformas. (Riegel; Juergen y Mayer, 2019)

La arquitectura de FreeCAD está enfocada de la siguiente manera:

- **Núcleo (Core):** Contiene las clases generales que definen los ajustes de aplicación, interfaces de usuario, estructuras bases usadas por otros componentes y el visor.
- **Bancos de trabajo (Workbenches):** Los bancos de trabajo son módulos que emplean el núcleo como base, son los distintos entornos de trabajo que ofrece FreeCAD.
- **Bibliotecas de terceros embebidas:** Bibliotecas adicionales que son añadidas al sistema.
- **Zipios++:** Biblioteca en C++ para la lectura y escritura de archivos *zip*.
- **PyCXX:** Conjunto de facilidades para la escritura de archivos Python en C++.
- **Macros de ayuda paramétricos:** Contiene un conjunto de módulos que definen series de instrucciones (macros) para manejar los ajustes y propiedades de determinados componentes.

Algunos módulos de FreeCAD se encuentran en fase de desarrollo: **MeshPart**, **Assembly** y **CAM**. Solo han sido definidas parcialmente hasta el momento algunas clases e interfaces de usuario. Todos los módulos constan de dos ámbitos o componentes principales:

- **App:** Donde son definidas las clases que contienen la carga lógica de la aplicación y a través de la consola o modo servidor puede funcionar.
- **GUI:** Donde se definen las interfaces de usuario específicas del módulo.

1.4.1. Núcleo

El núcleo de un sistema CAD debe contar con una serie de componentes que garanticen el cumplimiento de las funciones y características requeridas. Debido a los distintos campos en los que son aplicados los sistemas de diseño asistido es difícil establecer un modelo universal. A nivel general y sobre la base de las funciones a desempeñar, se puede establecer que los sistemas de diseño poseen al menos los siguientes componentes. (Torres, 2005):

- **Representación del Modelo:** es la representación computacional del objeto que se está diseñando, contiene toda la información necesaria para describir el objeto, tanto a nivel geométrico, como de características; es el elemento central del sistema, el resto de los componentes trabajan sobre él. Por tanto determinará las propiedades y limitaciones del sistema CAD.
- **Subsistema de edición:** permite la creación y edición del modelo, bien a nivel geométrico o especificando propiedades abstractas del sistema, en cualquier caso la edición debe ser interactiva para facilitar la exploración de posibilidades.
- **Subsistema de visualización:** se encarga de generar imágenes del modelo que permite representar gráficamente el ente que se está diseñando.
- **Subsistema de cálculo:** proporciona el cálculo de propiedades del modelo y la realización de simulaciones.

- **Subsistema de documentación:** se encarga de generar la documentación del modelo.

Las técnicas de representación y edición del modelo, como la visualización, el cálculo o la documentación, dependen del tipo de ente a modelar. FreeCAD utiliza OpenCASCADE como núcleo gráfico, permitiendo operaciones complejas en 3D, soporte nativo para conceptos como representación de límites (**brep**), curvas y superficies de spline (**nurbs**) de base racional no uniforme, una amplia gama de entidades geométricas, operaciones booleanas y filetes, y soporte integrado de formatos STEP e IGES.

1.4.2. Documento

Un documento de FreeCAD contiene todos los objetos de la escena, grupos y objetos hechos en cualquier entorno de trabajo (**workbenches**); por lo tanto, es posible cambiar entre los módulos, seguir trabajando en el mismo documento, e incluso tener datos de distintas naturalezas, es decir puede coexistir datos de pieza y datos de planos en un mismo documento. El documento contiene toda la información que se genera en los **workbenches** y es lo que se guarda en el disco en un determinado formato; se pueden abrir varios al mismo tiempo y abrir varias vistas del mismo; si hay uno o más abiertos, el que está activo es el que contiene la vista con la que se está trabajando en ese momento. (FreeCad, 2019)

Se puede observar que FreeCAD parece no seguir la misma tendencia que los sistemas comerciales, o al menos de algunos en lo que respecta al documento; por ejemplo, algunas de las aplicaciones como Inventor o Solid Edge, suelen tener varios escenarios de trabajos que parecen corresponder con módulos informáticos, los principales son los entornos de desarrollo de pieza, ensamble y el entorno de dibujo. Mediante la observación se puede constatar que cuando el usuario abre un documento, solamente contiene las funcionalidades de uno de los módulos. El módulo de pieza abre un documento de pieza, el de ensamble uno de tipo ensamble y así respectivamente, lo cuál es una diferencia circunstancial con lo que se ha descrito de FreeCAD anteriormente. La ventaja de FreeCAD es precisamente que se puede contener en un mismo documento todo el proceso de modelado, pero si estos modelos son grandes, pueden constituir un problema a manejar demasiado datos, además está la interrogante ¿por qué los sistemas comerciales no lo hicieron de esta manera?, todo parece indicar que lo más eficiente a trabajar con diferentes documentos es tener un tipo de documento por cada uno de estos. Por tanto, como parte de una recomendación, es que el documento se deba desarrollar independientemente de otros, se debe crear un documento experimental para ensamble y para planos.

1.4.3. Aplicación e interfaz de usuario

Como casi todo lo demás en FreeCAD, la interfaz de usuario (GUI) del entorno de trabajo de piezas está separada de la aplicación base de piezas. Los documentos también están divididos en dos partes: el documento de la aplicación, que contiene los objetos, y el documento vista, que contiene la representación en pantalla de los objetos. (*ibíd.*)

1.4.4. Visualización

La visualización es la formación de imágenes, se define como el mapeo de datos en representaciones que pueden ser percibidas. Los tipos de mapeo pueden ser visuales, auditivos, táctiles, etc. o una combinación de estos. (Roldán, 2003)

Para la visualización y el renderizado FreeCAD utilizan OpenInventor, Coin3D y VTK. Según los desarrolladores no utilizaron la visualización de OpenCASCADE ya que era carente en el tiempo que FreeCAD se estaba desarrollando, cuando esta mejoró significativamente ya era demasiado tarde para FreeCAD, otros problemas eran que la visualización de OpenCASCADE está altamente diseñada para la visualización de sus tipos, no está disponible una interfaz general para cosas arbitrarias y, por lo tanto, no era adecuada para FreeCAD, donde muchos espacios de trabajos dibujan todo tipo de cosas a través de API OpenInventor. (FreeCAD, 2017b)

VTK

VTK es un kit de herramientas de visualización, posee un sistema orientado a objetos para el procesamiento y la visualización de imágenes en tres dimensiones; esta es disponible para gráficos 3D, modelado, procesamiento de imágenes, renderizado volumétrico, visualización científica y visualización de información; incluye varias capas de interfaz interpretadas, tales como Tcl/Tk, Java y Python; soporta una amplia variedad de algoritmos de visualización: 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 son integrados en el sistema, lo cual permite la mezcla de datos y algoritmos gráficos de imágenes 2D/3D. (VTK, 2019)

Open Inventor y Coin3D

Open Inventor es una API de gráficos 2D y 3D orientada a objetos, destinada a compañías de software que quieran hacer uso de un framework 3D para construir y vender aplicaciones de alto rendimiento, está escrita en C++ y OpenGL, provee una capa de alto nivel de programación. Por su parte, Coin3D es un software libre completamente compatible con la API de Open Inventor versión 2.1. (Coin3D, 2019)

La selección de OpenInventor, según los desarrolladores, para el renderizado en FreeCAD, está basado en la licencia de software y el rendimiento, además de ser C++ orientado a objeto para gráficos 3D y diseño de aplicaciones de interfaces. (FreeCAD, 2017b)

1.4.5. Uso de OCAF en FreeCAD

Una de las principales interrogantes en esta investigación, es el por qué no se utilizó OCAF en el desarrollo de la aplicación. Respecto a esto, Jürgen Riegel, uno de los principales desarrolladores realiza esta aclaración en el forum del FreeCAD (Forum, 2009), el cuál cito: (...) *básicamente, hubo una versión de*

*FreeCAD alrededor de 2005 que utiliza el marco OCAF, pero en este momento había una serie de desventajas, por ejemplo: no fue posible cargar complementos en tiempo de ejecución. Otro problema fue que no había ningún algoritmo de resolución entregado para resolver el problema de dependencias de la solución de grafo. Entonces, lo que queda es más o menos algunas clases bases y cosas de contenedor, esto no es una gran ganancia. El marco de la función deshacer estaba bien y lo mejor era el **TNaming** para identificar una geometría después de realizar un cambio. ¡Me di por vencido! Decidí usar marcos maduros como **Xercesc**, **boost**, **python** y otras cosas que hacen el trabajo. Nunca me arrepentí hasta ahora. La solución ahora implementada en FreeCAD es **IMO** mucho mejor, es más orientado a objetos y tiene una integración de **Python** más profunda (y más fácil), admite la carga perezosa (tardía) de módulos y la división de aplicaciones y aplicaciones de GUI (Interfaz gráfica de usuario). Se tuvieron en cuenta además de algunos **bugs** que tiene **OCCT** (...)*

1.4.6. Método para guardar el historial de modificaciones a un modelo (History Based Feature)

Todos los sistemas de diseño modernos están basados en el paradigma diseño de rasgos (**feature based design**) también llamado diseño paramétrico (**parametric design**) o basado en historial (**history based design**) (Spitz y Rappoport, 2004).

Los sistemas CAD de la quinta generación son también llamados sistemas CAD basados en historial, son capaces de capturar el intento original del diseño del usuario, el software recuerda y fortalece relaciones entre objetos construidos por el diseñador. A medida que el usuario trabaja, el software construye árbol de historial basados en características, el cual lleva todo el proceso de las relaciones y parámetros, y almacena el orden en el cual el usuario crea una característica. Una vez que la parte es construída, el usuario solamente necesita escribir en variables para cambiar un modelo preprogramado. (Tornincasa y Monaco, 2010)

El paradigma de diseño seguido por la generación anterior de sistemas de CAD, se basaba en el modelado de sólidos partiendo de objetos primitivos o generados por barrido, transformaciones lineales y la utilización de las operaciones booleanas (modelado clásico), el procedimiento seguido por el usuario en este paradigma, empieza por crear un objeto base, realizando instancia de primitivas o por barrido, y después se crean los objetos que intervendrán en las operaciones booleanas para realizar uniones, intersecciones o diferencias. El uso de las transformaciones geométricas permite posicionar los objetos en el espacio las diferentes partes del diseño. (Espinoza y Carrera, 2015)

En FreeCAD se pueden realizar operaciones a través del árbol, se pueden crear grupos, mover objetos a grupos, cambiar el nombre o eliminarlos.

Comparación entre el árbol de FreeCAD con el de Inventor

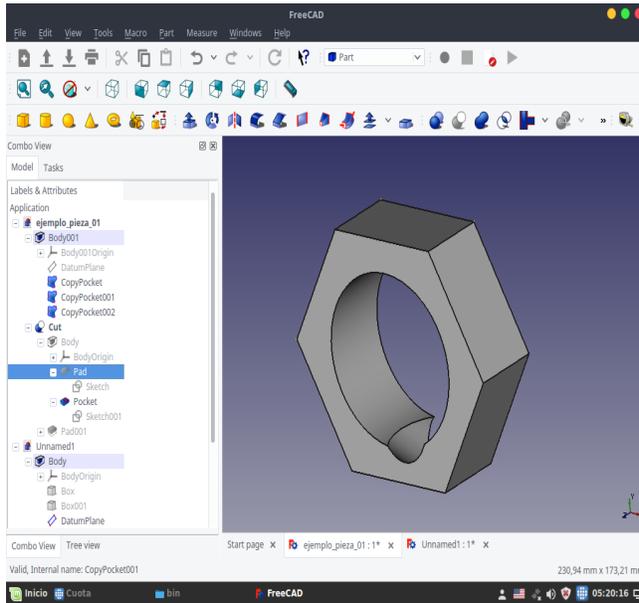


Figura 1.11. Árbol de FreeCAD

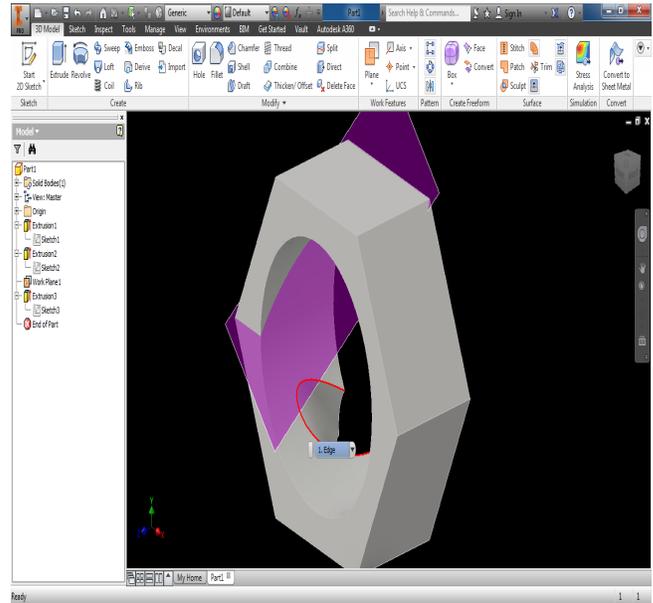


Figura 1.12. Árbol de Inventor

En las figuras 1.11 y 1.12 se muestra en un panel a la izquierda el historial de transformaciones que va sufriendo una pieza durante el proceso de diseño en dos aplicaciones: FreeCAD de software libre e Inventor. Lo primero que se realiza es la pieza, esta no desaparece sino que se van agregando rasgos; esto resulta beneficioso porque va mostrando todas las adiciones que se hace a la misma. En el caso de FreeCAD lo que ocurre es que en la medida que se va realizando una nueva transformación la anterior queda dentro de la última, esto no resulta tan explícito como en el caso de Inventor, ya que altera la secuencia en el proceso de modelado, es por esto, que una de las cuestiones a tomar en cuenta en el proceso de desarrollo de la aplicación Ingeniero, es la necesidad de modificar la forma en que se muestra las transformaciones en el árbol de la aplicación. En ambas vistas está hecha la misma pieza con la misma secuencia de pasos.

1.4.7. Archivos de intercambio

Muchos sistemas CAD de ingeniería mecánica proporcionan el intercambio de datos entre sus propios módulos de aplicación, estos permiten una descripción geométrica de una pieza modelada que pueden ser pasado por un módulo de visualización. Estos sistemas a menudo permiten secciones y proyecciones de modelos sólidos y ser importados a un programa donde se puede crear dibujos mecánicos tradicional. Este intra-sistema de intercambio de datos es posible porque el vendedor tiene control completo sobre la estructura de datos. La mayoría de los proveedores de CAD también proporcionan un formato externo para aquellos que desean interactuar con la base de datos de sistemas extranjeros (como DXF para AutoCAD y SIF para Intergraph). (TURNER, 1992)

El intercambio de datos es uno de los problemas más importante en modelado de sólido y un problema extremadamente significativo tanto científico como comercial. Científicamente el intercambio de archivo en diseño paramétrico basado en rasgos es un reto en términos de semánticas, algoritmos y estructura de datos de operaciones geométricas. (Rappoport, 2003)

A continuación se expone las características de los formatos nativos para entender mejor el uso y la diferencia que existe entre ellos. Cabe destacar que el sistema FreeCAD logra exportar e importar en cada uno de estos:

BREP

El formato BREP (Boundary Representation) se utiliza para almacenar modelos 3D que consta de vértices, aristas, alambres, caras, carcasas, sólidos compuestos, triangulaciones de aristas, triangulaciones de caras, polilíneas, ubicación y orientación del espacio. Cualquier conjunto de dichos modelos puede almacenarse como un modelo único que es un compuesto de los modelos.(Technology, 2013)

La estructura de este formato consiste en:

<Tipo de contenido>;

<Version>;

<Localización>;

<Geometría>;

<Formas>

El formato se describe en un orden que es conveniente para la comprensión y no en el orden en que las partes del formato se siguen unas a otras.

El formato BREP utiliza los siguientes términos BNF:

- <n>: es la secuencia de caracteres ASCII dependiente del sistema operativo que separa las cadenas de texto ASCII en el sistema operativo utilizado.
- <- n>: = * <n>; <->: = +; Es una secuencia no vacía de caracteres de espacio con código ASCII 21h. <flag>: = "0"|"1";
- <int>: es un número entero de -2^{31} a $2^{31} - 1$ que está escrito en el sistema denario; <real>: es un real de -1.7976931348623158 a 1.7976931348623158 que se escribe en formato decimal o E con base 10. El punto se utiliza como un delimitador de las partes enteras y fraccionarias; <Punto 2D>: = <real><-><real>; <Punto 3D>: = <real><-><real>>2; <Dirección 2D>: es un <punto 2D>x y, de modo que $x^2 + y^2 = 1$; <Dirección 3D>: es un <punto 3D>x y z, de modo que $x^2 + y^2 + z^2 = 1$;
- <+>: Es una operación aritmética de suma.

IGES

Especificación de intercambio gráfico inicial (IGES por sus siglas en inglés), fue diseñado para lograr la interoperatividad entre diferentes sistemas CAD, este puede representar modelos CAD con precisión. Tanto

modelos sólidos constructivos y métodos de representación acotada (B-Rep) son introducidos en la versión 5.0. Es implementado para casi cualquier sistema CAD/CAM comercial. (Kai; Jacob y Mei, 1997)

Este estándar define las siguientes entidades:

- Geometría en 2D y 3D.
- Bounded Geometry.
- Boundary Representation.
- Assemblies.
- Colores.
- Capas.
- Nombres.

Esta especificación define la estructura de archivos y los formatos de lenguaje para representar elementos geométricos, topológicos, y datos de definición de productos no geométricos. Estos formatos son independientes del método de modelado utilizados, y admiten el intercambio de datos mediante medios físicos o protocolos de comunicación electrónica (definido en otras normas).

STEP

El estándar para el intercambio de productos fue diseñado para encontrar las necesidades de tener un conjunto común de herramientas CAD/CAM dentro de compañías usando un formato neutral para el intercambio, con el objetivo de reducir el número de traducciones necesarias. La tecnología OpenCASCADE da la posibilidad de crear archivos de acuerdo a partes de STEP (AP203 y AP214).

- **AP203:** Esta especifica el protocolo de aplicación para la configuración controlada del diseño tridimensional. Está relacionada con la transferencia de productos de modelos de formas, estructura de ensambles, y control de la información de configuración, tanto para estatus de partes de versiones como para liberación del producto.
- **AP214:** Especifica datos núcleos para el proceso de diseño mecánico automático. Es una extensión de AP203 y también maneja herramientas de manufactura, tolerancia al dato, control de datos numéricos, dibujos 2D y manejo de la información del dato del producto.

El formato STEP es bien soportado en cualquier software CAD en 3D. Provee un mecanismo para describir un producto de dato a partir de cualquier sistema en particular.

STL

El formato se archivo STL es una representación poliédrica de un diseño con facetas triangular. Es generado a partir de un modelo preciso de un sistema CAD por un proceso conocido como mosaico, la cuál genera triángulos para aproximar el modelo CAD. El archivo STL tiene dos formatos: el ASCII y el binario. El primero es más grande que el binario y más entendible a la hora leerlo. En un archivo STL, las facetas

triangulares son descritas por un conjunto de coordenadas X,Y,Z por cada uno de los tres vértices y un vector normal unitario para indicar el lado de la faceta (ver figura 1.13) (Kai; Jacob y Mei, 1997)

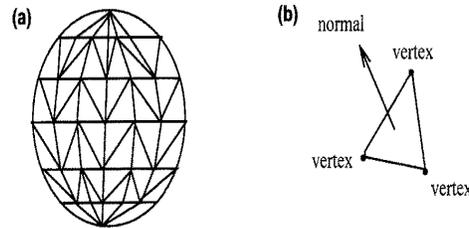


Figura 1.13. Modelos de facetas. (a) Archivo STL. (b) Una faceta.

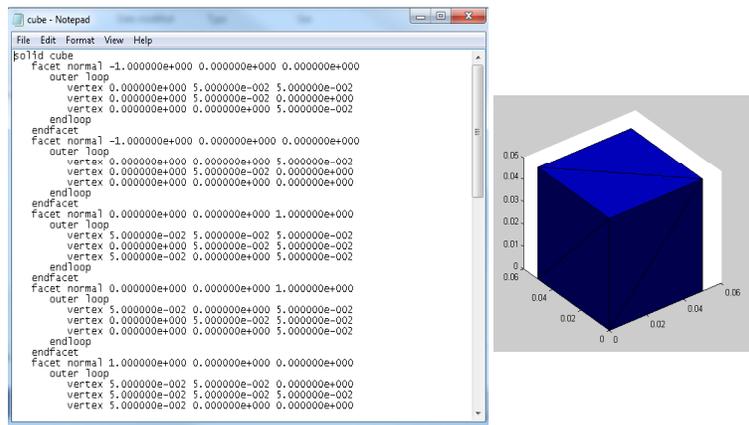


Figura 1.14. Formato STL de un cubo

1.5. Conclusiones del capítulo

En este capítulo se hizo un análisis de los sistemas comerciales *Inventor*, *Catia*, *Solid Edge* y *SolidWorks* se identificaron aspectos y tendencias que los hacen líderes en los mercados. Se realizó un estudio del marco para aplicaciones de OpenCASCADE como su arquitectura y las funcionalidades que este provee. Se caracterizó el sistema de código abierto FreeCAD donde se describieron aspectos de su arquitectura y formatos de guardado.

Metodología y tecnologías para el proceso de desarrollo

El proyecto **Ingeniero** es el resultado de los estudios realizados por el grupo de investigación SIPII, en este capítulo se exponen las herramientas utilizadas en la propuesta de solución tomando como base estos estudios en el desarrollo del sistema.

2.1. Lenguaje de desarrollo

C++ es un lenguaje compilado e imperativo orientado a objetos. Fue diseñado a mediados de 1980 por Bjarne Stroustrup como extensión del lenguaje C, se le han añadido elementos que le permiten un estilo de programación con alto nivel de abstracción como son nuevos tipos de datos, clases, plantillas, mecanismo de excepciones, sistema de espacios de nombres, funciones *inline*, referencias, operadores para manejo de memoria persistente, y algunas utilidades adicionales de librería. Este lenguaje no fue diseñado con la computación numérica en mente, sin embargo, muchos programas de ingeniería, numéricos y científicos son realizados en C++. Todo puede programarse con ellos, desde sistemas operativos y compiladores hasta aplicaciones de bases de datos y procesadores de texto, pasando por juegos, aplicaciones a medida, etc. (Stroustrup, 2013)

Se elige este lenguaje de programación para el desarrollo del sistema CAD por las características antes mencionadas y ya que muchos sistemas, herramientas y bibliotecas están implementadas en C++, tales como Boost, Coind3D, OpenCV, Qt siendo este último también usado en grupo de investigación como **framework** (Marco de trabajo).

2.2. Framework de desarrollo Qt

Qt es un marco de desarrollo de aplicaciones multiplataforma basado en C++, dentro de las que se encuentran: Linux, OS X, Windows, VxWorks, QNX, Android, iOS, Blackberry, Sailfish OS y otras (Creator, 2018). Proporciona módulos para el desarrollo multiplataforma en las áreas de redes, bases de datos,

OpenGL, tecnologías web, sensores, protocolos de comunicación (Bluetooth, puertos serie), XML y procesamiento JSON. Qt está disponible bajo varias licencias: licencia comercial y para software libre con varias versiones de la Licencia Pública General y Licencia pública general menor (GPL, LGPL por sus siglas en inglés) (QtWiki, 2017).

Se decide el uso de Qt en la presente investigación por ser un **framework** para el desarrollo de aplicaciones basado en C++ y por poseer una gran cantidad de librerías para disímiles tareas como son el uso de las interfaces.

2.3. Open Cascade Technology

La Tecnología OpenCASCADE (OCCT, por sus siglas en inglés), es una plataforma de desarrollo de software que proporciona servicios para superficies 3D y modelado de sólidos, el intercambio de datos, y la visualización. La mayor parte de la funcionalidad OCCT se encuentra disponible en forma de bibliotecas de C ++. OCCT puede ser aplicado en el desarrollo de software cuyo objetivo sea el modelado en 3D, fabricación / medición (Fabricación Asistida por Computadora (CAM, por sus siglas en inglés)) o simulación numérica (CAE). Es una tecnología de software libre; se puede distribuir y/o modificar bajo los términos de la Licencia Pública General de GNU (LGPL) versión 2.1, con excepción adicional (OpenCASCADETehnology, 2017).

Alternativamente, OpenCASCADE puede ser utilizada bajo los términos de licencia comercial o acuerdo contractual. Está diseñada para ser altamente portátil y es conocida por trabajar en una amplia gama de plataformas (UNIX, Linux, Windows, Mac OS X, Android). La versión (7.0.0. beta) está certificada oficialmente en las plataformas Windows (IA-32 y x86-64), Linux (x86-64), MAC OS X (x86-64) y Android (4.0.4 ARMv7).

OCCT permite el desarrollo de aplicaciones en modelados geométricos en 2D o 3D. Las librerías OCCT están agrupadas en seis módulos los cuales son usados por **TGeoCad**, los cuales son:

- **Clases Fundamentales:** Son usados en la descripción de formas geométricas elementales (puntos, vectores, líneas, círculos, cónicos planos) para el posicionamiento de goemtrías en el espacio o en un plano vía de un axis o coordenadas del sistema, y para la definición de transformaciones geométricas tales como traslaciones, rotaciones y simetrías.
- **Modelado de Datos:** Permite construir estructuras de datos puramente topológicas definiendo relaciones entre entidades geométricas simples y dando la posibilidad de asignar a una forma una orientación y localización. Usando topológicos abstractos en estructuras de datos cada forma puede ser dividida en varios componentes topológicos como en la figura 2.1

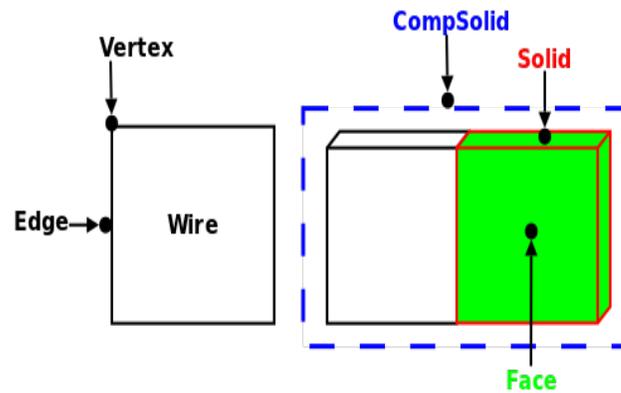


Figura 2.1. Estructura de dato topológico

- **Algoritmo de modelado:** Provee algoritmos geométrico y topológicos para el creado de vértices , aristas, caras y sólidos. Da la posibilidad para directamente construir objetos primitivos tales como cajas, wedges y objetos rotacionales.
- **Módulo de intercambio extendido de datos (XDE):** Permite escribir aplicaciones basadas en la tecnología de OpenCASCADE hacia archivos de formatos IGES y/o STEP (AP203, AP214).
- **OCAF:** Framework para el desarrollo rápido de aplicaciones usados para especificar y organizar datos de acuerdo a cuál estructura de dato son organizados a través de un modelo de referencia. (Luzzi y Carminati, 2017)

Opencascade Community Edition (OCE, por sus siglas en inglés) es una versión de OCCT en la que la comunidad del software libre aporta sus experiencias y recomendaciones de optimización a las versiones liberadas mediante foros o la página oficial de desarrollo de esta tecnología. Se decide el empleo en la presente investigación de OCE por todas las características antes mencionadas de la tecnología de OpenCASCADE, por ser de software libre y poseer un acelerado desarrollo por parte de la comunidad, evidenciándose su uso en la mayoría de los sistemas CAD de código abierto, siendo establecida como una tecnología a emplear en el grupo de investigación **SIPIL**.

Intercambio de datos extendido (XDE)

La base de XDE, llamada XCAF, es un marco basado en OCAF (OpenCASCADE Application Framework) y está destinado a ser utilizado con ensamblajes y con varios tipos de datos adjuntos (atributos), los atributos pueden ser individuales para una forma, especificando algunas características, o pueden ser de agrupación, especificando que una forma pertenece a un grupo dado. XDE trabaja en un documento OCAF con una aplicación específica definida en un módulo XCAF dedicado, varias funciones de XDE utilizan esta organización para intercambiar datos estandarizados que no sean formas y geometría. (OpenCASCADE, 2019b)

Los elementos básicos utilizados por XDE se introducen en el submódulo XCAF mediante el paquete XCAF-

Doc, estos elementos consisten en descripciones de estructuras de datos comúnmente utilizadas (aparte de las formas en sí mismas) en intercambios de datos normalizados. No se adjuntan a aplicaciones específicas ni traen semánticas específicas, sino que están estructuradas de acuerdo con el uso y las necesidades de los intercambios de datos. El documento utilizado por XDE generalmente comienza como un documento *TDocStd_Document*.

En la propuesta de solución además de usar algunas librerías de OpenCASCADE se fue necesario utilizar XDE a través de **XCAFAApp_Application.hxx** con el objetivo gestionar el guardado en los formatos de OCAF. Al utilizar OCE hubieron problemas con los *plugin* asociados a OCCT haciendo que librerías como **TDocStd_Application.hxx** no pudieran gestionar el guardado en los formatos de OCAF.

2.4. Sistema de control de versiones (GitLab)

Un sistema de control de versiones es una combinación de tecnologías y prácticas para seguir y controlar los cambios realizados en los ficheros del proyecto, en particular en el código fuente, en la documentación y en las páginas web. El sistema de control de versiones permite a una fuerza coordinadora central abarcar todas estas áreas, consecuentemente combina procedimientos y herramientas para gestionar diferentes versiones de objetos de configuración que se crean durante el proceso del software.(Sadaña, 2007). Se elige Gitlab proporcionar a los equipos un único almacén de datos, una interfaz de usuario y un modelo de permiso en todo el ciclo de vida de DevOps (Desarrollo y Operaciones), lo que permite a los equipos colaborar, reduciendo significativamente el tiempo de ciclo y centrándose exclusivamente en crear software de gran calidad rápidamente (GitLab, 2018). Además de esto se puede ejecutar acciones como son las siguientes.

- **Conmutación de contexto sin fricción:** Crear una rama para probar una idea, comience varias veces, cambie de nuevo a la rama de donde se ramificó, aplique un parche, cambie de nuevo a la rama donde está experimentando y fijarlo.
- **Reglas Basadas en el Rol:** Tener una rama que siempre contiene solo lo que va a la producción, otra que se fusiona en el trabajo para la prueba, y varias más pequeñas para el trabajo diario.
- **Flujo de trabajo basado en funciones:** Crear nuevas ramas para cada nueva función en la que esté trabajando, de modo que pueda cambiar sin problemas entre ellas y, a continuación, suprimir cada una de las ramas cuando se fusione en su línea principal.
- **Experimentación desechable:** Crear una rama para experimentar, darse cuenta de que no va a funcionar, y solo eliminarlo, abandonar el trabajo.(TORVALDS y HAMANO, 2010)

Durante el desarrollo de la propuesta de solución debido a las características antes mencionadas y a las indicaciones del proyecto se emplea GitLab.

El proyecto CAD del grupo SIPII tiene dos ramas fundamentales:

- **Kernel:** Contiene componentes y librerías principales del núcleo del sistema.
- **AllModuleDev:** Contiene los demás módulos funcionales que se han desarrollado hasta el momento.

Cada vez que se realiza un cambio en el proyecto, se va a tener un control de las clases agregadas o modificadas, esto garantiza un buen seguimiento y control del desarrollo del sistema **Ingeniero**.

2.5. Doxygen

Es la herramienta estándar para generar documentación a partir de fuentes C++ anotadas, pero también es compatible con otros lenguajes de programación populares como C, Objective-C, CSharp, PHP, Java, Python, IDL, Fortran y hasta cierto punto D.

Beneficios de Doxygen:

- Puede generar un navegador de documentación en línea (en HTML) y / o un manual de referencia a partir de un conjunto de archivos fuente documentados. También hay soporte para generar resultados en RTF (MS-Word), PostScript, PDF con hipervínculos, HTML comprimido y páginas de manual Unix. La documentación se extrae directamente de las fuentes, lo que hace que sea mucho más fácil mantener la documentación coherente con el código fuente.
- Se puede configurar doxygen para extraer la estructura de código de archivos de origen no documentados. Esto es muy útil para encontrar rápidamente su camino en grandes distribuciones de fuentes. Doxygen también puede visualizar las relaciones entre los distintos elementos mediante gráficos de dependencia, diagramas de herencia y diagramas de colaboración, que se generan automáticamente.
- También se puede usar doxygen para crear documentación estándar. (Doxygen, 2018)

Se ha usado esta tecnología en varias ocasiones en el sistema CAD **Ingeniero** ya que nos basados en el sistema FreeCAD, y se fue necesario entender la manera en que estaba distribuída las clases y las dependencias que hay entre estas, brindándonos un mejor entendimiento.

2.6. Visual Paradigm

Es una herramienta de Ingeniería de Software Asistida por Computación (CASE, por sus siglas en inglés). La misma propicia un conjunto de funcionalidades que apoyan al desarrollo de programas informáticos, desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación.

Visual Paradigm ha sido concebida para soportar el ciclo de vida completo del proceso de desarrollo del software a través de la representación de todo tipo de diagramas, constituye una herramienta privada disponible en varias ediciones, cada una destinada a satisfacer diferentes necesidades: Enterprise, Profesional, Community, Standard, Modeler y Personal. Existe una alternativa libre y gratuita de este software, la versión Visual Paradigm UML 8.0 Community Edition. (Paradigm, 2018)

Esta versión fue la utilizada en la propuesta de solución para el diagrama de clases, entre sus principales características están:

- Disponibilidad para las plataformas Windows y Linux.
- Diseño centrado en casos de uso y enfocado al negocio que generan un software de mayor calidad.
- Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
- Capacidades de ingeniería directa e inversa.
- Modelo y código que permanece sincronizado en todo el ciclo de desarrollo.

Visual Paradigm es la herramienta de ingeniería a emplear por la amplia gama de posibilidades que ofrece, acelerando no solo el proceso de desarrollo de software sino que facilita la documentación del mismo. En la UCI es empleada tanto en la docencia como en la producción.

2.7. CLOC

Count Lines Of Code (contar las líneas de códigos), como su nombre lo indica, es una herramienta capaz de contar las líneas en blanco, comentarios y líneas físicas de código fuente en muchos lenguajes de programación, puede calcular diferencias en blanco, comentario y líneas de origen; está escrito en su totalidad en Perl sin dependencias fuera de la distribución estándar de Perl v5.6 y superior (el código de algunos módulos externos está integrado en *cloc*) y, por lo tanto, se ejecuta en muchas versiones de Linux, FreeBSD, NetBSD, OpenBSD, Mac OS X, AIX, HP-UX, Solaris, IRIX, z/OS y Windows.

Se elige esta herramienta dadas a algunas características las cuales son: (CLOC, 2019)

- Maneja los nombres de archivos y directorios con espacios y otros caracteres inusuales.
- Permite que los resultados de varias ejecuciones se sumen por lenguaje y por proyecto.
- Puede leer las definiciones de comentarios de un archivo y, por lo tanto, trabajar potencialmente con lenguajes informáticos que aún no existen.

2.8. Metodología de desarrollo

Una metodología es un conjunto integrado de técnicas y métodos que permite abordar de forma homogénea y abierta cada una de las actividades del ciclo de vida de un proyecto de desarrollo. Es un proceso de software detallado y completo. Una definición estándar de metodología puede ser el conjunto de métodos que se utilizan en una determinada actividad con el fin de formalizarla y optimizarla. La metodología para el desarrollo de software es un modo sistemático de realizar, gestionar y administrar un proyecto para llevarlo a cabo con altas posibilidades de éxito. Comprende los procesos a seguir sistemáticamente para idear, implementar y mantener un producto de software desde que surge la necesidad del mismo hasta que se cumple el objetivo por el cual fue creado. En una metodología aplicada a la ingeniería del software se puede destacar que:

- Optimiza el proceso y el software.
- Proporciona métodos que guían en la planificación y en el desarrollo del software.

- Define qué hacer, cómo y cuándo durante todo el desarrollo y mantenimiento de un proyecto.

Algunos de los elementos son:

- **Fases:** tareas a realizar en cada fase. Productos: entradas y salidas de cada fase, documentos.
- **Procedimientos y herramientas:** apoyo a la realización de cada tarea.
- **Criterios de evaluación del proceso y del producto:** permiten determinar si se han logrado los objetivos. (LABORATORIO, 2009)

Al no existir una metodología de software universal, ya que toda metodología debe ser adaptada a las características de cada proyecto (equipo de desarrollo, recursos, etc.) exigiéndose así que el proceso sea configurable. Se decidió hacer una variación de la metodología AUP (Proceso Unificado Ágil), de forma tal que se adapte al ciclo de vida definido para la actividad productiva de la UCI.

- **Fases:** La Metodología AUP variante UCI contiene tres fases las cuales son inicio, ejecución y cierre porque así se adapta mejor al ciclo de vida de los proyectos de la UCI.
- **Disciplinas:** La AUP variante UCI propone siete las cuales son modelado de negocio, requisitos, análisis y diseño, implementación, pruebas internas, pruebas de liberación y aceptación. Para la gestión de la variación UCI se cubren con las áreas de procesos que define CMMI-DEV v1.3 para el nivel 2, serían gestión de la configuración, planeación de proyecto y monitoreo y control de proyecto.
- **Roles:** AUP propone 9 roles (Administrador de proyecto, Ingeniero de procesos, Desarrollador, Administrador de Base de Datos, Modelador ágil, Administrador de la configuración, Stakeholder, Administrador de pruebas y Probador), se decide para el ciclo de vida de los proyectos de la UCI tener 11 roles, manteniendo algunos de los propuestos por AUP y unificando o agregando otros. (SÁNCHEZ, 2015)

AUP-UCI define cuatro escenarios en los que se puede ubicar el desarrollo de una aplicación de acuerdo a sus características.

Siguiendo la política de desarrollo de software de la institución, se define como metodología a emplear la AUP-UCI en el escenario número 4 debido a la necesidad de una metodología que responda con facilidad a los cambios continuos, por estar el cliente siempre acompañando el desarrollo

2.9. Conclusiones del capítulo

Una vez identificadas las herramientas a utilizar para la propuesta de solución se puede concluir que los sistemas CAD requieren de tecnologías que le permitan realizar modelados geométricos y tratamiento de los datos, necesitan un lenguaje que pueda gestionar el gran uso de memoria con rapidez; y al ser estos proyectos tan grandes les es necesario a los desarrolladores tener el control de las versiones y cambios realizados de los mismos.

En este capítulo se exponen los elementos referentes al diseño e implementación de la propuesta de solución, estos son: Requisitos funcionales y no funcionales, arquitectura (estilos), patrones de diseño, diagrama de clases y detalles de implementación.

3.1. Descripción de las funcionalidades

Las funcionalidades serán diseñadas para la integración de la arquitectura OCAF en el proyecto **Ingeniero** que está desarrollando el grupo de investigación Soluciones Informáticas para la Ingeniería y la Industria (**SIPII**), la cual poseerá la opción de crear un nuevo documento de tipo OCAF; al activarse se crea nuevo documento, se la adjuntará el visor de OpenCASCADE donde el usuario podrá insertar las primitivas.

- El botón *New Ocaf Part Document* permite crear un documento OCAF para las funcionalidades de la sesión de trabajo para el módulo **Part**.
- Los íconos correspondientes a la creación de las primitivas (Esfera, Cono, Cubo, Torus, Cilindro), mostrará un diálogo donde se introducirán los valores de acuerdo a los parámetros de la creación de las mismas, además de su posición en las coordenadas X, Y, Z; Al presionar aceptar se mostrarán en el visor.
- El botón *Undo Ocaf*, con su respectivo atajo por el teclado (ctrl + z) volverá a un estado anterior en el proceso de diseño.
- El botón *Redo Ocaf*, con su respectivo atajo por el teclado (ctrl + y) volverá a un estado posterior en el proceso de diseño.
- El botón *Export Ocaf*, abrirá una ventana donde el usuario seleccionará donde desea exportar la información contenida en el documento. Los formatos para la importación son **IGES**, **BREP**, **STL**, **STEP**.
- El botón *Save Ocaf* salvará la sesión de trabajo en algunos de los formatos de OCAF (XmlOcaf, BinOcaf, MTDV-Standard).

En caso de que el usuario quiera acceder a algunas de estas funcionalidades sin tener un documento de tipo OCAF activo se le mostrará un mensaje diciendo que estas funcionalidades pertenecen a un documento de este tipo.

3.2. Estándar de codificación

Un estándar de codificación comprende todos los aspectos de la generación de código. Un código fuente completo debe reflejar un estilo regular, como si se hubiese realizado por un programador. La forma usada va a depender de la facilidad para entender el código y retomar ciertas partes realizadas por otros integrantes, así como la eliminación de las mismas. La elaboración y utilización de estos estándares permiten a todos los desarrolladores realizar la implementación siguiendo las pautas predefinidas y así el resto de los desarrolladores puedan entender el código, facilitando que un proyecto de software se convierta en un sistema legible, uniforme y fácil de mantener.(ASLP, 2019)

Descripción	Ejemplo
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.	<pre>class Foo{ cuerpo de la clase } class FooFirst{ cuerpo de la clase }</pre>
Siempre se declara para todas las clases implementadas su respectivo destructor de clase.	<pre>virtual ~Foo()</pre>
La declaración de funciones o métodos siempre comenzarán en letra inicial minúscula. En caso de ser un nombre compuesto se regirá por la normativa CamelCase-lowerCamelCase.	<pre><Tipo dato retorno> funcion() <Tipo dato retorno> funcionCompuesta() <Tipo dato retorno>funcionDobleCompuesta()</pre>

Figura 3.1. Estándar de codificación

Los atributos siempre estarán escritos con letra minúscula. En caso de ser un nombre compuesto se regirá por la normativa CamelCase-lowerCamelCase.	<Tipo dato> atributo; <Tipo dato> atributoNombreCompuesto;
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.	<Tipo dato retorno> funcion(<tipo><id1>, <tipo><id2>, <tipo><idN>)
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.	Clase(<tipo><id1>, <tipo><id2>, <tipo><idN>)
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	x + y; x == y; idFuncion.miembro(); idFuncion->miembro(); array[];

Figura 3.2. Estándar de codificación

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 <i>framework</i> de Qt.	Para las estructuras if, else, if else : <estructura control>{condición}{ tarea a ejecutar } Para los bucles while, for, do while y otros: <bucle>{condiciones}{ tarea a ejecutar }
Comentarios en el código según el estándar de C++.	
Comentarios pequeños.	/* comentario sencillo */
Otros comentarios.	/* *Comentario */
Comentario de versión, descripción de clase y otras características de la clase o paquete.	/* ***** *Comentario amplio* ***** */

Figura 3.3. Estándar de codificación

En la figura 3.4 se muestra un ejemplo de código empleando el estándar de codificación *CamelCase*

- RF4. Rehacer cambios realizados.
- RF5. Exportar en los formatos de intercambio **IGES, STEP, STL, BREP**.
- RF6. Salvar el documento en los formatos de OCAF (XmlOcaf, BinOcaf, MTDV-Standard).

3.3.2. Requisitos No Funcionales

A continuación, se exponen los requisitos no funcionales de la propuesta do solución.

- RNF1. Software: SO.: GNU-Linux, GCC: 4.2.4.
- RFB2. Restricciones de diseño e implementación: Arquitectura de OCAF, lenguaje de programación C++, biblioteca OpenCASCADE, Coin3D, OpenGL.

3.4. Historias de usuario

Las historias de usuario (HU) son tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. El tratamiento de las historias de usuario es muy dinámico y flexible, en cualquier momento se pueden modificar, reemplazar por otras más específicas o generales o añadirse nuevas. Cada historia de usuario debe ser lo suficientemente comprensible y delimitada para que los programadores puedan implementarla (Anderson; Hendrickson y Jeffries, 2001)

La HU presenta la siguiente estructura:

- Número: A cada HU se le asigna un número para facilitar su identificación.
- Nombre: Nombre descriptivo de la HU.
- Prioridad: Grado de prioridad que se le asigna a la HU en dependencia de las necesidades del cliente, (Alta, Media o Baja).
- Complejidad: Grado de complejidad que se le asigna a la HU luego de ser analizada. (Alta, Media o Baja).
- Puntos Estimados: Unidades de tiempo estimadas por el equipo de desarrollo para darle cumplimiento a la HU a través de puntos.
 - **Complejidad:** Cuanto más compleja sea una historia más puntos historia se le aplicarán.
 - **Esfuerzo:** Una historia que implique un mayor esfuerzo asumirá más puntos historia que una que suponga un esfuerzo menor.
 - **Incertidumbre:** Se refiere a la posibilidad de que se produzca una situación inesperada, por falta de información previa. Una historia tendrá más puntos historia cuanto mayor sea su incertidumbre.
Estos valores van variar de 0 a 1.

- Iteración: Número de la iteración en la cual será implementada la HU.
- Descripción: Descripción simple sobre lo que debe hacer la funcionalidad en cuestión.
- Información Adicional: Breve información que ayude a comprender algún dato no especificado antes. (PENADÉS, 2013)

Las historias de usuario se encuentran en el Apéndices A.

3.5. Diseño

La esencia del diseño del software es la toma de decisiones sobre la organización lógica del software. Algunas veces, se representa esta organización lógica como un modelo en un lenguaje definido de modelado tal como UML y otras veces simplemente se utiliza notaciones informales y esbozos para representar el diseño (Sommerville, 2006). El proceso de diseño tiene asociado la decisión del tipo arquitectura y los patrones de diseño que empleará el sistema, así como la confección de distintos diagramas que favorezcan el trabajo en la fase de implementación.

3.5.1. Estilo y patrón arquitectónico del software

Un estilo arquitectónico es una transformación impuesta al diseño de todo un sistema. El objetivo es establecer una estructura para todos los componentes del sistema. En caso de que una arquitectura existente se vaya a someter a reingeniería, la imposición de un estilo arquitectónico desembocará en cambios fundamentales en la estructura del software, incluida una reasignación de la funcionalidad de los componentes. (Pressman, 2005)

En el proyecto **Ingeniero** existen dos estilos arquitectónicos predominantes, estos son el basado en capas y el de llamada y regreso. En la propuesta de solución se añadió otro estilo arquitectónico que es el de propuesto por OCAF que es Aplicación-Documento-Atributo, que son explícitamente para sistemas CAD. Los patrones de diseño responden a necesidades o subtareas específicas de cada uno de los componentes. La integración de muchos patrones de diseño trabajando en conjunto trae como consecuencia el surgimiento de patrones compuestos.

LLamada y retorno

Es un estilo arquitectónico que permite a un diseñador de software obtener una estructura de programa que resulte relativamente fácil modificar y cambiar de tamaño. En esta categoría hay dos subestilos (*ibíd.*).

- Arquitectura de programa principal/subprograma. Esta estructura de programa clásica separa la función en una jerarquía de control donde un programa principal invoca a varios componentes de programa, que a su vez pueden invocar a otros componentes. Una aplicación CAD competente para uso general es un programa considerablemente complejo y extenso, que requiere la integración de muchos módulos. Para ello es necesario ofrecer una macro-estructura que brinde soporte a un programa

principal, que solo contenga los componentes base para el desarrollo del resto. En la imagen 3.5 se muestra un esquema que ilustra como se evidencia dicha arquitectura en **Ingeniero**.

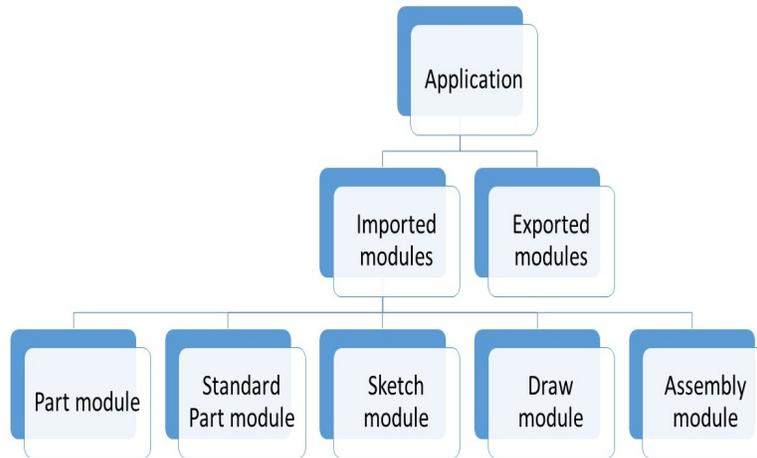


Figura 3.5. Diagrama de Arquitectura de programa principal/subprograma

La aplicación principal carga los módulos importados y exportados a través de dos subprogramas controladores, los módulos cargados son a su vez responsables de inicializar los componentes necesarios para su funcionamiento.

Arquitectura Aplicación-Documento-Atributo

Esta arquitectura es la que se utiliza en la propuesta de solución para integrar el marco de trabajo para aplicaciones al sistema **Ingeniero**, la explicación de la misma se encuentra en la fundamentación teórica. 1.3.1.

3.5.2. Patrones de Diseño

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interfaces.

Los patrones GRASP (General Responsibility Assignment Software Patterns) fueron elegidos para diseñar con éxito de estos se utilizaron:

- Experto: Es la asignación de responsabilidades a cada clase, este patrón se evidencia en la clase **TOcafApplication** que es la encargada de tener la información referente a la construcción de las primitivas.
- Creador: permite crear objetos de una clase determinada, soportando mayor claridad y encapsulamiento. Este patrón se evidencia en la clase **TOcaf_Application** la cual crea una instancia hacia una librería de OpenCASCADE para tener acceso al método de crear un documento de OCAF.

- Bajo acoplamiento: soporta el diseño de clases más independientes, que reducen el impacto de los cambios, se encarga de asignar una responsabilidad para mantener las clases lo menos relacionadas posibles. Esto se evidencia en las clases **OCCViewer** y **TOcafFunction_Driver** donde la primera se encarga de gestionar el visor de gráficos y la segunda de asignarle la función de construcción a la primitiva; por lo que estas no dependen una de la otra.
- Alta cohesión: Es la forma de medir cuán relacionadas están las responsabilidades de una clase de modo que la cohesión siga siendo alta. Este patrón está presente en **OCCViewer** y **TOcaf_Application**, la primera espera que se cree un documento gestionado por la segunda para poder asignarle el visor.

Patrones **GOF** definido en las funcionalidades:

- *Singleton*: Aplicaciones que definen varios módulos, en el que cada uno accede a un entorno visual común para propósitos distintos deben lidiar con el problema del acceso compartido. Muchas instancias de una clase siendo usada en varias secciones del programa conllevan a errores difíciles de depurar y problemas con la memoria. Para ello el “patrón de creación” Singleton se asegura de que solo exista una única instancia u objeto de determinada clase en la ejecución del programa, proveyendo un único punto de acceso al mismo. Esta es usada en la clase **TOcafApplication**.
- *Observer*: El patrón Observer entra en la clasificación de “patrones de conducta”, define una dependencia de uno a muchos entre objetos, cuando el estado de uno cambia todos los dependientes son notificados y pueden actualizarse automáticamente. (Collins, 2016). La unión entre las clases que definen la interfaz de usuario y las clases que pertenecen al nivel de aplicación son a través de este mecanismo. Este patrón se evidencia con la función *mustExecute()* perteneciente a la clase **TOcafFunctionBoxDriver** donde se leas asignan las etiquetas a los parámetros de la creación de una primitiva.

3.5.3. Diagrama de clases

Un diagrama de clases es una representación estática que describe la estructura de un sistema modelando las clases y sus métodos, atributos y relaciones.[3.6](#)

- **TDF_Label.hxx**: clase que proporciona operaciones básicas para definir una etiqueta en una estructura de datos, es una característica en la jerarquía de rasgos donde se adjuntan atributos que contienen la información de los componentes del software. La base para definir un atributo para el almacenamiento de datos de topología y nombres. Este atributo contiene dos partes:
 - El tipo de evolución, un término de la enumeración **TNaming_Evolution**.
 - Una lista de pares de formas llamada forma *antigua* y forma *nueva*. El significado depende del tipo de evolución.
- **TopoDS_Shape.xx**: librería principal utilizada para los formatos de retornos, los cuales son mostrados por el visor propio de la aplicación.
- **BRepPrimAPI_MakeBox.xx**,
BRepPrimAPI_MakeCone.hxx,
BRepPrimAPI_MakeCylinder.hxx,
BRepPrimAPI_MakeSphere.hxx,
BRepPrimAPI_MakeTorus.hxx
ofrecen funciones para construir los objetos. Estos objetos proporcionan un **framework** para:
 - definir la construcción de las primitivas .
 - implementar y consultar el algoritmo de construcción.
- **TNaming_Builder.hxx**: permite agregar pares de "forma antiguaaz "forma nuevaçon la evolución especificada a esta.
- **AIS_Shape.hxx**: marco para gestionar la presentación y selección de formas, contiene funciones estándar disponibles que le permiten preparar operaciones de selección en los elementos constitutivos de formas (vértices, bordes, caras, etc.) en un contexto local abierto.
- **AIS_InteractiveContext.hxx**: permite administrar el comportamiento gráfico y la selección de objetos interactivos en uno o más visores.
- **OpenGL_GraphicDriver.hxx**: define un **driver** e gráficos para *OpenGL*.
- **Graphic3d_GraphicDriver.hxx**: permite la definición de un controlador gráfico para interfaz 3d.
- **V3d_View.hxx** Los métodos de esta clase permiten editar y consultar los parámetros vinculados a la vista.
- **V3d_Viewer.hxx**: define servicios en objetos tipo visor, los métodos de esta clase permiten la edición e interrogación de los parámetros vinculados al visor (vista, luz, plano).
- **Aspect_DisplayConnection.hxx**: crea y proporciona conexión con un servidor X, levanta luna excepción si no se puede conectar este servidor.
- **Xw_Window.hxx**: define la ventana *XLib* destinada a la creación de contexto *OpenGL*.
- **TPrsStd_AISViewer** base para definir un atributo de visor interactivo., este atributo almacena un contexto interactivo en la etiqueta raíz.
- **TPrsStd_AISPresentation.hxx** atributo para asociar un objeto *AIS_InteractiveObject* a una etiqueta en un visor AIS. Este atributo funciona en colaboración con *TPrsStd_AISViewer*.

Además se utilizaron las librerías correspondiente a los atributos de OCAF, (ver apartado 1.3.1)

Para la solución fue necesario implementar varias clases:

- **TOcaf_Application**: Esta clase hereda de la librería *TDocStd_Application.hxx* para gestionar la creación de los documentos e inicializar las vistas.
- **Handle_TOcaf_Application**: puntero que hace referencia a *TOcaf_Application*.
- **TOcaf_Commands**: hace las conversiones de *TopoDS-Shape* a etiquetas.
- **TOcafFunction_BoxDriver, TOcafFunction_ConeDriver, TOcafFunction_CylinderDriver, TOcafFunction_SphereDriver, TOcafFunction_TorusDriver**: maneja los *drivers* de construcción correspondientes a las primitivas.
- **Handle_TOcafFunction_BoxDriver, Handle_TOcafFunction_ConeDriver, Handle_TOcafFunction_SphereDriver, Handle_TOcafFunction_TorusDriver**: punteros que hacen referencias a las primitivas y donde se hacen lo casteos.
- **DlgBox, DlgCone, DlgSphere, DlgTorus, DlgCylinder**: crea una ventana para los parámetros de construcción de las primitivas.
- **CommandOcc**: se declaran los eventos del *mouse*.
- **OCCViewer**: se crea el visor de OpenCASCADE y se implementan los eventos del *mouse*.

Se implementaron los métodos en la clase principal (*MainWindow*) relacionados con las funcionalidades de OCAF:

- **newDocOcafPart()**: crea documento de OCAF para el módulo *Part*.
- **createView()**: se le adjunta el visor al documento creado.
- **addWindowOcc(...)**: crea la ventana donde el visor se va a mostrar.
- **makeCone(), makeCylinder(), makeBox(), makeTorus() y makeSphere()**: crean las primitivas correspondientes.
- **SalvarOcaf()**: salva el documento de OCAF en los formatos definidos.
- **Undo**: deshace un paso en proceso de modelado.
- **Redo**: rehace un paso en proceso de modelado.
- **exportShapeOcaf()**: exporta en los formatos **STEP, STL, BREP, IGES**

Se utilizó 20 ficheros cabeceras(.h) y 14 ficheros fuentes(.cpp) con un total de 3147 líneas de código y 629 líneas en blanco.

3.5.5. Resultados de la implementación

En este apartado se muestran imágenes de capturas de pantalla con los resultados obtenidos luego de integrar la arquitectura de OCAF y haber implementado las funcionalidades.

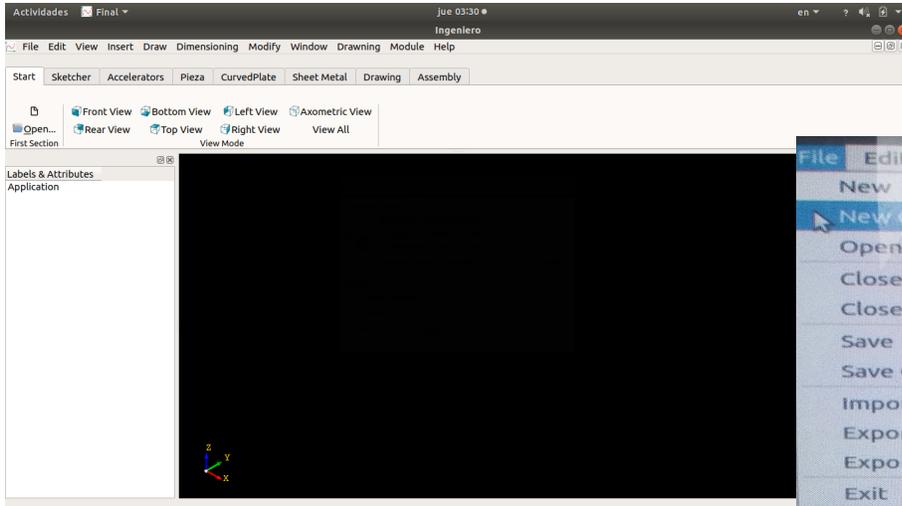


Figura 3.7. Nuevo documento OCAF

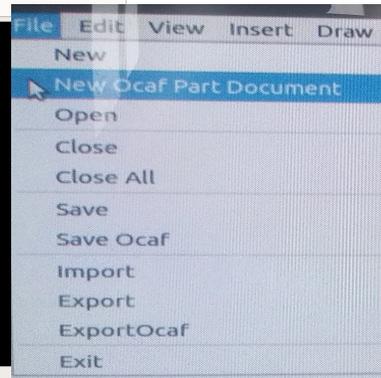


Figura 3.8. Menú con la opción crear nuevo documento

Para crear un nuevo documento OCAF se va a *file*, *New Ocaf Part Document*, este agregará el visor de OpenCASCADE.

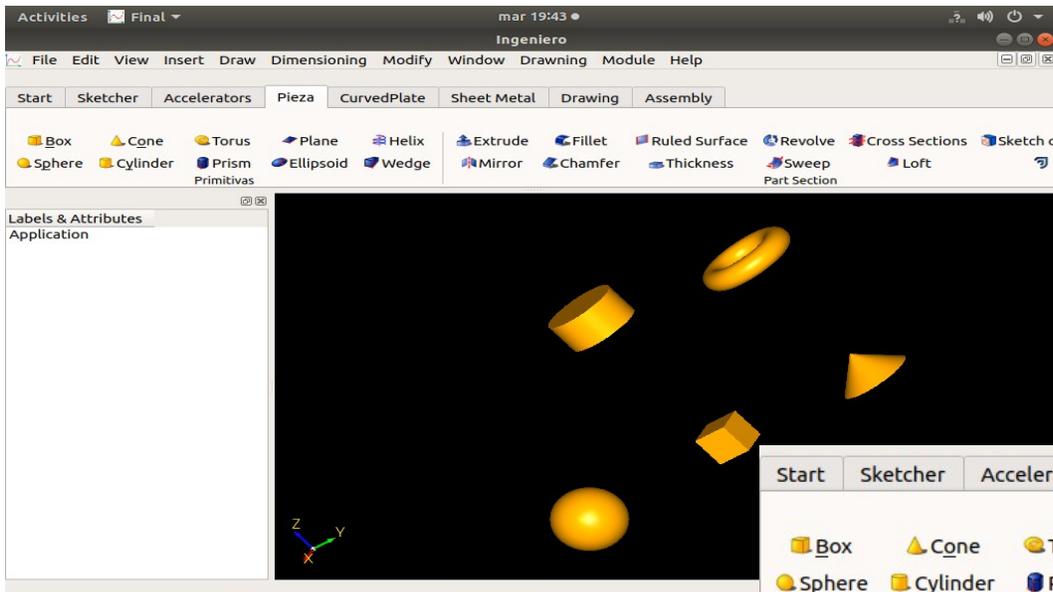


Figura 3.9. Insertar Primitivas

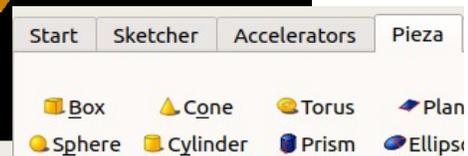


Figura 3.10. Íconos que crean las primitivas

Para crear las primitivas se va al *Piezas* y se selecciona el ícono correspondiente, a partir de este se muestra un dialogo con los parámetros de construcción.[3.11](#)

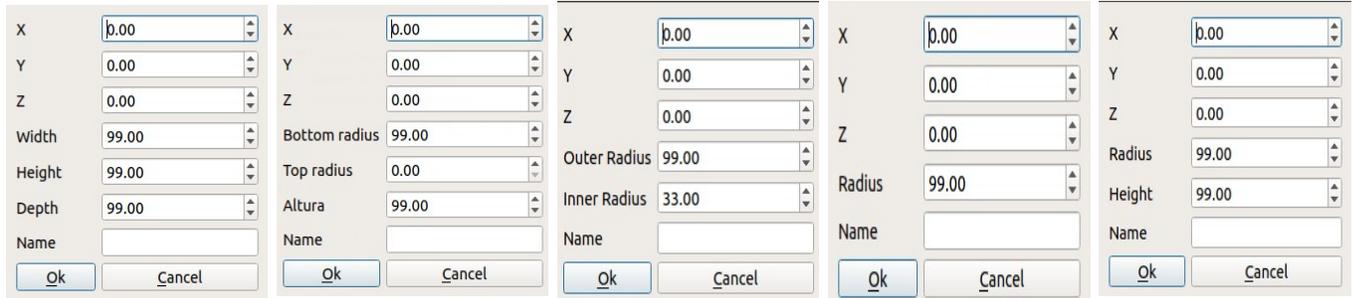


Figura 3.11. Prototipo de interfaz para la caja, el cono el torus, la esfera y el cilindro respectivamente

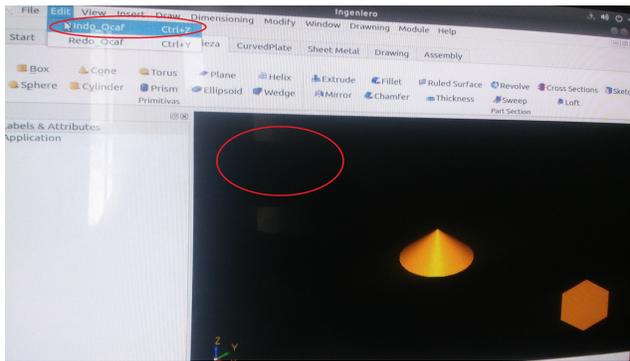


Figura 3.12. Deshacer

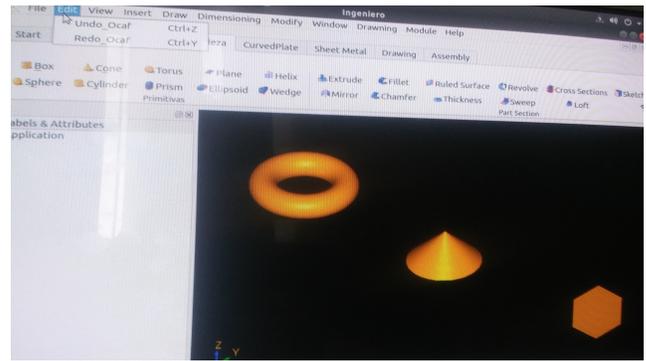


Figura 3.13. Rehacer

En la figura 3.12 se deshace un paso, en la figura 3.13 rehace un paso. Para hacer estas operaciones se va al menú *edit* en la opción *redo Ocaf* y *undo Ocaf* respectivamente.

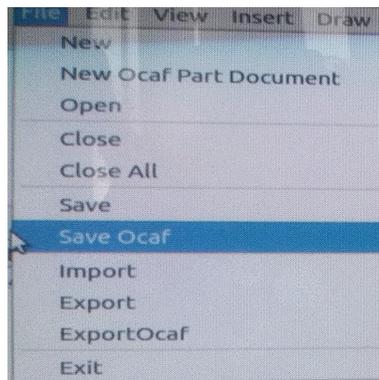


Figura 3.14. Salvar documento de OCAF

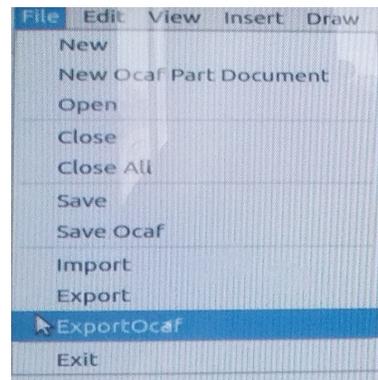


Figura 3.15. Exportar documento de OCAF

Para guardar el documento de OCAF (3.14) se va *File/Save Ocaf* y para exportarlo (3.15) en los formatos de intercambio *File/ExportOcaf*, esto abrirán el explorador donde se seleccionará la ubicación y la extensión de archivo.

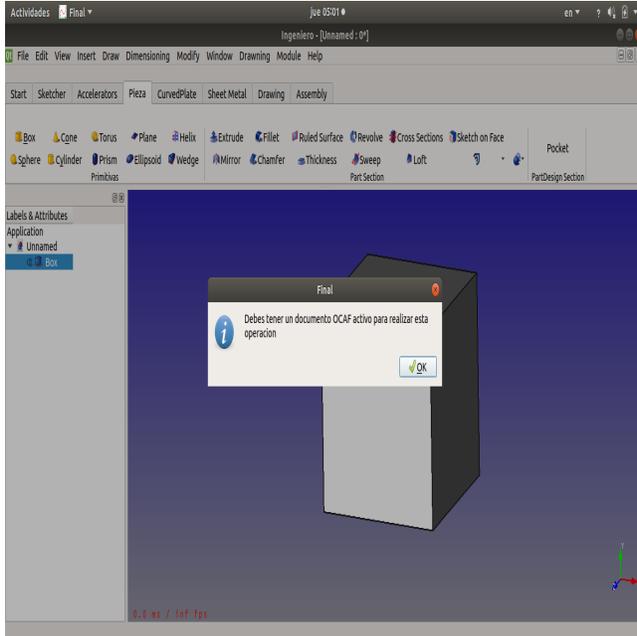


Figura 3.16. Visor de Inventor

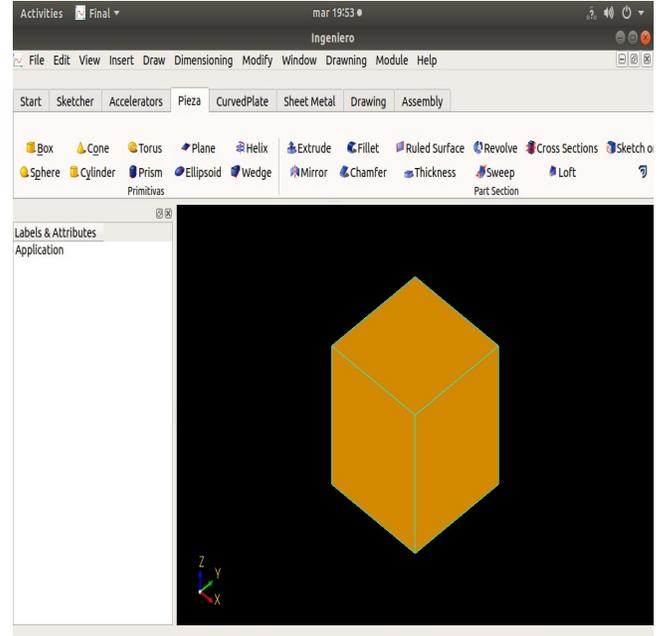


Figura 3.17. Visor de OpenCASCADE

En caso de tener un documento del sistema **Ingeniero** (figura 3.16) y trata de acceder a alguna funcionalidad de OCAF, la aplicación le mostrará al usuario una alerta avisándolo de esto.

3.6. Conclusiones del capítulo

Una vez analizados los elementos de diseño y técnicos de la propuesta de solución se arriba a las siguientes conclusiones:

- Se logró integrar la arquitectura del marco de trabajo para aplicaciones de OpenCASCADE a la aplicación **Ingeniero** y a partir de esto se implementaron funcionalidades básicas para el tratamiento de los datos.
- La propuesta de solución es una buena base para continuar con el desarrollo de esta arquitectura.

Evaluación del resultado

Las pruebas constituyen una actividad en la cual un sistema o componente es ejecutado bajo condiciones específicas, se observan o almacenan los resultados y se realiza una evaluación de algún aspecto del sistema o componente. (Sommerville, 2006)

4.1. Pruebas

El proceso de pruebas de software tiene los siguientes objetivos:

- Demostrar al desarrollador y al cliente que el software satisface sus requisitos. Esto significa que debería haber al menos una prueba para cada requerimiento o característica que se incorporará a la entrega del producto.
- Descubrir defectos en el software en el que el comportamiento de este es incorrecto, no deseable o no cumple su especificación. La prueba de defectos está relacionada con la eliminación de todos los tipos de comportamientos del sistema no deseables, tales como caídas del sistema, interacciones no permitidas con otros sistemas, cálculos incorrectos y corrupción de datos. (ibíd.)

4.1.1. Niveles de prueba

Para aplicarle pruebas a un sistema se deben tener en cuenta una serie de objetivos en diferentes escenarios y niveles de trabajo, debido a que las pruebas son agrupadas por niveles que se encuentran en distintas etapas del proceso de desarrollo. Los niveles de prueba son: (Tenorio, 2010)

- **Prueba Unitaria o de Unidad:** se centra en el proceso de verificación de la menor unidad del diseño del software: el componente de software o módulo. Se emplea para detectar errores debidos a cálculos incorrectos, comparaciones incorrectas o flujos de control inapropiados. Las pruebas del camino básico y de bucles son técnicas muy efectivas para descubrir una gran cantidad de errores en los caminos.

- **Prueba de Integración:** es una técnica sistemática para construir la estructura del programa mientras que, al mismo tiempo, se llevan a cabo pruebas para detectar errores asociados con la interacción. El objetivo es tomar los módulos probados mediante la prueba unitaria y construir una estructura de programa que esté de acuerdo con lo que dicta el diseño.
- **Pruebas de Sistema:** está constituida por una serie de pruebas diferentes cuyo propósito primordial es ejercitar profundamente el sistema basado en computadora. Aunque cada prueba tiene un propósito diferente, todas trabajan para verificar que se han integrado adecuadamente todos los elementos del sistema y que realizan las funciones apropiadas. Algunas de estas son: pruebas de recuperación, seguridad, resistencia, entre otras.
- **Pruebas de Aceptación:** es la realización de una serie de pruebas de caja negra que demuestran la conformidad con los requisitos. Un plan de prueba traza la clase de pruebas que se han de llevar a cabo, y un procedimiento de prueba define los casos de prueba específicos en un intento por descubrir errores de acuerdo con los requisitos.

4.2. Métodos de pruebas

El principal objetivo del diseño de casos de prueba es obtener un conjunto de pruebas que tengan la mayor probabilidad de descubrir los defectos del software. Para llevar a cabo este objetivo, se emplearán los dos métodos de prueba:

- **Prueba de Caja Blanca:** se centran en la estructura de control del programa. Se obtienen casos de prueba que aseguren que durante la prueba se han ejecutado, por lo menos una vez, todas las sentencias del programa y que se ejercitan todas las condiciones lógicas.
- **Prueba de Caja Negra:** son diseñadas para validar los requisitos funcionales sin fijarse en el funcionamiento interno de un programa, solo se fijan en las funciones que realiza el software. Las técnicas de prueba de caja negra se centran en el ámbito de información de un programa, de forma que se proporcione una cobertura completa de prueba.

4.2.1. Niveles de caso de prueba

Los casos de prueba incluyen todas las funciones que el programa es capaz de realizar. Deben tener en cuenta el uso de todo tipo de datos de entrada/salida, cada comportamiento esperado, todos los elementos de diseño, y cada clase de defecto. Todos los requisitos deberán ser cubiertos por los casos de prueba, de manera tal que cubra el software a fondo. Se puede decir que son la descripción de las actividades que se van a ejecutar con el fin de validar la aplicación. (Duarte, 2015)

Algunas características son:

- Se usan las mismas técnicas, pero con otro objetivo.
- No hay programas de prueba, sino sólo el código final de la aplicación.

- Se prueba el programa completo.
- Uno o varios casos de prueba por cada requisito.
- Se prueba también rendimiento, capacidad, etc. (y no sólo resultados correctos).
- Pruebas alfa (desarrolladores) y beta (usuarios).

Para aplicarle pruebas a un sistema se deben tener en cuenta una serie de objetivos en diferentes escenarios y niveles de trabajo, debido a que las pruebas son agrupadas por niveles que se encuentran en distintas etapas del proceso de desarrollo. Los niveles de prueba son:

- **Prueba Unitaria o de Unidad:** se centra en el proceso de verificación de la menor unidad del diseño del software: el componente de software o módulo. Se emplea para detectar errores debidos a cálculos incorrectos, comparaciones incorrectas o flujos de control inapropiados. Las pruebas del camino básico y de bucles son técnicas muy efectivas para descubrir una gran cantidad de errores en los caminos.
- **Prueba de Integración:** es una técnica sistemática para construir la estructura del programa mientras que, al mismo tiempo, se llevan a cabo pruebas para detectar errores asociados con la interacción. El objetivo es tomar los módulos probados mediante la prueba unitaria y construir una estructura de programa que esté de acuerdo con lo que dicta el diseño.
- **Pruebas de Sistema:** está constituida por una serie de pruebas diferentes cuyo propósito primordial es ejercitar profundamente el sistema basado en computadora. Aunque cada prueba tiene un propósito diferente, todas trabajan para verificar que se han integrado adecuadamente todos los elementos del sistema y que realizan las funciones apropiadas. Algunas de estas son: pruebas de recuperación, seguridad, resistencia, entre otras.
- **Pruebas de Aceptación:** es la realización de una serie de pruebas de caja negra que demuestran la conformidad con los requisitos. Un plan de prueba traza la clase de pruebas que se han de llevar a cabo, y un procedimiento de prueba define los casos de prueba específicos en un intento por descubrir errores de acuerdo con los requisitos. Para validar el correcto funcionamiento de la herramienta se le realizarán pruebas Unitarias, de Sistema y de Aceptación.

Los Casos de Pruebas han sido realizados sobre la base de las Historias de Usuarios y tienen como objetivo fundamental encontrar la mayor cantidad posible de deficiencias existentes en las funcionalidades implementadas. Estos se encuentran el Apéndice [B](#)

4.2.2. Pruebas unitarias

Para la realización de las pruebas unitarias se empleó Qt Test, el cual es un **framework** para pruebas de este tipo a aplicaciones y librerías. Qt Test provee todas las funcionalidades comunes de los **framework** de pruebas unitarias, así como extensiones para probar interfaces gráficas de usuario. (Test, 2019) A continuación se muestra una imagen de las pruebas unitarias realizadas durante el proceso de implementación, en la que se evidencia la aceptación de todas las verificaciones

```
***** Start testing of TestUnitTest *****
Config: Using QtTest library 5.5.1, Qt 5.5.1 (x86_64-little_endian-lp64
shared(dynamic) released build GCC 5.4.0 20190621)
PASS: TestUnitTest::initTestCase()
PASS: TestUnitTest::testmakeBox()
PASS: TestUnitTest::testmakeCone()
PASS: TestUnitTest::testmakeCylinder()
PASS: TestUnitTest::testmakeSphere()
PASS: TestUnitTest::testmakeTorus()
Totals: 5 passed, 0 failed, 0 skipped, 0 blacklisted, 71ms
***** Finished testing of TestUnitTest *****
```

Figura 4.1. Prueba unitaria realizada con Qt Test.

4.2.3. Pruebas de integración

Estas pruebas valoran si los componentes individuales trabajan en conjunto tal y como se espera de ellos. En las pruebas de integración se utilizó la estrategia Big-Bang, ya que es la única estrategia de pruebas integradas que no es incremental. La estrategia Big-Bang se integra únicamente en el momento en el que se dispone de todos los componentes. (Pressman, 2005)

4.2.4. Prueba de aceptación

Una Prueba de Aceptación tiene como propósito demostrar al cliente el cumplimiento de un requisitos del software. Estas se caracterizan por: (Letelier, 2007)

- Describir un escenario (secuencias de pasos) de ejecución o uso del sistema desde la perspectiva del cliente.
- Puede estar asociada a requisitos funcionales o no funcionales.
- Un requisito tiene una o más Pruebas de Aceptación asociadas.
- Las Pruebas de Aceptación cubren desde escenarios típicos/frecuentes hasta los más excepcionales.

Se realizó una entrevista al cliente y a los estudiantes del grupo de investigación para comprobar la aceptación de las funcionalidades, con esta entrevista se obtuvo un resultado de 87 % de satisfacción entre el personal, debido a que la forma de verificar que funciona en la mayoría de los casos.

4.2.5. Resultados de las pruebas funcionales

El resultado de las pruebas de software tiene gran importancia, porque ayuda a mejorar la calidad del mismo. Se utiliza para obtener las fallas que presenta el sistema y poder analizar las futuras. Al corregir estas fallas se disminuye el mayor número de no conformidades posible, mejorando así el grado de aceptación por parte del cliente. Se realizaron tres iteraciones mediante las pruebas de caja negra las cuales identificaron las siguientes no conformidades:

No. NC	Requisito Funcional	Descripción	Complejidad	Estado
1	RF. 1	Conflictos entre librerías de OpenCASCADE con las configuraciones del Kernel Ingeniero , causando que no el visor no se mostrara.	Alta	Resuelta
2	RF. 1	El visor no se mostraba en la ventana correspondiente y no tenía un tamaño adecuado.	Alta	Resuelta
3	RF. 1	Cuando el visor se mostraba en la ventana en tamaño completo y se cambiaba a forma de ventana el visor se dejaba de mostrar.	Media	Resuelta
4	RF. 1	Después que se creaban varios documentos si no se cerraban en el mismo orden en que fueron abiertos, la aplicación lanzada el error de <i>SegmentationFault</i> .	Alta	Resuelta.
5	RF. 1	No pueden coexistir los dos visores (Inventor y OpenCASCADE) en la misma aplicación.	Alta	No resuelta
6	RF. 2	Los íconos de las primitivas tenía dependencia con ambos documentos (el de aplicación Ingeniero con respecto al nuevo de OCAF).	Media	Resuelta.
7	RF. 2	Las primitivas no se lograban mostrar en el visor.	Media	Resuelta.
8	RF. 2	Una primera primitiva no se mostraba hasta que se insertara otra nueva.	Baja	Resuelta
9	RF. 2	Solo se mostraban el cascarón de la primitiva (wired)	Baja	Resuelta
10	RF. 5	Cuando se exportaba en el formato STL , tenía los mismos atributos que el de formato BREP	Media	Resuelta
11	RF. 5	No se exportaba en formato STL cuando se le ponía directamente la extensión de archivo	Baja	Resuelta
12	RF. 6	No se mostraba en el disco los formatos de OCAF	Alta	Resuelta

Tabla 4.1. No conformidades.

En la figura 4.2 se muestran los datos correspondientes a cada iteración de prueba por las que transitó las funcionalidades.

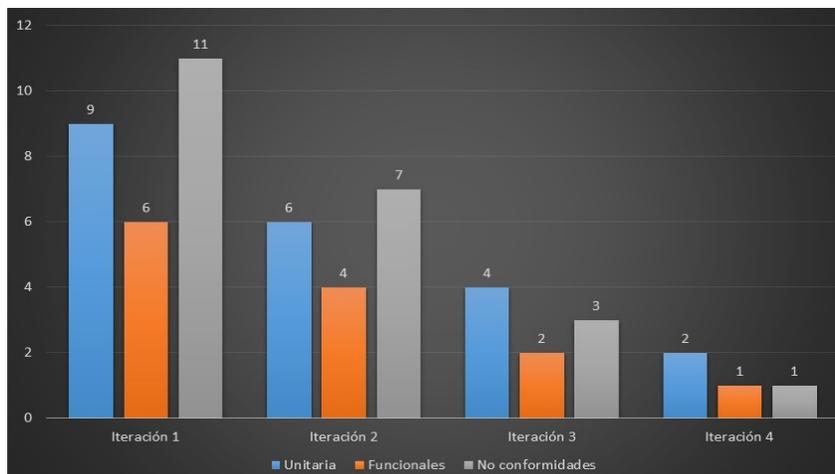


Figura 4.2. Iteraciones de las pruebas

4.3. Conclusiones del capítulo

Una vez realizado el proceso de validación de los resultados obtenidos, se comprobó que las funcionalidades implementadas para lograr la integración de OCAF, logran satisfacer en gran por ciento los requisitos propuestos, se logró resolver inconformidades importantes que arrojaron algunas de las funcionalidades. A través de las pruebas se puede constatar que el sistema **Ingeniero** puede funcionar bajo ningún riesgo.

Conclusiones

Las funcionalidades desarrolladas para el sistema **Ingeniero** permiten integrar la arquitectura Aplicación-Documento-Atributo basadas en OCAF, a partir de esto se hicieron funciones que para el manejo de los datos; siendo esto un punto inicial para el avance de esta aplicación.

Como sugerencia del presente trabajo quedan las siguientes temáticas a tener en cuenta:

- Generalizar para el resto de las piezas tridimensionales el resultado obtenido.
- Desarrollar las funcionalidades para que los dos visores puedan coexistir en la misma aplicación.
- Desarrollar el árbol de historial de rasgos de esta arquitectura, para esto recomiendo estudiar las diferencias entre OCE y OCCT, y la librería *Inspector.hxx*.
- Importar y cargar los datos de otros ficheros.
- Agregar otras formas de guardado para esta arquitectura.

Referencias bibliográficas

- ANDERSON, A.; HENDRICKSON, C y JEFFRIES, R, 2001. *Extreme Programming Installed* (vid. pág. 36).
- ASLP, 2019. *Estándares de codificación*. Url: <http://www.aspl.es/fact/files/aspl-fact/estandares-node2.html> (vid. pág. 33).
- CARDOZO., DR, 2016. *Desarrollo de Software: Requisitos, Estimaciones y Análisis* (vid. pág. 35).
- CATIA, 2019. Url: <https://www.3ds.com/products-services/catia> (vid. pág. 6).
- CLOC, 2019. *CLOC Count Lines of Code*. Url: <http://cloc.sourceforge.net/> (vid. pág. 30).
- COIN3D, 2019. Url: <https://bitbucket.org/Coin3D/>. (vid. pág. 19).
- COLLINS, Scott, 2016. A deeper look at signals and slots (vid. pág. 39).
- CREATOR, Qt, 2018. *Qt Creator* [WebSite]. Url: https://wiki.qt.io/About_Qt (vid. pág. 25).
- DOXYGEN, 2018. Url: <http://www.doxygen.nl/> (vid. pág. 29).
- DUARTE, María Elena Valle, 2015. Url: <https://prezi.com/jcygazqtotrn/casos-de-prueba-y-tipos-de-pruebas/> (vid. pág. 47).
- ECURED, 2019. Url: <https://www.ecured.cu/CATIA> (vid. pág. 6).
- ESPINOZA, Br. Mayela y CARRERA, Prof. Vãctor, 2015. Análisis comparativo de herramientas computacionales CAD basado en versión libre con programas comerciales (vid. pág. 20).
- FORUM, FreeCAD, 2009. Url: <https://forum.freecadweb.org/viewtopic.php?f=8&t=69&p=504&hilit=ocaf#p504> (vid. pág. 19).
- FREECAD, 2017a. Url: <http://freecadweb.org> (vid. pág. 16).
- FREECAD, 2017b. Module developers guide to FreeCAD source code (vid. pág. 19).
- FREECAD, 2019. *Estructura del documento*. Url: https://www.freecadweb.org/wiki/Document_structure/es (vid. pág. 18).
- GAMMA, Erich; HELM, Richard; JOHNSON, Ralph y VLISSIDES, John, 1994. *Addison-Wesley Professional Computing Series*. Design Patterns Elements of Reusable Object-Oriented Software (vid. pág. 12).
- GITLAB, 2018. *About GitLab*. Url: <https://about.gitlab.com/> (vid. pág. 28).

- INVENTOR, SEMCO Autodesk, 2019. Url: http://www.semco.com.pe%20/web/curso%20_autodesk_inventor.jsp (vid. pág. 4).
- IZARD, Anaya Eulalia, 2017. Ampliación de cálculo de elementos de máquinas (vid. pág. 7).
- KAI, Chua Chee; JACOB, Gan G. K. y MEI, Tong, 1997. Interfaces between CAD and Rapid Prototyping Systems. Part 1: A Study of Existing Interfaces. *Advanced Manufacturing Technology* (vid. págs. 23, 24).
- LABORATORIO, NACIONAL DE CALIDAD DEL SOFTWARE, 2009. Ingeniería de software: metodologías y ciclos de vida (vid. pág. 31).
- LETÉLIER, Patricio, 2007. Pruebas de Aceptación como conductor del Proceso Software. *Universidad Politécnica de Valencia* (vid. pág. 49).
- LUZZI, C y CARMINATI, F, 2017. TGeoCad: an Interface between ROOT and CAD Systems. *Journal of Physics: Conference Series 523 012017 University of Ferrara, Via Giuseppe Saragat, 1, 44020 FE, Italy CERN, 1211 Geneva 23, Switzerland* (vid. pág. 27).
- OPENCASCADE, 2019a. Url: https://www.opencascade.com/doc/occt-7.2.0/overview/html/occt_user_guides__ocaf.html (vid. págs. 8, 13).
- OPENCASCADE, 2019b. XDE. Url: https://www.opencascade.com/doc/occt-7.0.0/overview/html/occt_user_guides__xde.html (vid. pág. 27).
- OPENCASCADETECHNOLOGY, 2017. *Open CASCADE Technology: Overview*. Url: http://dev.opencascade.org/doc/overview/html/index.html#OCCT_OVW_SECTION_1 (vid. pág. 26).
- PARADIGM, Visual, 2018. *Visual Paradigm ToolCase*. Url: <https://www.visualparadigm.com> (vid. pág. 29).
- PENADÉS, Patricio Letelier, 2013. Metodologías ágiles para el desarrollo de software: Extreme Programming (vid. pág. 37).
- PRESSMAN, Roger S, 2005. *Ingeniería de Software. Un enfoque práctico. 6ta Edición* (vid. págs. 37, 49).
- QTWIKI, 2017. *About Qt - Qt Wiki*. Url: https://wiki.qt.io/About_Qt (vid. pág. 26).
- RAPPOPORT, Ari, 2003. An Architecture for Universal CAD Data Exchange. (Vid. pág. 22).
- RIEGEL; JUERGEN y MAYER, 2019. *About FreeCad - FreeCad Documentation*. Url: http://www.freecadweb.org/wiki/index.php?title=About_FreeCAD (vid. pág. 17).
- ROLDÁN, P.K.S, 2003. *Animación y Visualización de Fenómenos Naturales. Ingeniería en Sistemas Computacionales. Puebla, México: Universidad de las Américas Puebla*. Url: http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/suarez_r_pk (vid. pág. 19).
- SADAÑA, Gabriel, 2007. Introducción a Git: Un sistema de control de versiones ... bien hecho (vid. pág. 28).
- SÁNCHEZ, TAMARA RODRIGUEZ, 2015. *Metodología de desarrollo para la actividad productiva de la UCI* (vid. pág. 31).

- SIEMENS, 2019. *Siemens PLM Software*. Url: http://www.plm.automation.siemens.com/es_sa/products/solidedge/ (vid. pág. 5).
- SIMULACIÓN-SOLIDWORKS, 2017. *Simulación-SolidWorks*. Url: <http://www.solidworks.es/sw/products/simulation/packages.htm> (vid. pág. 7).
- SOLIDWORKS-CORP, 2017a. *Diseño Eléctrico-SolidWorks*. Url: <http://www.solidworks.es/sw/products/electrical-design/packages.htm> (vid. pág. 7).
- SOLIDWORKS-CORP, 2017b. *Diseño Eléctrico-SolidWorks*. Url: <http://www.solidworks.es/sw/products/electrical-design/packages.htm> (vid. pág. 8).
- SOLIDWORKS-CORP, 2017c. *Gestión de datos-SolidWorks*. Url: <http://www.solidworks.es/sw/products/product-data-management/packages.htm> (vid. pág. 8).
- SOMMERVILLE, Ian, 2006. *Ingeniería de Software. 8va Edición* (vid. págs. 37, 46).
- SPITZ, Steven y RAPPOPORT, Ari, 2004. Integrated Feature-Based and Geometric CAD Data Exchange. *ACM Symposium on Solid Modeling and Applications* (vid. pág. 20).
- STROUSTRUP, Bjarne, 2013. *C++ Programming Language*. Ed. por 978-0321563842, Addison-Wesley ISBN (vid. pág. 25).
- TECHNOLOGY, Open Cascade, 2013. *BRep Format*. Url: https://www.opencascade.com/doc/occt-6.7.0/overview/html/occt_brep_format.html (vid. pág. 22).
- TENORIO, R. Ruiz, 2010. Las Pruebas de Software y su Importancia en las Organizaciones (vid. pág. 46).
- TEST, QT, 2019. Url: <http://doc.qt.io/qt-5/qttest-index.html> (vid. pág. 48).
- TORNINCASA, Stefano y MONACO, Francesco Di, 2010. THE FUTURE AND THE EVOLUTION OF CAD. *14 th International Research/Expert Conference Trends in the Development of Machinery and Associated Technology TMT 2010, Mediterranean Cruise* (vid. pág. 20).
- TORRES, J.C, 2005. *Diseño asistido por ordenador*. Url: <http://lsi.ugr.es/~cad/teoria/Tema1/RESUMENTEMA1.PDF>. (vid. pág. 17).
- TORVALDS, L. y HAMANO, J., 2010. *Git:Fast version control system*. Url: <http://git-scm.com> (vid. pág. 28).
- TURNER, JAMES A, 1992. Conceptual Modeling Applied to Computer-aided Architectural Design (vid. pág. 21).
- VTK, 2019. Url: <http://www.vtk.org/overview> (vid. pág. 19).

Apéndices

Historias de usuarios

Tabla A.1. Historia de usuario # 1

Historia de usuario	
Número: 1	Nombre: Crear nuevo documento de tipo OCAF con un visor asociado.
Usuario: Especialista	
Prioridad en negocio: Alta	Riesgo en desarrollo: Alta
Puntos estimados: 0.99	Iteración asignada: 1
Programador responsable: Eduardo González Castillo	
Descripción: Se creará un nuevo documento de tipo OCAF con el visor de OpenCASCADE asociado, para lograr este objetivo se tuvieron precondiciones: Se deben establecer las bases de la arquitectura de OCAF, este documento debe estar independizado del documento del sistema Ingeniero.	
Observaciones: Para crear el nuevo documento se llama el método <i>newDocOcafPart()</i> a través de una señal emitida al presionar New Ocaf Part Document, el método realiza una instancia de la clase TOcafApplication y esta llama a la función <i>NewDocument()</i> , posteriormente se ejecuta el método createView() que se encarga de inicializar el visor y adjuntarlo a este nuevo documento creado.	

Tabla A.2. Historia de usuario # 2

Historia de usuario	
Número: 2	Nombre: Insertar primitivas en el nuevo visor.
Usuario: Especialista	
Prioridad en negocio: Alta	Riesgo en desarrollo: Alta
Puntos estimados: 0.76	Iteración asignada: 2
Programador responsable: Eduardo González Castillo	
Descripción: El usuario creará las primitivas seleccionando la Esfera, Cilindro, Cubo, Cono o Torus; introducirá los parámetros referente a la creación y posicionamiento en la nueva ventana y presionará aceptar. precondicion: Se debe haber creado un documento OCAF	

Continúa en la próxima página

Tabla A.2. Continuación de la página anterior

Observaciones: Los métodos *makeCone()*, *makeCylinder()*, *makeBox()*, *makeTorus()*, *makeSphere()* crean el *Dialog*, garantizan el proceso de conversión a *TDFLabel*, asignarle la función de servicio, notificar a la aplicación del proceso, asignarle el atributo de presentación y mostrar en el visor. Este proceso incluyen funciones de las clases: *TOcafFunctionBoxDriver*, *TOcafFunctionConeDriver*, *TOcafFunctionCylinderDriver*, *TOcafFunctionSphereDriver*, *TOcafFunctionTorusDriver*, *TOcafCommands*, *TOcafApplication*

Prototipo de interfaz : (apartado 3.11)

Tabla A.3. Historia de usuario # 3

Historia de usuario	
Número: 3	Nombre: Deshacer cambio.
Usuario: Especialista	
Prioridad en negocio: Alta	Riesgo en desarrollo: media
Puntos estimados: 0.59	Iteración asignada: 3
Programador responsable: Eduardo González Castillo	
Descripción: El usuario volverá a un estado anterior en el proceso de diseño presionado <i>Undo Ocaf</i> . Como pre-condición se debe hacer creado un documento de tipo OCAF y haber realizado una operación.	
Observaciones: Esto se realiza en la función <i>undo</i> ubicada en el <i>MainWindow</i> ; si no se tiene un documento OCAF activo y el usuario intenta acceder a esta funcionalidad se le mostrará un mensaje con ese aviso.	

Tabla A.4. Historia de usuario # 4

Historia de usuario	
Número: 4	Nombre: Reshacer cambio.
Usuario: Especialista	
Prioridad en negocio: Alta	Riesgo en desarrollo: media
Puntos estimados: 0.59	Iteración asignada: 4
Programador responsable: Eduardo González Castillo	
Descripción: El usuario volverá a un estado posterior en el proceso de diseño presionado <i>Redo Ocaf</i> . Como pre-condición se debe hacer creado un documento de tipo OCAF y haber realizado una operación <i>undo</i> .	
Observaciones: Esto se realiza en la función <i>redo</i> ubicada en el <i>MainWindow</i> ; si no se tiene un documento OCAF activo y el usuario intenta acceder a esta funcionalidad se le mostrará un mensaje con ese aviso.	

Tabla A.5. Historia de usuario # 5

Historia de usuario	
Número: 5	Nombre: Exportar en los formatos de intercambio IGES, STEP, STL, BREP.
Usuario: Especialista	
Prioridad en negocio: Alta	Riesgo en desarrollo: baja

Continúa en la próxima página

Tabla A.5. Continuación de la página anterior

Puntos estimados: 0.21	Iteración asignada: 5
Programador responsable: Eduardo González Castillo	
Descripción: El usuario podrá exportar lo información contenida en el documento OCAF en formatos de intercambio presionando en la funcionalidad <i>Export Ocaf</i> , una vez hecho esto se elige el tipo de formato y la ubicación en el disco; se presiona guardar. Como precondición se debe tener un documento OCAF activo.	
Observaciones: Esta función se realiza en MainWindow con su respectiva conexión. Si no se tiene un documento OCAF activo se le notificará al usuario con un mensaje. En caso que tenga un documento del sistema Ingeniero se le notificará que esa función es perteneciente a OCAF.	

Tabla A.6. Historia de usuario # 6

Historia de usuario	
Número: 6	Nombre: Salvar el documento en los formatos de OCAF.
Usuario: Especialista	
Prioridad en negocio: Alta	Riesgo en desarrollo: Media
Puntos estimados: 0.87	Iteración asignada: 6
Programador responsable: Eduardo González Castillo	
Descripción: El usuario podrá guardar lo información contenida en el documento OCAF en sus formatoa presionando en la funcionalidad <i>Save Ocaf</i> , una vez hecho esto se elige el tipo de formato y la ubicación en el disco; se presiona guardar. Como precondición se debe tener un documento OCAF activo.	
Observaciones: Esta función se realiza en MainWindow con su respectiva conexión. Si no se tiene un documento OCAF activo se le notificará al usuario con un mensaje. En caso que tenga un documento del sistema Ingeniero se le notificará que esa función es perteneciente a OCAF.	

Diseño de casos de prueba

Tabla B.1. Diseño de Casos de Prueba RF 1

Descripción general			
Crear nuevo documento de tipo OCAF con un visor asociado.			
Escenario	Descripción	Respuesta del sistema	Flujo central
EC 1.1 Opción <i>New Ocaf part document</i>	Presiona el botón <i>New Ocaf part document</i> donde debe permitir crear el documento	Se muestra el visor de OpenCASACE en el área de trabajo.	Gui::MainWindow /newDocOcafPart()

Tabla B.2. Diseño de Casos de Prueba RF 2

Descripción general			
Insertar primitivas al visor			
Escenario	Descripción	Respuesta del sistema	Flujo central
EC 1.1 Insertar Cubo.	Selecciona el ícono correspondiente al Cubo.	Crea una ventana con los parámetros asociados a la construcción, los muestra en el visor.	Gui::MainWindow /makeBox()
EC 1.2 Insertar Cono.	Selecciona el ícono correspondiente al Cono.	Crea una ventana con los parámetros asociados a la construcción, los muestra en el visor.	Gui::MainWindow /makeCone()
EC 1.3 Insertar esfera.	Selecciona el ícono correspondiente al esfera.	Crea una ventana con los parámetros asociados a la construcción, los muestra en el visor.	Gui::MainWindow /makeSphere()

EC 1.4 Insertar Torus.	Selecciona el ícono correspondiente al Torus.	Crea una ventana con los parámetros asociados a la construcción, los muestra en el visor.	Gui::MainWindow /makeTorus()
EC 1.5 Insertar Cilindro.	Selecciona el ícono correspondiente al Cilindro.	Crea una ventana con los parámetros asociados a la construcción, los muestra en el visor.	Gui::MainWindow /makeCone()

Tabla B.3. Diseño de Casos de Prueba RF 3

Descripción general			
Deshacer Cambios			
Escenario	Descripción	Respuesta del sistema	Flujo central
EC 1.1 Opción <i>Undo Ocaf</i>	Presiona el botón <i>Undo Ocaf</i> donde debe permitir deshacer cambios	Deshace un paso en el proceso de modelado	Gui::MainWindow /Undo()
EC 1.2 Documento OCAF no activo	Presiona el botón <i>Undo Ocaf</i> donde debe permitir deshacer cambios.	Muestra un mensaje notificando el error.	Gui::MainWindow /Undo()

Tabla B.4. Diseño de Casos de Prueba RF 3

Descripción general			
Rehacer Cambios			
Escenario	Descripción	Respuesta del sistema	Flujo central
EC 1.1 Opción <i>Redo Ocaf</i>	Presiona el botón <i>Redo Ocaf</i> donde debe permitir rehacer cambios	Rehace un paso en el proceso de modelado	Gui::MainWindow /Redo()
EC 1.2 Documento OCAF no activo	Presiona el botón <i>Redo Ocaf</i> donde debe permitir rehacer cambios.	Muestra un mensaje notificando el error	Gui::MainWindow /Redo()

Tabla B.5. Diseño de Casos de Prueba RF 5

Descripción general
Exportar en los formatos de intercambio IGES, STEP, STL, BREP

Escenario	Descripción	Respuesta del sistema	Flujo central
EC 1.1 Opción <i>ExportOcaf</i>	Presiona el botón <i>ExportOcaf</i> donde debe permitir guardar un fichero en el ordenador.	Exporta un modelo con las extensiones .brep, *.rle, *.igs, *.iges .stp, *.step, *.stl.	Gui::MainWindow /exportShapeOcaf()
EC 1.2 Documento OCAF no activo	Presiona el botón <i>ExportOcaf</i> donde debe permitir guardar un fichero en el ordenador.	Muestra un mensaje notificando el error	Gui::MainWindow /exportShapeOcaf()

Tabla B.6. Diseño de Casos de Prueba RF 6

Descripción general			
Salvar el documento en los formatos de OCAF (XmlOcaf, BinOcaf, MTDV-Standard)			
Escenario	Descripción	Respuesta del sistema	Flujo central
EC 1.1 Opción <i>Save Ocaf</i>	Presiona el botón <i>Save Ocaf</i> donde debe permitir guardar el documento el ordenador.	Guarda un modelo con las extensiones .std, *.cbf, *.xml.	Gui::MainWindow /salvarOcaf()
EC 1.2 Documento OCAF no activo	Presiona el botón <i>Save Ocaf</i> donde debe permitir guardar un fichero en el ordenador.	Muestra un mensaje notificando el error	Gui::MainWindow /salvarOcaf()