99.

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

INTERFAZ ENTRE UN MODELO DEL PRODUCTO Y UN MODELADOR DE SOLIDOS

T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACION

P R E S E N T A :

VEGA GONZALEZ FABIOLA SOCORRO

DIRECTOR DE TESIS: DR. VICENTE BORJA RAMIREZ

MÉXICO, D. F. 2002

AL DISTINGUIDO JURADO

PRESIDENTE:	ING. JUAN JOSÉ CARREÓN GRANADOS
VOCAL:	DR. VICENTE BORJA RAMÍREZ
SECRETARIO:	ING. LAURA SANDOVAL MONTAÑO
1ER. SUPLENTE:	DR. JESÚS MANUEL DORADOR GONZÁLEZ
2DO. SUPLENTE:	M.I. ALVARO AYALA RUIZ

CON GRAN RESPETO Y ADMIRACIÓN POR SU AYUDA PROFESIONAL

DEDICO ESTE TRABAJO A LAS PERSONAS MÁS IMPORTANTES EN MI VIDA:

MAMÁ

POR TU AMOR, TU CONFIANZA, POR TU FORTALEZA
ANTE LA VIDA Y POR DARME LO MEJOR DE TI.
GRACIAS.

PAPÁ

POR AYUDARME A DESCUBRIR QUE PUEDO
LOGRAR CUALQUIER COSA QUE DESEE.
GRACIAS.

PANCHITA

PORQUE PARA TI NO HAY IMPOSIBLES NI
MAÑANAS.
TE QUIERO MUCHO.

NORMA, ADRIANA, SUSANA Y NINA

GRACIAS A CADA UNA POR EL APOYO INCONDICIONAL Y PORQUE SÉ QUE SIEMPRE ESTAREMOS JUNTAS.

DIEGO, ANDY Y SANDY

POR SER TAN ESPECIALES Y MARAVILLOS. LOS QUIERO MUCHO.

TIN Y JORGE

POR SU GRAN AMISTAD Y CARIÑO. GRACIAS.

DANIEL

ESPECIALMENTE A TI, POR COMPARTIR TU VIDA CONMIGO, POR CADA MOMENTO QUE HEMOS PASADO JUNTOS, POR TU LOCURA, POR SER EL MEJOR.
TE ADMIRO Y TE AMO.

TABLA DE CONTENIDO

INTRODUCCIÓN

Actualmente las empresas tienen la necesidad de reducir el tiempo y el costo requeridos en el diseño y manufactura de sus productos, sin dejar a un lado la satisfacción del cliente. Para lograr lo anterior, se tiene que recurrir al uso de diferentes sistemas de cómputo que asisten a los diseñadores en la manufactura de un producto analizando, diseñando, planeando procesos, generando código para control numérico, control de calidad, etc. Dichos sistemas son llamados sistemas CAE (Ingeniería Asistida por Computadora). Algunos sistemas CAE deben consultar y almacenar información en bases de datos estructuradas de tal forma que puedan contener toda la información referente a un producto. Estas bases de datos se llaman *Modelos de Información* y a los sistemas que los utilizan se les llama sistemas *CAE basados en Modelos de Información.*

La importancia del uso de modelos de información es que permiten integrar toda la información concerniente a un producto para que pueda ser utilizada por diferentes departamentos y /o empresas y diferentes programas, sin las complicaciones de transferir datos de un programa a otro. Además el compartir la información propicia el trabajo en equipo y la consideración de aspectos de manufactura durante el diseño de productos, logrando así reducir tiempos y costos.

En el Centro de Diseño y Manufactura (CDM) de la Facultad de Ingeniería de la UNAM, se está desarrollando el proyecto "*Sistema Auxiliar para el Diseño de Ejes de Transmisión: SADET*", que consiste en diseñar, implementar y probar un sistema de cómputo que asistirá el diseño para manufactura de ejes de transmisión [PAPIIT

proyecto IN110398, CONACYT proyecto J-27775U]. *SADET* está formado por dos modelos de información y diferentes programas que hacen uso de estos modelos, por lo que ayudará a demostrar la utilidad y la aplicación potencial del uso de modelos de información para asistir el diseño para manufactura.

Como parte del proyecto *SADET*, se debe implementar una *Interfaz* que permita crear una representación en tres dimensiones de los ejes de transmisión, diseñados con el sistema, en un modelador de sólidos. Los ejes de transmisión estarán representados en una base de datos llamada Modelo del Producto (MP), que también forma parte de *SADET*. La información representada en el MP debe transferirse, a través de programación, a un modelador de sólidos, para que con base en esta información se genere el modelo en tres dimensiones.

Esta *Interfaz* es el trabajo específico reportado en esta tesis y tiene el objetivo de recuperar información del Modelo del Producto y a partir de ella generar un modelo sólido en un programa CAD[1] comercial. Cuenta con la capacidad de crear el modelo en tres dimensiones de cualquier eje con características similares a las representadas en el MP, por lo que se tendrá la base para representar cualquier pieza que se desee manufacturar (no solamente ejes de transmisión). Una vez creado el modelo 3D de un eje se tienen los beneficios siguientes:

- Se pueden utilizar estándares como STEP (Standard for the Exchange of Product Model Data [ISO, 1994]) que define un formato de datos neutral para la representación e intercambio de datos del productos, es decir, con STEP es posible consultar un modelo en distintos sistemas CAD, por ejemplo un modelo hecho en SolidWorks es posible verlo en AutoCad y viceversa.

[1] Los sistemas CAD son una herramienta importante para crear planos de un producto y modelos logrando visualizar el producto final antes de su construcción y haciendo, en muchos casos, innecesaria la construcción de prototipos o reduciendo el número de ellos. Una vez listo el modelo geométrico, permiten simular su comportamiento, su proceso de fabricación, etc., y la larga reducen el tiempo del proceso total. Estos sistemas de cómputo aportan rapidez y mayor exactitud en el diseño de un modelo.

- Es posible utilizar herramientas que realizan análisis de elemento finito (análisis de la estructura y material de una pieza sometida a carga para evitar posibles fracturas).
- Puede transferirse a sistemas de simulación y a sistemas que permitan manufacturarlo.
- Se puede obtener su plano, indicando las acotaciones y tolerancias para tener documentado dicho modelo, etc.

Esta tesis se integra con el trabajo de estudiantes de maestría y doctorado de ingeniería mecánica, los cuales desarrollan los diferentes programas que forman parte de *SADET*.

En el primer capítulo de este trabajo se explica brevemente lo que es la programación orientada a objetos (POO) y algunos conceptos relacionados como el Análisis y Diseño Orientado a Objetos (AOO Y DOO), Bases de Datos, se describe la notación conocida como UML (Unified Modeling Language) utilizada en el desarrollo de la *Interfaz*, se introduce lo que es el Diseño basado modelos de Información y se presenta con mayor detalle el proyecto SADET. En el segundo capítulo se exponen los requerimientos de la *Interfaz* y se explica cómo se lleva a cabo la representación de los ejes en el MP. En el tercer capítulo se analizan las alternativas de solución para la *Interfaz*, la evaluación y la selección de las herramientas que se utilizan. En el capítulo 4 se lleva a cabo el desarrollo del análisis y diseño para la construcción de la *Interfaz*, utilizando la metodología y la notación seleccionadas. En el capítulo 5 se habla sobre las herramientas usadas para lograr la implementación del programa, las cuales son: ObjectStore (herramienta usada para la implementación de las bases de datos), Visual C++ (Lenguaje de Programación) y SolidWorks (Modelador de sólidos). Posteriormente se describen las ventanas que conforman *la Interfaz* y la forma en que funciona. A continuación, el capítulo 6 presenta el caso de estudio. Finalmente se presentan conclusiones, comentarios y las referencias bibliográficas.

OBJETIVOS

El objetivo de esta tesis es el desarrollo, documentación y prueba de una interfaz que genere un modelo sólido, en un programa CAD comercial, de los ejes de transmisión representados en el Modelo del Producto (MP) de SADET.

La interfaz debe cumplir con los siguientes requerimientos:

- Tomar la información necesaria de un eje seleccionado del MP.
- Convertir la información a un lenguaje de programación.
- Indicar al modelador de sólidos que genere un modelo en tres dimensiones basándose en dicha información.
- Guardar el modelo sólido del eje creado.

La documentación de la interfaz debe ser elaborada con base en una metodología para desarrollo de software formal, debe comprender todas las etapas de desarrollo y la totalidad de las funciones del software.

La prueba de la interfaz consistirá en el desarrollo de un caso de estudio.

CAPÍTULO 1
ANTECEDENTES

En este capítulo se tratan temas en que se fundamenta el desarrollo del sistema *Interfaz* y tiene el objetivo de facilitar su comprensión.

1.1 Tipos de Programación

Una computadora es una herramienta-máquina que tiene la capacidad de realizar operaciones y tomar decisiones a una velocidad que supera millones de veces al ser humano. Para realizar sus tareas las computadoras interpretan un conjunto de instrucciones y datos; las instrucciones le indican qué procesos realizar, cuándo debe realizarlos y cómo mostrar los resultados obtenidos (en pantalla o en impresora) con base en los datos dados. Este conjunto de instrucciones y datos se llama 'Programa de Computadora' o 'Sistema de Computadora' y existen diferentes lenguajes de programación que permiten crearlos y, con ello, facilitar el uso de la computadora.

Los datos que se utilizan en los programas pueden ser de varios tipos, por ejemplo, existen datos que son números (enteros o reales), cadenas de palabras, tipo booleano (este tipo de dato únicamente puede tomar dos valores: Verdadero ó Falso), etc.

Existen principalmente dos tipos de programación: La programación estructurada y la programación orientada a objetos.

1.1.1 Programación Estructurada

La Programación Estructurada es una técnica para producir programas formados por una función o procedimiento principal *(main)* y una o más funciones que se ejecutan si así lo indica dicha función principal. Los programas estructurados utilizan tres estructuras lógicas de control que se combinan para que realicen cualquier tarea de procesamiento de información:

a. Secuencia: Sucesión simple de dos o más operaciones.

b. Selección: bifurcación condicional de una o más operaciones.

c. Iteración: Repetición de una operación mientras se cumple una condición.

La mayor desventaja en este tipo de programación es el difícil mantenimiento del programa, es decir, si es necesario hacerle correcciones, debe reestructurarse casi todo el código. Sin embargo, ofrece las siguientes ventajas:

1. Los programas son más fáciles de entender. Un programa estructurado puede ser leído en secuencia, de arriba hacia abajo, sin necesidad de estar saltando de un sitio a otro en la lógica, lo cual no sucede con otros estilos de programación. La estructura del programa es muy clara y es posible comprender lo que hace cada función ya que las instrucciones están más ligadas o relacionadas entre sí.

2. Reducción del esfuerzo en las pruebas. El seguimiento de las fallas ("debugging") se facilita debido a la lógica más visible, de tal forma que los errores se pueden detectar y corregir fácilmente.
3. Programas más sencillos y más rápidos.
4. Los programas quedan mejor documentados internamente.

1.1.2 Programación Orientada a Objetos

"La programación orientada a objetos es un método de implementación en el que los programas se organizan como colecciones cooperativas de objetos, cada uno de los cuales representa una instancia de alguna clase, y cuyas clases son, todas ellas, miembros de una jerarquía de clases unidas mediante relaciones de herencia." [Booch, 1996].

En este tipo de programación un objeto es la representación de algo real o abstracto. La representación de un objeto se hace a través de sus propiedades y comportamientos relevantes, dejando a un lado los detalles del mismo.

"Una clase es un grupo de objetos con propiedades (atributos) similares, comportamiento común (operaciones), relaciones comunes entre objetos, y semántica común." [Rumbaugh, 1999].

Para que un lenguaje de programación sea clasificado como orientado a objetos debe utilizar objetos, no algoritmos, como sus bloques lógicos de construcción, cada objeto debe ser una instancia de alguna clase y las clases deben estar relacionadas con otras clases por medio de relaciones de herencia. A continuación se definen algunos términos implícitos en la programación orientada a objetos.

Términos utilizados en la Programación Orientada a Objetos

Además del manejo de clases y objetos, la programación orientada a objetos cuenta con las características que a continuación se definen:

- *Herencia.* Se habla de herencia cuando una clase tiene el mismo comportamiento (atributos y métodos o funciones) que otra clase, pero adicionalmente tiene su propio comportamiento (atributos y métodos o funciones). La clase que hereda su comportamiento a otra clase es llamada clase padre o superclase y la clase heredera se llama subclase. Una subclase, aparte de los atributos y métodos propios, tiene incorporados los atributos y métodos heredados de la clase padre. Una subclase puede a su vez comportarse como una clase padre y heredar a otras clases, creando de esta manera la jerarquía de herencia. Cuando una clase hereda de más de una superclase se conoce como herencia múltiple (ver figura 1.1).

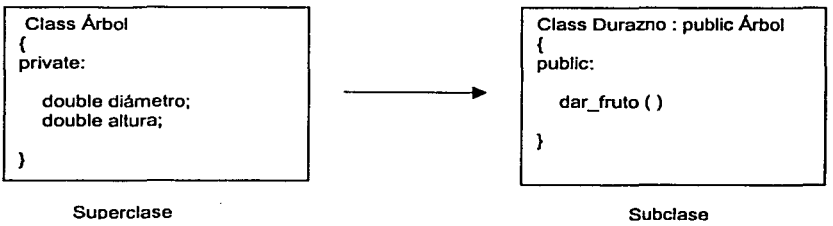

Figura 1.1 . Creación de una subclase a partir de una superclase

En la figura 1.1 se observa la notación en el lenguaje de programación orientado a objetos C++ para crear la subclase Durazno que es heredera de la clase Árbol. Debe notarse que en la subclase no se especifican los atributos diámetro y altura ya que como es heredera de la clase Árbol ya cuenta con

esos atributos, además tiene la función de dar fruto, mientras que la clase Árbol por sí misma no tiene esa función.

- *Envío de Mensajes.* La forma en que los objetos realizan sus tareas en la programación orientada a objetos es a través de mensajes. Un objeto recibe un estímulo externo que solicita un servicio, el estímulo es la invocación de una función o método de este objeto. Al ejecutarse la función, el objeto puede solicitar servicios de otros objetos, enviándoles mensajes que implican a su vez la invocación de sus funciones, y estas funciones pueden nuevamente invocar servicios de otros objetos y así sucesivamente.

- *Polimorfismo.* Como ya se mencionó, los objetos se comunican a través de mensajes, y los mensajes pueden enviarse a los objetos de una superclase y a todos los objetos de sus clases derivadas o subclases. Cuando los objetos de la subclase responden a un mensaje, cada uno lo hace de forma diferente debido a que cada objeto está definido de diferente forma, con sus propias características y funciones aunque tengan una estructura y comportamientos comunes; a esto se le llama polimorfismo. El polimorfismo permite a cada objeto subclase responder a los mensaje de una forma diferente, de acuerdo a su definición.

La programación orientada a objetos cuenta con las siguientes ventajas:

1. Reutilización no solo de software, sino de diseños enteros de aplicaciones.
2. Los sistemas orientados a objetos son frecuentemente más pequeños que sus implantaciones equivalentes no orientadas a objetos.
3. Beneficio en los costos y planificación de programas pequeños.
4. Mayor flexibilidad al cambio ya que al agregar código no se altera la estructura y funcionamiento principal del programa, haciendo un software más potente.
5. Reducción de riesgos de desarrollo.

1.2 Análisis Orientado a Objetos

Siempre que se va a desarrollar una aplicación de software se debe definir claramente cuál es el problema. Para lograr lo anterior se hace un "Análisis". Este análisis es una investigación del problema sin contemplar su posible solución.

En el Análisis Orientado a Objetos se deben identificar y describir los conceptos u objetos más relevantes dentro del dominio del problema, para que con base en éstos se determinen los requisitos que tiene que satisfacer el sistema.

1.3 Diseño Orientado a Objetos

Para completar el desarrollo de la aplicación es necesario definir una solución con base en la lógica que satisfaga los requerimientos y necesidades de los futuros usuarios del sistema. En el "Diseño" se investiga esta posible solución.

El Diseño Orientado a Objetos es un método a seguir para identificar y describir los conceptos u objetos lógicos del software a desarrollar, creando con esto un modelo del sistema a construir que, posteriormente, será llevado a un lenguaje de programación orientado a objetos. Al identificar los objetos involucrados en el problema que se quiere resolver se está realizando una descomposición de objetos, por lo que de esta descomposición se deriva el nombre de este método.

1.4 Bases de Datos

Una base de datos se puede definir como un sistema de cómputo para almacenar información relevante que está relacionada entre sí. La información que se almacena en una base de datos es la representación de cosas reales como objetos: una mesa, un coche, etc., seres vivos o conceptos abstractos: ideas. Cada cosa representada es llamada entidad, y cada entidad tiene características propias llamadas atributos: color, tamaño, edad, etc.

A continuación se describen los dos tipos de bases de datos más utilizadas:

a) Bases de Datos Relacionales
b) Bases de Datos Orientada a Objetos

a) Bases de Datos Relacionales

Para crear una base de datos relacional se debe hacer un diseño de ésta utilizando un diagrama en el cual se puede observar su estructura; en él se representan las entidades y sus relaciones. Este diagrama es llamado *modelo entidad-relación y* se llama así porque la relación es su elemento básico.

Un modelo relacional debe contener los siguientes elementos:

- *Entidad.* Es la representación general de un conjunto de cosas ó seres vivos que contiene las mismas características y que existen en la realidad. Un ejemplo concreto puede ser la entidad llamada *coche*, pero como existen miles de coches en todo el mundo, la entidad coche está formada por un conjunto de coches. A cada coche que forma parte de la entidad se le llama instancia.

- *Atributo.* Los atributos son características que pertenecen a una entidad, por ejemplo, las características de la entidad coche son las siguientes: color, número de puertas, modelo, número de placas, etc.

- *Relación.* Es una asociación entre varias entidades. La mayoría de las relaciones son binarias, sin embargo pueden existir relaciones que incluyan a más de dos entidades. Por ejemplo, entre un libro y un estudiante, que son entidades, existe la relación *comprar*: un estudiante *compra* uno o varios libros. Para identificar las relaciones generalmente se piensa en una acción o verbo.

- *Cardinalidad.* La cardinalidad se refiere a la cantidad de entidades que se deben asociar, por ejemplo:

    - UNA A UNA: Una entidad A se puede asociar únicamente con una entidad B.
    - UNA A MUCHAS: Una entidad A se puede asociar con muchas entidades B.
    - MUACHAS A UNA: Muchas entidades A se pueden asociar a una entidade B.
    - MUCHAS A MUCHAS: Muchas entidades de A se pueden asociar con muchas entidades de B.

  Un ejemplo de cardinalidad MUCHAS a UNA sería: MUCHOS coches pertenecen a UN propietario.

- *Llave primaria.* La llave primaria es uno o varios atributos de una entidad que tiene un valor único y, por lo tanto, permite identificar a la entidad y evita que la información se duplique. Por ejemplo, la entidad *coche* tiene los siguientes atributos:

Entidad: coche
Atributos: color, número de puertas, modelo, número de placas
Llave primaria: número de placas

En este ejemplo el número de placas es único para cada coche diferente y a través de ese número se puede identificar al carro del que se quiere obtener información.

- *Campo.* Un campo es la representación de cada atributo de una instancia de una entidad. Cada atributo contiene información de un tipo determinado y, de acuerdo a esa información, los campos también toman su tipo: alfabéticos, numéricos, alfanuméricos, booleanos (falso o verdadero), etc.

- *Registro.* Un registro es el conjunto de todos los campos que pertenecen a una instancia de una entidad específica.

Todos los conceptos anteriores se muestran en la figura 1.2 tomando el ejemplo anterior de la entidad coche. Las entidades o relaciones se representan en una tabla, los atributos o características se representan en columnas, un registro se representa como una fila, la cardinalidad se representa como el número de registros y la llave primaria es un identificador único para cada tabla.

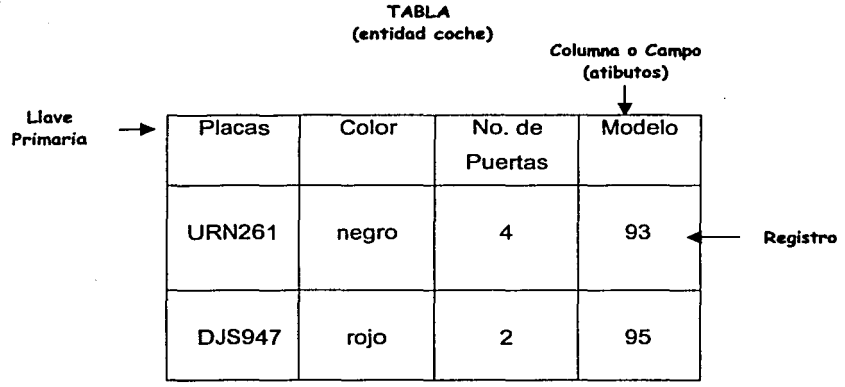

Figura 1.2. Estructura de una Base de Datos Relacional

b) Bases de Datos Orientadas a Objetos

A finales de los 80's aparecieron las primeras bases de datos orientadas a objetos (BDOO). Las BDOO almacenan información representada como objetos (almacena datos y métodos), proporcionando una estructura flexible con acceso rápido y con gran capacidad de modificación. Esta flexibilidad es una ventaja importante de las bases de datos orientadas a objetos ya que soporta el manejo de tipos de datos complejos como voz, imágenes, sonido, dibujos, planos, así como los datos alfanuméricos.

Otra ventaja importante de las BDOO, es que hace una manipulación de los datos complejos de forma rápida y ágil debido a que no se requieren búsquedas en tablas o uniones para crear relaciones. También cuenta con la ventaja de ser un sistema de almacenamiento eficaz al no perder información que se haya almacenado en él, ya que evita el acceso a los datos (objetos).

Las BDOO permiten que múltiples usuarios compartan objetos complejos y los manipulen en un ambiente seguro y estructurado pero no permiten modificarlos, el sistema debe desactivarse cuando es necesario modificar estructuras de objetos y métodos.

Existen otras características que debe cumplir un sistema de bases de datos orientado a objetos, las cuales consisten en:

- Persistencia. "La persistencia es la propiedad de un objeto por la que su existencia trasciende el tiempo y/o el espacio." [Booch Grady, 1996]. Esto se refiere a la conservación de los datos de un objeto al terminarse un proceso, con el objeto de utilizarse en otros procesos.

- Concurrencia. Permite que varios usuarios tengan acceso a la Base de Datos al mismo tiempo.

- Recuperación. Debe mantenerse el estado de la Base de Datos aún cuando no se haya podido realizar alguna operación.

- Facilidad de Consultas.

- Abstracción (Denota características esenciales). "La abstracción surge de un reconocimiento de las similitudes entre ciertos objetos, situaciones o procesos del mundo real, y la decisión de concentrarse en esas similitudes e ignorar por el momento las diferencias." [Dahl, Dijkstra y Hoare, 1972].

- Encapsulación. "El encapsulamiento es el proceso de almacenar en un mismo compartimento los elementos de una abstracción que constituyen su estructura y comportamiento; sirve para separar el interfaz contractual de una abstracción y su implantación." [Booch Grady, 1996].

- Tipos ó Clases (Mismo comportamiento). "Los tipos son la puesta en vigor de la clase de los objetos, de modo que los objetos de tipos distintos no pueden intercambiarse o, como mucho, pueden intercambiarse solo de formas muy restringidas." [Booch Grady, 1996].

- Modularidad (Abstracciones con cierta relación). Durante el desarrollo de grandes sistemas de cómputo es recomendable hacer una descomposición de éstos en módulos independientes para ayudar a manejar la complejidad.

- Jerarquía (Ordenación de abstracciones). La jerarquía es una clasificación u ordenación de abstracciones cuyo objetivo es simplificar la comprensión del problema.

También las Bases de Datos Orientadas a Objetos tienen algunas desventajas como lo es la inmadurez del mercado, es decir, la orientación a objetos en bases de datos es nueva y existen pocos productos para trabajar en la construcción de estos sistemas.

1.5 Lenguaje Unificado para la Construcción de Modelos: UML

Unified Modeling Language ó UML es un lenguaje o notación estándar (compuesto en su mayor parte de esquemas o diagramas) usado para construir modelos orientados a objetos. Con el UML se especifican, se visualizan, se construyen y se documentan los componentes de un sistema, lo que lo hace una herramienta poderosa que permite entender la problemática de un sistema y realizar un diseño óptimo de éste antes de empezar su construcción.

Esta notación es la que se utiliza en este trabajo para el desarrollo del análisis, diseño y construcción del la *Interfaz*. A continuación se listan los componentes de la notación UML[1]:

- *Actor*. Un actor es algo externo al sistema cuya función es estimularlo con el objeto de comenzar un proceso.

- *Casos de Uso*. "El caso de uso es un documento narrativo que describe la secuencia de eventos de un actor (agente externo) que utiliza un sistema para completar un proceso." [Larman Craig, 1999].

- *Diagrama de Casos de Uso*. El diagrama de Casos de Uso está formado por dos componentes: Casos de Uso y Actores.

- *Casos Expandidos de Uso*. Los casos expandidos de uso son la descripción detallada de los casos de uso anteriores para tener un conocimiento mayor de los procesos y requerimientos.

[1] Consultar el Anexo A para una descripción más detallada del UML

- *Modelo Conceptual.* El modelo conceptual es la representación en un diagrama de todos los objetos o conceptos que intervienen en la creación de un sistema.

- *Diagramas de Secuencia.* Un diagrama de secuencia describe a detalle los pasos u operaciones que el actor debe realizar en el sistema para obtener la respuesta deseada.

- *Contratos.* Un contrato es la descripción de lo que sucederá en el sistema con cada operación aplicada a éste descrita en el diagrama de secuencia.

- *Casos Reales Expandidos de Uso.* En un caso real de uso se describe la interacción de bajo nivel con la interfaz gráfica.

- *Diagramas de Colaboración.* El objetivo de los diagramas de colaboración es explicar cómo trabajan los objetos u conceptos con los mensajes para cumplir con sus tareas.

- *Diagramas de Clases.* Este tipo de diagramas muestra las clases y sus especificaciones que van a participar en la construcción del software o sistema de cómputo.

1.4 Diseño Basado en Modelos de Información

Para la construcción de los productos actuales, que manejan una complejidad y una variedad en algunos aspectos, se requiere de diversos programas de cómputo que asistan a equipos de diseño, por ejemplo los sistemas CAD (Diseño Asistido por Computadora), sistemas CAM (Manufactura Asistida por Computadora) y sistemas CAE (Ingeniería Asistida por Computadora). Cada programa de cómputo cuenta con su propio depósito de información (base de datos) y maneja la información en diferentes estructuras y lenguajes, por lo que no es posible compartir información entre dichos programas dificultando así el proceso de construcción.

Para un mejor desempeño reduciendo tiempo y costos en la realización de un producto es necesario integrar los diferentes sistemas de cómputo usados. "Esto es posible si todos los programas consultan y almacenan datos en la misma base de datos." [Prasad 1996, Fowler 1995]. La base de datos debe tener una estructura que permita guardar y utilizar toda la información que los diferentes sistemas de cómputo requieren. La estructura y contenido de las bases de datos están determinados por Modelos de Información y a los programas que interactúan con ellos se les llama aplicaciones basadas en información.

A esta nueva área de investigación, que comparte información del producto, se le ha llamado *Diseño basado en Modelos de Información.* "Los sistemas CAE son integrados por diferentes programas que interactúan con la misma base de datos, cada programa es independiente para realizar sus tareas pero todos están en un ambiente que les permite trabajar integrados, ya sea utilizando o dando información para que otro programa la utilice." [Ayala, 2001].

1.6 SADET

El desarrollo de herramientas computacionales para asistir a la manufactura de piezas ha originado diferentes líneas de investigación. Una de ellas es el modelado de información que es el fundamento para representar productos y técnicas en bases de datos. En Loughborough University, Reino Unido, se ha explorado el uso de modelos de información en el diseño de productos [Wolme A. 1995, Costa C. A. 2000, Cunningham J. J. 1995, Sorge V. 1997, Duncan J. W. 2001].

Siguiendo esta línea de investigación, en la UNAM y con la colaboración del CENIDET y dos empresas del área metalmecánica, se desarrolla un sistema CAE basado en modelos de información llamado *SADET*. Las siglas de *SADET* tienen el siguiente significado: "*Sistema Auxiliar para el Diseño de Ejes de Transmisión*" y es un proyecto que contribuirá a la demostración de la utilidad y de la potencialidad de aplicaciones que usan modelos de información (estructura y contenido de las bases de datos) con el fin de asistir el diseño para manufactura [PAPIIT proyecto IN110398, CONACyT proyecto J-27775A].

El objetivo de SADET es diseñar, implementar y probar un sistema de cómputo, para asistir el diseño por manufactura de ejes de transmisión.

La arquitectura general de SADET se muestra en la figura 1.3 y a continuación se describen las partes que lo conforman:

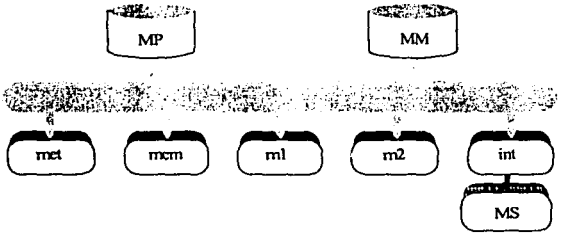

Figura 1.3. Arquitectura de SADET

- *Modelo del producto* (MP) [Mirón, 2001]. Modelo de información que especifica bases de datos que almacenarán la estructura, la geometría (cilindros, radios, estriados, engranes, etc.), dimensiones, tolerancias (dimensionales, geométricas y de posición), acabados superficiales e información de manufactura (operaciones, recursos de producción requeridos y geometría de las piezas asociadas) del eje secundario de una transmisión automotriz. El modelo estará basado en características (features).

- *Modelo de manufactura* (MM) [Ayala, 2001]. Modelo de información que especificará bases de datos que almacenarán las capacidades de manufactura de las máquinas herramienta y herramentales necesarios para producir ejes de transmisión.

- *Modelador de ejes de transmisión* (met) [Mirón, 2001]. Este programa creará bases de datos que representen ejes de acuerdo a la especificación del MP y asistirá al diseñador a ingresar datos a ellas por medio de menús, íconos y ventanas.

- *Modelador de celdas de manufactura* (mcm) [Ayala, 2001]. Este programa creará bases de datos que representen celdas de manufactura de acuerdo a la especificación del MM y asistirá al diseñador a ingresar datos a ellas por medio de menús, iconos y ventanas.

- *Módulo 1 de diseño para manufactura* (m1). Este programa tomará información del MP del sistema y revisará al eje representado considerando criterios de facilidad para manufactura. Estos contemplarán análisis de geometría, acabados superficiales y tolerancias. En caso de encontrar algún problema, el m1 notificará al diseñador y propondrá una acción correctiva.

- *Módulo 2 de diseño para manufactura* (m2). Este programa tomará información del MP del sistema y comparará los requerimientos del eje representado con las capacidades de manufactura de las celdas representadas en el MM del sistema. Los requerimientos a analizar incluirán geometría, dimensiones, acabados superficiales, tolerancias y ajustes. En caso de encontrar algún problema, el m2 notificará al diseñador y propondrá una acción correctiva.

- *Interfaz* (int). Este programa permitirá seleccionar un eje que pertenezca al MP, tomará la información correspondiente a la estructura de dicho eje y ésta será leída e interpretada por un modelador de sólidos para generar un modelo en tres dimensiones que describa físicamente al eje.

Para llevar a cabo la construcción de SADET se decidió utilizar un lenguaje de programación orientado a objetos (Visual C++) porque este tipo de programación facilita el desarrollo del software (capítulo1 sección 1.1.2).

El MM y el MP son dos sistemas de bases de datos así que, una vez que se decidió trabajar con la orientación a objetos, se propuso utilizar un sistema que

permita la creación de bases de datos orientadas a objetos, ya que, además de las ventaja ya expuestas de la orientación a objetos, las BDOO son aproximadamente dos veces más rápidas que las bases de datos relacionales (las bases de datos relacionales fueron diseñadas para manejar tipos de datos alfanuméricos y por esto difícilmente pueden manipular objetos y métodos) para almacenar y recuperar la información compleja.

Las actividades del proyecto son desempeñadas por un equipo integrado por profesores y alumnos de licenciatura, maestría y doctorado de las áreas de ingeniería mecánica y en computación. La *Interfaz (int)* de SADET es el elemento que se reporta en esta tesis.

Beneficios del proyecto

La implementación del modelo del producto será una base inicial para la industria con el objetivo de que sus áreas de diseño y manufactura compartan información. Esto se debe a que los datos del producto y manufactura que utilizan ambas áreas serán identificados y estructurados en un modelo que permitirá que se centralicen en una sola base de datos. Una contribución importante de compartir la información es el trabajo en equipo y la consideración de aspectos de manufactura, en particular capacidades y limitaciones de herramental e infraestructura, durante el diseño de productos.

Por otro lado, la creación de SADET, permitirá sintetizar los datos relevantes de un producto y de su proceso de producción, e identificar el modo en que son usados y quién los emplea. Esto promueve un mayor y mejor uso de la información de que se dispone, y ayuda a evitar duplicidad. Lo anterior origina consultas eficientes de datos produciendo importantes ahorros de tiempo y archivos reducidos.

El proyecto permitirá explorar el potencial de los sistemas basados en modelos de productos y manufactura para asistir el diseño de partes. Esto a través de un seguimiento de las técnicas y métodos usados durante la realización del proyecto y del uso de los resultados obtenidos. El proyecto también dará una visión de las tendencias actuales para el desarrollo de los futuros sistemas CAE.

CAPÍTULO 2
REQUERIMIENTOS DE LA INTERFAZ

En el presente capítulo se habla del objetivo del programa *Interfaz* y la serie de pasos que se realizaron para su construcción. Uno de esos pasos es obtener información de una base de datos llamada Modelo del Producto ó MP, considerada también como un modelo de información, (ambos términos explicados en el capítulo anterior). Posteriormente se explica la estructura del Modelo del Producto.

2.1 ¿Qué debe hacer el programa?

El objetivo de la *Interfaz* es crear el modelo en tres dimensiones de los ejes de transmisión representados en el Modelo de Producto (MP) de SADET. Como primer paso tomará información necesaria de un eje seleccionado del MP, a continuación esta información será llevada a un lenguaje de programación y, finalmente, le indicará al modelador de sólidos que genere un modelo en tres dimensiones de dicho eje. La información mínima para poder hacer esta Interfaz debe incluir geometría y dimensiones del eje.

2.2 Representación de Componentes en el Modelo del Producto

"Un *Modelo del Producto* es una representación de toda la información relevante concerniente a un producto dado durante su ciclo de vida, desde su planeación inicial hasta su venta." [Krause, F.,Kimura F.,Kjelberg, T.,1993].

Como ya se explicó, en la programación orientada a objetos se definen clases, por lo tanto para crear el Modelo del Producto (MP) se diseñó un diagrama de clases de su estructura principal utilizando UML (Unified Modeling Language) y se presenta en la figura 2.1.

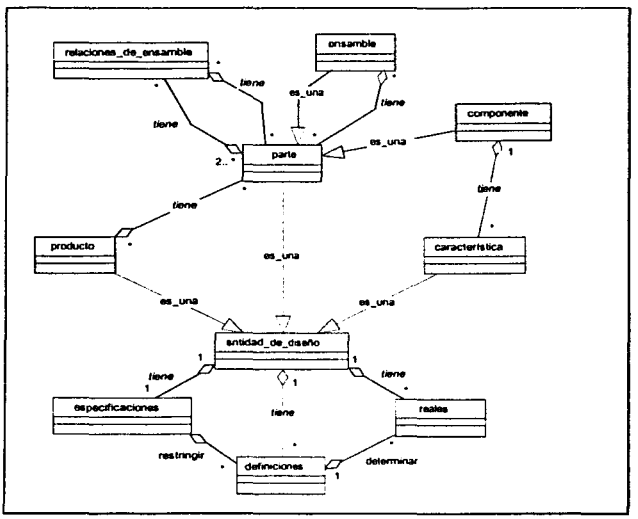

Figura 2.1. Diagrama de Clases del MP [Borja y Mirón, 1997]

La clase *entidad_de_diseño* es el núcleo de MP. Una entidad de diseño es un objeto físico o un elemento. La clase *definiciones* es usada para representar una referencia de la entidad. Un *componente* es un elemento indivisible físicamente y está formado por *características* que son formas básicas de la estructura de dicho componente. En la tabla 1 se da una breve definición de las clases del MP:

Clase	Especificación
entidad_de_diseño	Objeto físico o elemento
producto	Representa productos electro-mecánicos, eléctricos, como automóviles, grúas, etc.
parte	Componentes estándar o componentes manufacturados como motores, llantas, flechas, ejes, etc.
característica	Formas básicas puestas juntas por razones funcionales o de manufactura
componente	Elemento indivisible físicamente formado por características
ensamble	Representa un conjunto de partes
relaciones_de_ensamble	Define las relaciones entre las diferentes partes de un ensamble
especificaciones	Representa la declaración del problema, lo cual define el ámbito de la actividad de diseño
definiciones	Es usada para representar una referencia de la entidad
definición_funcional	La información contenida en esta clase representa la funcionalidad de los productos
descripción_física	Contiene información sobre atributos físicos del producto, como dimensiones, tolerancias, materiales, etc.
información_de_manufactura	Contiene información sobre la materia prima, el costo total de manufactura, el número de operaciones y el tiempo requerido para ejecutarlas
reales	Representa información de la entidad real manufacturada

Tabla 1. Clases del MP

El eje de transmisión en estudio, es considerado como un componente al que es necesario dividir en características para que su representación en el MP sea posible.

La estructura de las características se muestra a continuación en la figura 2.2:

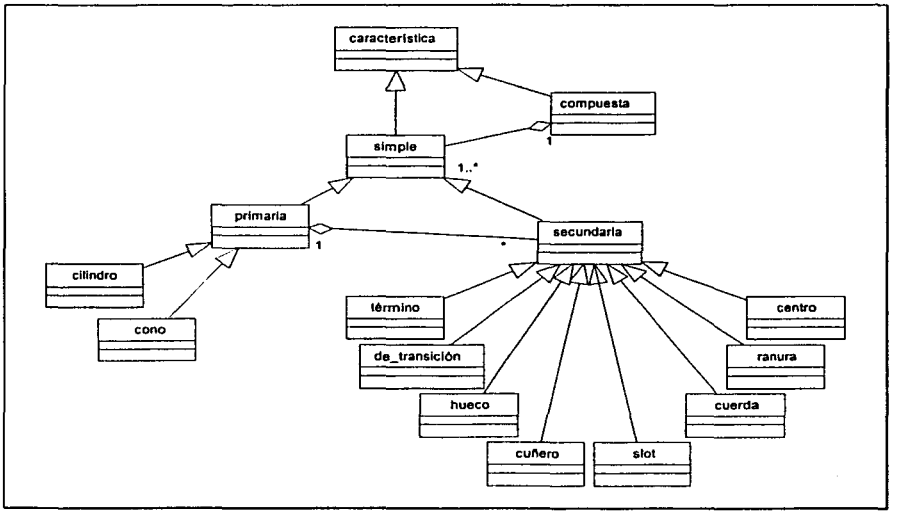

Figura 2.2. Estructura de las Características del MP [Borja y Mirón, 1997]

Las características pueden ser simples o compuestas. Las simples se subdividen en primarias (en este caso solo cilindros) y secundarias (chaflanes, estrías, etc.), las primarias forman la estructura principal del componente y las secundarias se aplican sobre las primarias para satisfacer requerimientos funcionales o de manufactura. Las características compuestas están formadas por dos o mas simples. Por lo tanto, los

componentes están formados por una serie de características primarias, asociadas una con otra, las cuales tiene características secundarias.

Para representar los atributos físicos o características de diseño de los componentes se diseñó la estructura mostrada en la figura 2.3:

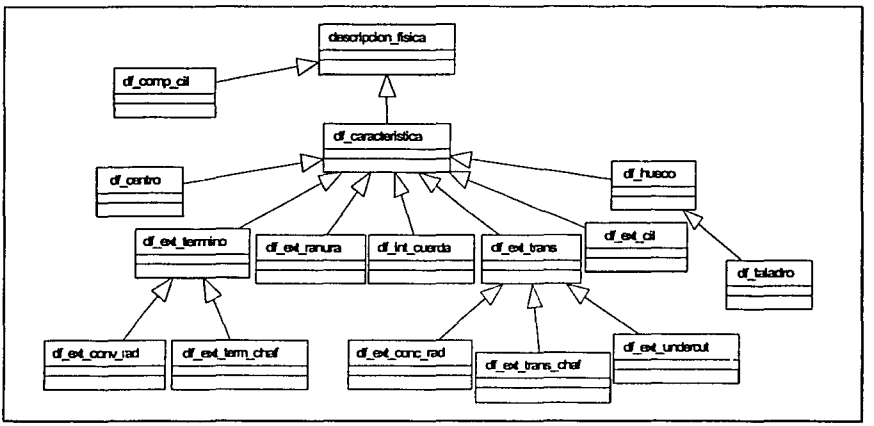

Figura 2.3. Atributos físicos de los Componentes [Borja y Mirón, 1997]

La característica *término* es asociada con el borde de una característica primaria cuando este borde no está junto a otra característica primaria. La característica de transición es asociada con el borde de una característica primaria cuando este borde está junto a una característica primaria.

En el MP las características se almacenan en un orden determinado, siendo las características primarias representadas con la letra "F" y un número, comenzando

desde cero en un orden ascendente: F0, F1, F2, etc. Las características secundarias siempre van asociadas a una característica primaria y son representadas con la letra "f" y un número comenzando desde cero en un orden ascendente: f1, f2, f3, etc. Tomado en cuenta lo anterior, la representación completa de una característica secundaria (incluyendo la característica primaria a la que está asociada) es la siguiente: F0_f1, F2_f0, F1_f5, etc. En la figura 2.4 se muestra la representación del eje de transmisión.

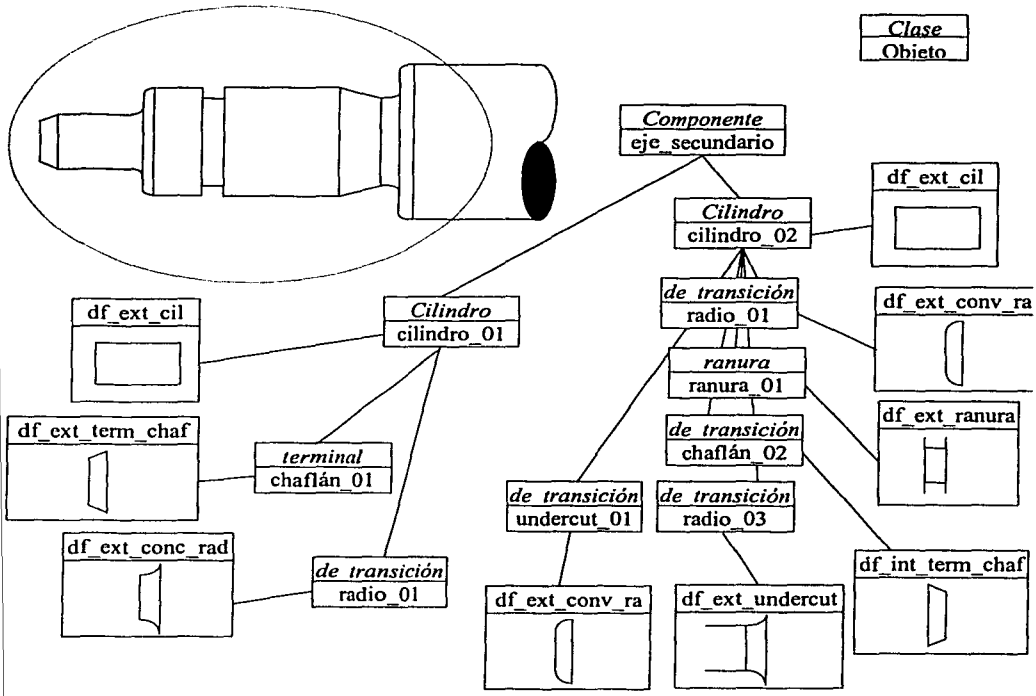

Figura 2.4. Representación del eje de transmisión secundario [Borja y Mirón, 1997]

CAPÍTULO 3
LA INTERFAZ

En este capítulo se presenta el desarrollo de la *Interfaz* utilizando los métodos definidos en el capítulo 1, tales como el análisis y el diseño orientado a objetos utilizando UML.

3.1 Modelo UML

El modelo UML que a continuación se desarrolla permite entender el problema que debe resolverse y, posteriormente, construir un sistema que ofrezca una solución óptima a dicho problema.

3.1.1 Requerimientos

El modelo de UML tiene por objeto representar la interfaz que se utilizará para crear el modelo 3D[1] de un eje de transmisión determinado. Debe recuperar información del MP (base de datos orientada a objetos) y a partir de ella generar un modelo sólido en un programa comercial.

[1] Tres Dimensiones

Funciones del Sistema

El sistema deberá:

- Ejecutar el modelador de sólidos si es necesario
- Pedir el nombre de la base de datos de la cual va a tomar la información necesaria (MP)
- Accesar a dicha base de datos (MP)
- Desplegar una lista de los ejes disponibles en la base de datos (MP)
- Permitir la selección de un eje de la base de datos (MP)
- Desplegar una lista de las versiones que existen del eje seleccionado
- Permitir la selección de una versión del eje seleccionado
- Desplegar la lista de características que componen al eje seleccionado
- Verificar que las dimensiones de las características que componen al eje estén completas
- Si falta alguna dimensión identificar cuál es y desplegar un mensaje de aviso
- Abrir una sesión del modelador de sólidos
- Dibujar modelo sólido en dicha sesión
- Indicar que ha concluido el modelo sólido
- Guardar el modelo sólido terminado

3.1.2 Actores y Casos de Uso

Un Actor es un agente externo al sistema que por lo general inicia un proceso. Un caso de uso es la descripción de un proceso que debe efectuar el sistema (Ver Anexo A.2). En este sistema a construir el Actor y los Casos de Uso son los siguientes:

Actor:	Usuario o Diseñador
Casos de Uso:	1. Obtener información
	2. Generar Modelo 3D

3.1.3 Diagrama de Casos de Uso

Como su nombre lo indica, el diagrama de Casos de Uso es la representación gráfica de los casos de uso del sistema y del actor (Ver Anexo A.3), como se muestra en la Figura 3.1:

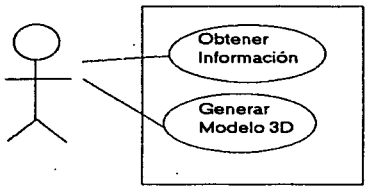

Figura 3.1 . Diagrama de Casos de Uso

3.1.4 Casos Esenciales Expandidos de Uso

Los Casos Esenciales Expandidos de Uso son una descripción detallada de los Casos de Uso (Ver Anexo A.4). Como se vio anteriormente, hay dos casos de uso: *Obtener información* y *Generar Modelo 3D,* así que también hay dos Casos Esenciales Expandidos de Uso.

Caso Expandido de Uso Obtener Información:

Actores:	Usuario
Propósito:	Tener los elementos necesarios para que se pueda generar el modelo
Resumen:	El usuario accesa al MP, selecciona un eje, selecciona una versión del eje, observa sus características y verifica que no falte alguna dimensión que impida la construcción del modelo

En la tabla 1 se muestran, en orden de ejecución, los pasos que sigue un actor y la respuesta del sistema dentro del *Caso de Uso Obtener Información*; estos pasos son llamados *Curso Normal de los Eventos y, junto con la información anterior forma el Caso Esencial Expandido de Uso Obtener Información* :

Acción de los Actores	Respuesta del Sistema
1. Comienza cuando el usuario desea obtener el modelo 3D de un eje representado una Base de Datos (MP).	
2. El usuario indica al sistema que desea abrir una Base de Datos específica.	
	3. El sistema abre la Base de Datos (MP).
4. El usuario indica al sistema que desea seleccionar un eje.	
	5. El sistema abre una ventana que contiene una lista de los ejes en el MP.
6. El usuario selecciona un eje.	
7. El usuario indica al sistema que desea seleccionar una versión del eje.	
	8. El sistema abre una ventana que contiene una lista de las versiones del eje seleccionado.
9. El usuario selecciona una versión.	
10. El usuario indica al sistema que desea consultar las características del eje.	
	11. El sistema abre una ventana que contiene todas las características del eje.
12. El usuario indica al sistema que verifique las dimensiones de las características que componen al eje seleccionado.	
	13. El sistema hace la verificación y avisa si las dimensiones están completas ó, en caso de faltar alguna, indica cuál es.

Tabla 1. Curso Normal de los Eventos para el Caso de uso *Obtener Información*

Caso de Uso Generar Modelo 3D

Actores:	Usuario
Propósito:	Crear el modelo 3D en el modelador de sólidos
Resumen:	El usuario indica al sistema que comience a dibujar, el sistema dibuja las características de acuerdo al orden en que se encuentran definidas.

En la tabla 2 se muestra el *Curso Normal de los Eventos* para el Caso de Uso Generar Modelo 3D, las características primarias se abrevian con las letras CP y las secundarias con las letras CS :

Acción de los Actores	Respuesta del Sistema
1. El usuario indica al sistema que comience a dibujar.	
	2. El sistema lee la 1ra. CP, obtiene sus parámetros.
	3. El sistema lee la 1ra CS izquierda que corresponde a la 1ra CP, obtiene sus parámetros y la dibuja.
	⋮
	4. El sistema lee la última CS izquierda que corresponde a la 1ra. CP, obtiene sus parámetros y la dibuja.
	5. El sistema lee la 1ra CS derecha que corresponde a la 1ra CP, obtiene sus parámetros y la dibuja.
	⋮
	6. El sistema lee la última CS derecha que corresponde a la 1ra CP, obtiene sus parámetros y la dibuja.
	⋮
	7. El sistema lee la última CP, obtiene sus parámetros.
	8. El sistema lee la 1ra. CS izquierda que corresponde a la última CP, obtiene sus parámetros y la dibuja.
	⋮
	9. El sistema lee la última CS derecha que corresponde a la última CP, obtiene sus parámetros y la dibuja.
	10.El sistema despliega un mensaje para indicar al usuario que el modelo está completo y guarda el modelo en un archivo en caso de que éste lo confirme.

Tabla 2. Curso Normal de los Eventos para el Caso de Uso *Generar Modelo 3D*

En la Tabla 2 se describe cómo se va dibujando el eje a través de sus características primarias y secundarias, la figura 4.2 muestra gráficamente lo cómo se van dibujando dichas características:

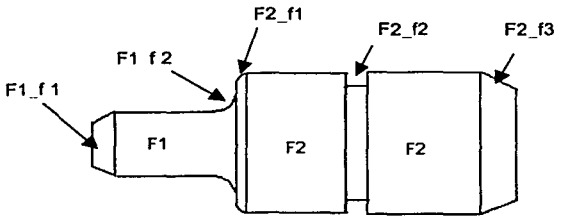

Figura 3.2 . Creación las características primarias y secundarias del eje

El sistema primero dibuja la primera característica primaria, que se representa con una letra "F" y el número "1" (consultar capítulo 2.2), después busca la primera característica secundaria representada con la letra "f" y el número 1, y continúa así hasta dibujar la última característica secundaria de la primera característica primaria, después continúa con la segunda característica primaria y así sucesivamente. Por lo tanto, el orden en que se han dibujado las características de la figura 3.2 (siempre de derecha a izquierda) es el siguiente:

1. F1
2. F1_f1
3. F1_f2
4. F2
5. F2_f1
6. F2_f2
7. F2_f3

3.1.5 Modelo Conceptual

En el modelo conceptual se representan todos los objetos o conceptos que intervienen en la creación de un sistema (Ver Anexo A.5). Como punto de partida para crear el *Modelo Conceptual* se crea una lista de conceptos pensando en objetos físicos, descripciones de cosas, lugares, transacciones, papel de las personas, contenedores de otras cosas, cosas dentro de un contenedor, otro sistema de cómputo o externo al sistema, eventos, etc., que formen parte del dominio del problema en cuestión. En este problema a resolver, los conceptos sugeridos son los que se listan a continuación:

Lista de Conceptos:

Estación de Diseño
Eje o Componente
Descripción
Depto. de Diseño
Modelo 3D
Diseñador
MP
Modelador de Sólidos (SW)
Entidad de Diseño
Definición
Característica
Simple
Primaria
Secundaria
DescripciónFísica
Cilindro
Selección
SelecciónEje
SelecciónVersión
Consulta
ConsultaCaracterística
Cilindro
Ranura
Centro
Hueco

En la figura 3.3 se presenta el Modelo Conceptual con los conceptos sugeridos.

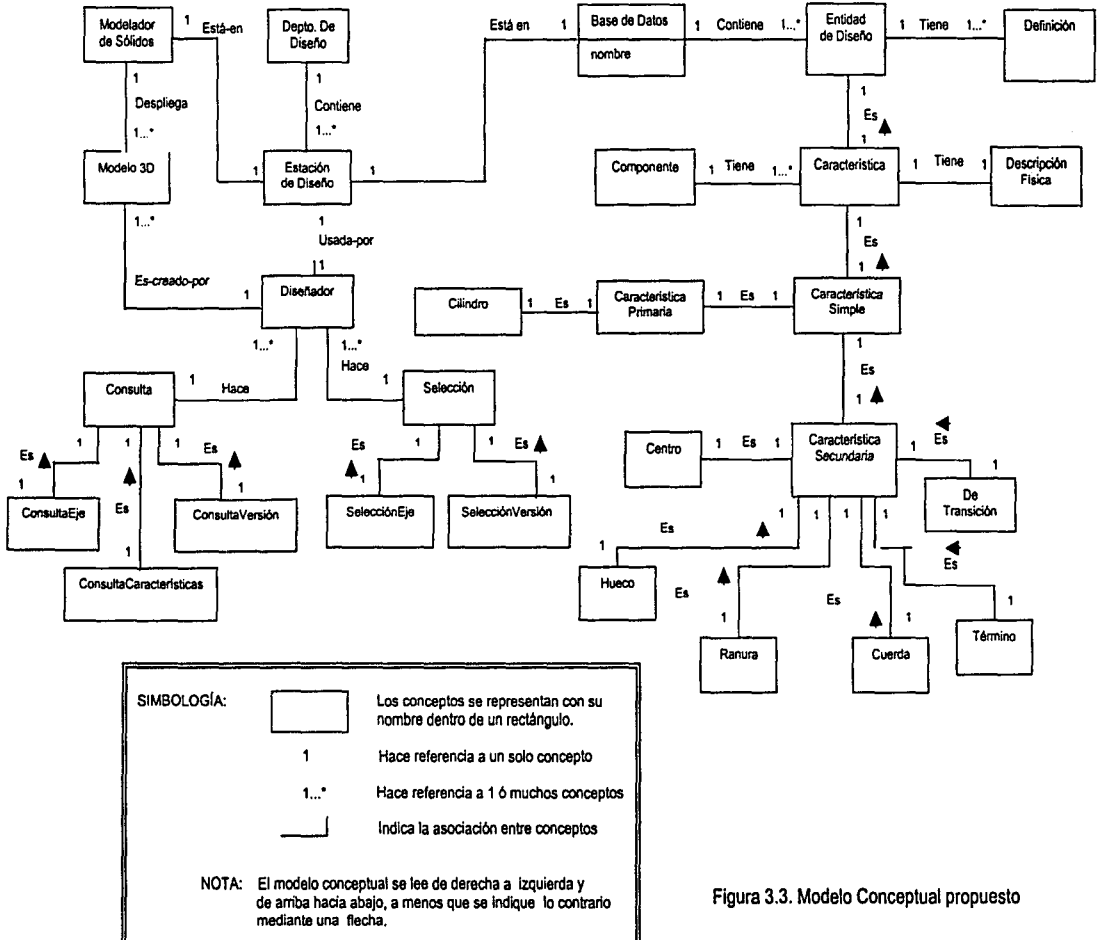

Figura 3.3. Modelo Conceptual propuesto

El Modelo Conceptual de la figura 3.3 se lee de derecha a izquierda y de arriba hacia abajo, siempre y cuando no se indique lo contrario a través de una flecha que indique el sentido. Por ejemplo, la figura 3.4 se lee así:

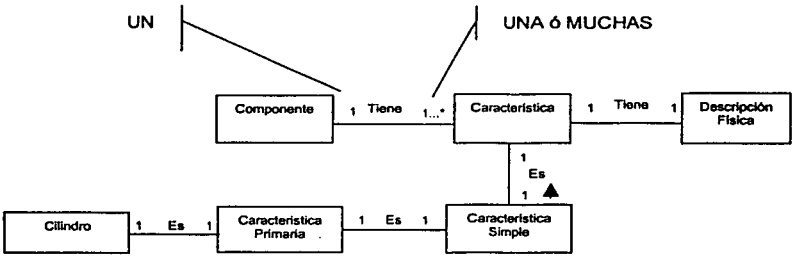

Figura 3.4 Modelo parcial del Modelo Conceptual propuesto

- UN Componente tiene UNA ó MUCHAS Características (de derecha a izquierda)
- UNA Característica tiene UNA Descripción Física (de derecha a izquierda)
- UNA Característica Simple es UNA Característica (de abajo hacia arriba, pues así lo indica una flecha)
- UNA Característica Primaria es UNA Característica Simple (de derecha a izquierda)
- UN Cilindro es UNA Característica Primaria (de derecha a izquierda)

3.1.6 Diagramas de Secuencia

Este tipo de diagrama trata de mostrar gráficamente las operaciones que efectúa sobre el sistema un actor externo, que en este caso es el diseñador o usuario, y la respuesta que el sistema da a cada una de esas operaciones (Ver Anexo A.6).

A continuación en la figura 3.5 se muestra el diagrama de secuencia que corresponde al caso de uso *Obtener Información*

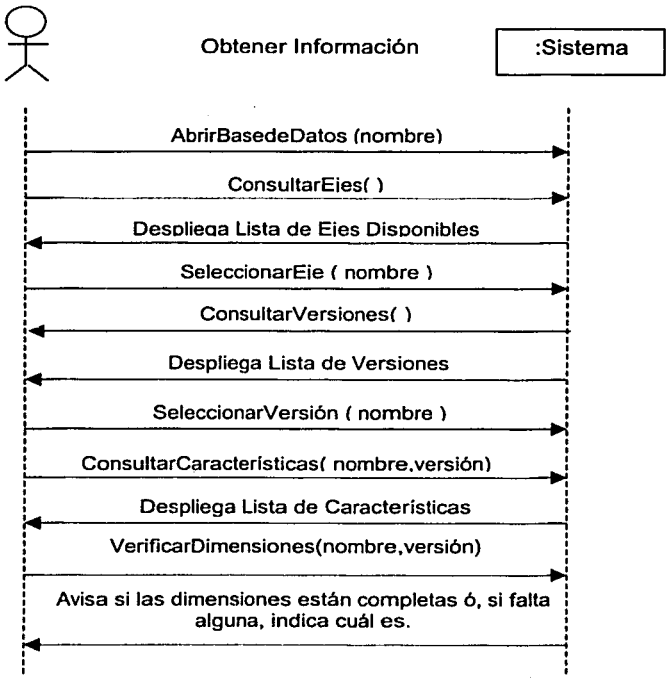

Figura 3.5. Diagrama de Secuencia para el Caso de Uso Obtener Información

El diagrama se interpreta de la forma siguiente, de acuerdo al sentido de las flechas y de la parte superior a la parte inferior:

- El usuario indica al sistema que abra una Base de Datos.
- El usuario indica al sistema que desea consultar los ejes que hay en la Base de Datos.
- El sistema responde al usuario abriendo una ventana que contiene una lista de los ejes existentes.
- El usuario selecciona uno de los ejes.
- El usuario indica al sistema que desea consultar las versiones existentes del eje ya seleccionado.
- El sistema responde al usuario abriendo una ventana que contiene una lista de las versiones de los ejes existentes.
- El usuario selecciona una de las versiones de los ejes.
- El usuario pide al sistema que muestre las características que forman al eje seleccionado.
- El sistema abre una ventana que contiene las características solicitadas.
- El usuario pide al sistema que verifique si las dimensiones de las características están completas.
- El sistema despliega un mensaje donde indica al usuario si las dimensiones están completas, si falta alguna indica cuál es.

De forma similar se lee el diagrama de la figura 3.6, en donde Característica Primaria se denota como CP y Característica Secundaria como CS :

Diagrama de secuencia para el caso de uso *Generar Modelo 3D:*

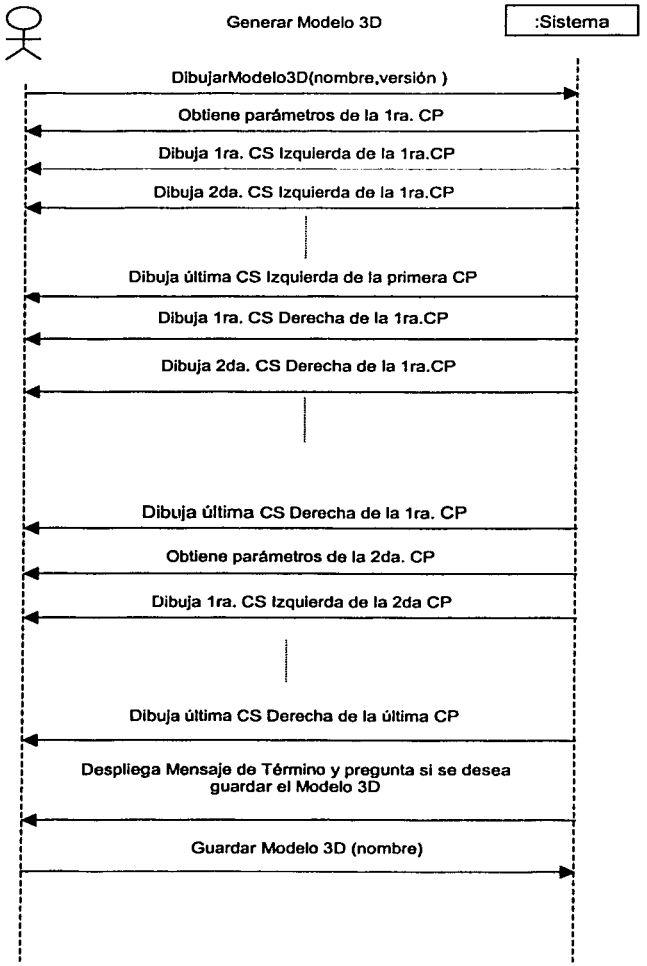

Figura 3.6. Diagrama de Secuencia para el Caso de Uso Generar Modelo 3D

Contratos

Un contrato es un documento que describe lo que sucederá en el sistema y, para cada operación que efectúa dicho sistema, se elabora un contrato (Ver Anexo A.7). El caso de uso *Obtener Información* está formado por siete operaciones, de acuerdo al diagrama de secuencia de la figura 3.5, así que habrá siete contratos en este caso de uso.

Contratos para el Caso de Uso Obtener Información:

- Contrato para la operación: *AbrirBasedeDatos*

Responsabilidades:	Abrir una ventana de navegación para poder seleccionar una BD y, posteriormente, abrirla
Poscondiciones:	Se ha establecido BasedeDatos.estaabierta a verdadero
Precondiciones:	Debe existir un archivo que corresponda a una BD que contenga ejes de transmisión
Excepciones:	Los archivos de la BD cuya extensión sea diferente de *.db no podrán ser utilizados

- Contrato para la operación: *ConsultarEjes*

Responsabilidades:	Abrir una ventana que muestre una lista con los nombres de los ejes almacenados en la BD
Poscondiciones:	Se genera una consulta y está abierta una ventana con la lista de ejes existentes en la Base de Datos.
Precondiciones:	Debe estar abierta una BD
Excepciones:	Si la BD se encuentra vacía, no habrá datos que mostrar

- Contrato para la operación: *SeleccionarEje*

  Responsabilidades: Establecer como eje activo a un eje que pertenezca a la BD abierta
  Poscondiciones: Se ha establecido Eje.estaActivo a verdadero
  Precondiciones: Debe estar indicado el nombre del eje que se desee activar

- Contrato para la operación: *ConsultarVersiones*

  Responsabilidades: Abrir una ventana que muestre una lista de las versiones correspondientes al eje activo
  Poscondiciones: Se genera una consulta y se muestra una lista de las versiones que corresponden al eje activo
  Precondiciones: Debe existir un eje activo

- Contrato para la operación: *SeleccionarVersión*

  Responsabilidades: Establecer una versión activa del eje activo
  Poscondiciones: Se ha establecido Versión.estaActiva a verdadero
  Precondiciones: Debe estar indicado el nombre de la versión que se va a activar

- Contrato para la operación*: ConsultarCaracterísticas*

Responsabilidades: Mostrar una lista de todas las características que pertenezcan al eje y versión activos

Poscondiciones: Se crea una consulta y se muestra una lista con las características del eje activo

Precondiciones: Debe existir un eje activo y una versión activa

Excepciones: Si aún no se han creado las características para el eje seleccionado, no habrá datos que mostrar

- Contrato para la operación*: VerificarDimensiones*

Responsabilidades: Indicar al usuario si las dimensiones de las características que componen al eje y versión activos están completas ó, en caso de que falte alguna, indicar cuál es

Poscondiciones: Se emite un mensaje que informa al usuario si las dimensiones de las características que componen al eje y versión activos están completas o, en caso de que falte alguna, indicar cuál es

Precondiciones: Debe existir un eje activo y una versión activa

Excepciones: Si aún no se han definido las características del eje Seleccionado no podrá efectuarse la verificación

El caso de uso Generar Modelo 3D está formado por las siguientes dos operaciones, como se muestra en la figura 4.6:

Contratos para el Caso de Uso *Generar Modelo 3D:*

- Contrato para la operación*: DibujarModelo3D*

  Responsabilidades: Dibujar las características del eje activo hasta completar el Modelo 3D, desplegar un mensaje de término y preguntar si se desea guardar el Modelo 3D

  Poscondiciones: Se creó un modelo 3D. Se muestra mensaje preguntando si se desea guardar el modelo 3D

  Precondiciones: La información del Eje debe estar completa y ser consistente

- *Operación: GuardarModelo3D*

  Responsabilidades: Guardar el Modelo 3D en un archivo

  Poscondiciones: Existe una instancia de la clase Eje

  Precondiciones: Debe existir un Modelo 3D terminado

3.1.8 Casos Reales Expandidos de Uso

En un caso real de uso se describe la interacción de bajo nivel con la interfaz gráfica (Ver Anexo A.8). Primero se definir lo siguiente:

Caso de Uso: Obtener Información
Actores: Usuario
Propósito: Tener los elementos necesarios para que se pueda generar el Modelo 3D
Resumen: El usuario accesa al MP, selecciona un eje, selecciona una versión del eje, observa sus características y verifica que no falte alguna dimensión que impida la construcción del modelo 3D

En las figuras 3.7, 3.8, 3.9 y 3.10 se muestran las ventanas que forman parte de la *Interfaz* y que es necesario observar para continuar con este análisis:

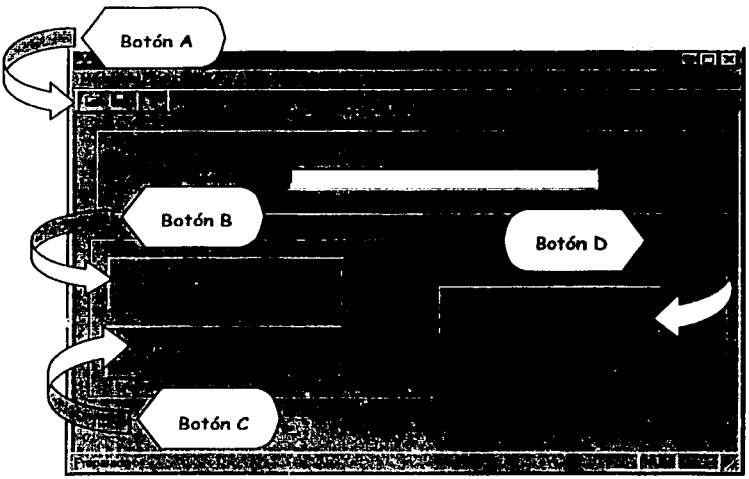

Figura 3.7. Ventana A1 de la Interfaz de Usuario

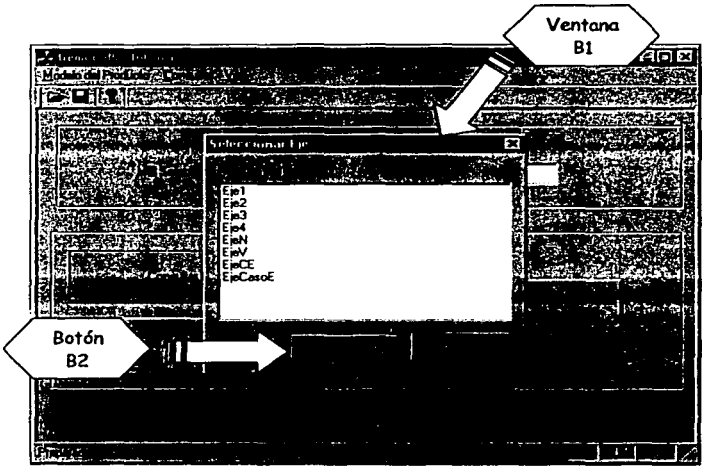

Figura 3.8. Ventana B1 de la Interfaz de Usuario

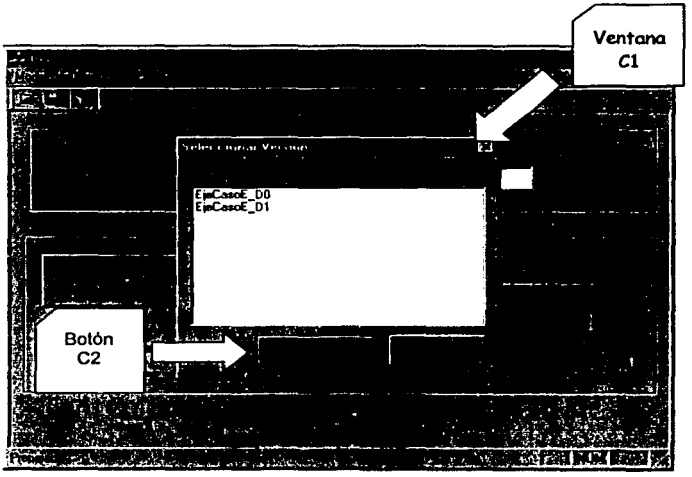

Figura 3.9. Ventana C1 de la Interfaz de Usuario

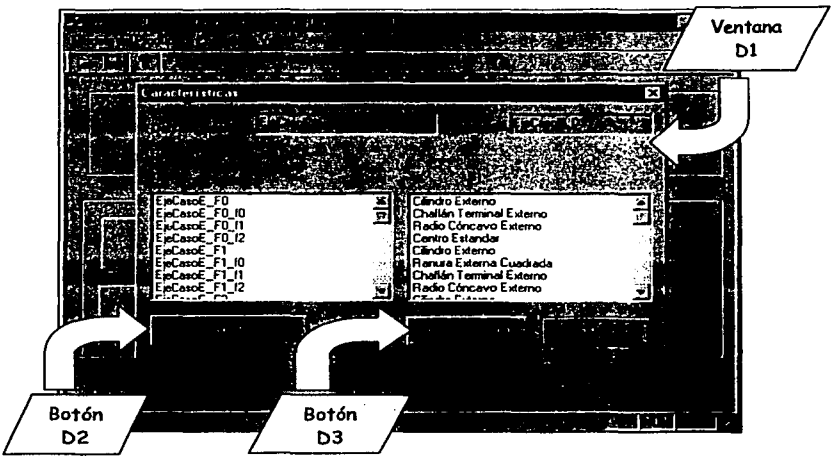

Figura 3.10. Ventana D1 de la Interfaz de Usuario

Con base en las figuras anteriores, a continuación se presenta el Curso Normal de los Eventos:

Acción de los Actores	Respuesta del Sistema
1. Comienza cuando el usuario desea obtener el modelo 3D de un eje almacenado en el MP.	
2. El usuario hace clic en el botón A (figura 3.6) para abrir una BD (MP).	
	3. El sistema abre la BD (MP).
4. El usuario hace clic en el botón B (figura 3.6) para seleccionar un eje.	
	5. El sistema abre la ventana B1 (figura 3.7) que contiene una lista de los ejes que existen en la BD (MP).
6. El usuario selecciona un eje y hace clic en el botón B2 (figura 3.7).	
	7.El sistema cierra la ventana B1 (figura 3.7)
8. El usuario hace clic en el botón C (figura 3.6) para seleccionar una versión del eje.	
	9. El sistema abre la ventana C1 (figura 3.8) que contiene una lista de las versiones del eje seleccionado.
10. El usuario selecciona una versión y hace clic en el botón C2 (figura 3.8).	
	11. El sistema cierra la ventana C1 (figura 3.8).
12. El usuario hace clic en el botón D (figura 3.6) para consultar las características del eje.	
	13. El sistema abre la ventana D1 (figura 3.9) que contiene todas las características del eje.
14. El usuario hace clic en el botón D2 (figura 3.9) para que el sistema verifique las dimensiones de las características.	
	15. El sistema verifica que las dimensiones estén completas ó indica si falta alguna.

Tabla 4. Curso Normal de los Eventos en el Caso Real Expandido de Uso *Obtener Información*

Caso de Uso:	Generar Modelo 3D
Actores:	Usuario
Propósito:	Crear el modelo 3D en el modelador de sólidos
Resumen:	El usuario indica al sistema que comience a dibujar, el sistema dibuja las características de acuerdo al orden en que se encuentran definidas.

A continuación en la Tabla 5 se muestra el Curso Normal de los Eventos para el Caso Real Expandido de Uso Generar Modelo 3D:

Acción de los Actores	Respuesta del Sistema
1. El usuario hace clic en el botón D3 (figura 3.9) para que el sistema comience a dibujar.	
	2. El sistema lee la 1ra. CP, obtiene sus parámetros.
	3. El sistema lee la 1ra CS izquierda que corresponde a la 1ra CP, obtiene sus parámetros y la dibuja.
	⋮
	4. El sistema lee la última CS izquierda que corresponde a la 1ra. CP, obtiene sus parámetros y la dibuja.
	5. El sistema lee la 1ra CS derecha que corresponde a la 1ra CP, obtiene sus parámetros y la dibuja.
	⋮
	6. El sistema lee la última CS derecha que corresponde a la 1ra CP, obtiene sus parámetros y la dibuja.
	⋮
	7. El sistema lee la última CP, obtiene sus parámetros.
	8. El sistema lee la 1ra. CS izquierda que corresponde a la última CP, obtiene sus parámetros y la dibuja.
	⋮
	9. El sistema lee la última CS derecha que corresponde a la última CP, obtiene sus parámetros y la dibuja.
	10.El sistema despliega un mensaje para indicar al usuario que el modelo está completo y guarda el modelo en un archivo en caso de que éste lo confirme.

Tabla 5. Curso Normal de los Eventos en el Caso Real Expandido de Uso Generar Modelo 3D

3.1.9 Diagramas de Colaboración

Los diagramas de colaboración explican cómo trabajan los objetos u conceptos con los mensajes para cumplir con sus tareas (Ver Anexo A.9).

Diagrama de Colaboración para la operación AbrirBasedeDatos

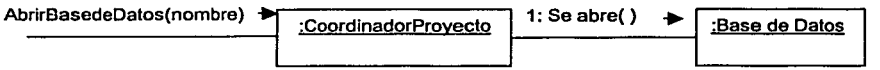

Figura 3.11. Diagrama de colaboración *AbrirBasedeDatos*

El diagrama de la Figura 3.11 se interpreta de la forma siguiente:

1. El mensaje AbrirBasedeDatos es enviado a una instancia de CoordinadorProyecto. La instancia responde y manda el mensaje a la instancia Base de Datos.

Diagrama de Colaboración para la operación ConsultarEjes

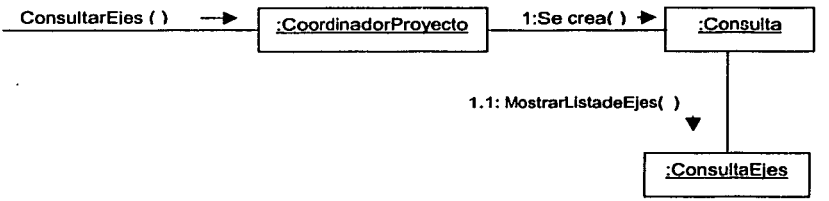

Figura 3.12. Diagrama de colaboración *ConsultarComponentes*

El diagrama de la Figura 3.12 se interpreta de la forma siguiente:

1. El mensaje ConsultarEjes se envía a una instancia de CoordinadorProyecto. La instancia corresponde al mensaje y crea una Consulta.
2. El objeto Consulta envía el mensaje MostrarListadeEjes a la instancia ConsultaEje. La instancia corresponde al mensaje mostrando la lista de los ejes existentes en la base de datos (MP).

Diagrama de Colaboración para la operación SelecionarEje

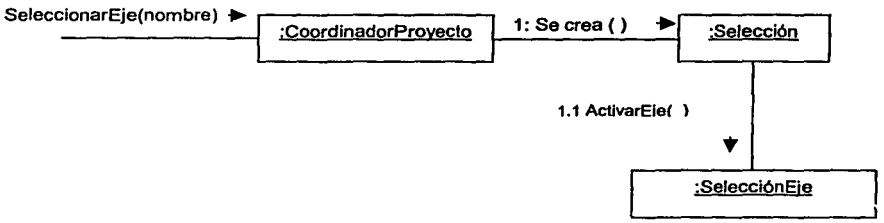

Figura 3.13. Diagrama de colaboración SeleccionarEje

El diagrama de la Figura 3.13 se interpreta de la forma siguiente:

1. El mensaje SeleccionarEje se envía a una instancia de CoordinadorProyecto. La instancia corresponde al mensaje y crea una Selección.
2. El objeto Selección envía el mensaje ActivarEje a la instancia SelecciónEje. La instancia corresponde al mensaje y crea un eje activo.

Diagrama de Colaboración para la operación ConsultarVersiones

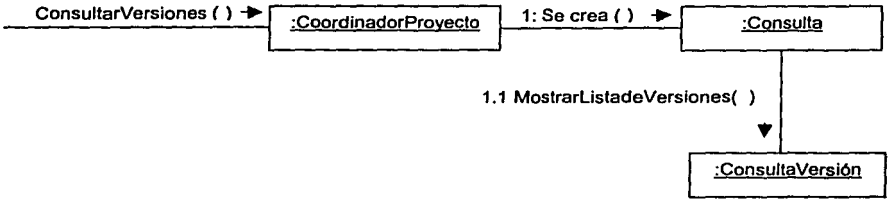

Figura 3.14. Diagrama de colaboración *ConsultarVersiones*

El diagrama de la Figura 3.14 se interpreta de la forma siguiente:

1. El mensaje ConsultarVersiones se envía a una instancia de CoordinadorProyecto. La instancia corresponde al mensaje y crea una consulta.
2. El objeto Consulta envía el mensaje MostrarListadeVersiones a la instancia ConsultaVersión. La instancia corresponde al mensaje y muestra la lista de versiones solicitada.

Diagrama de Colaboración para la operación SelecionarVersión

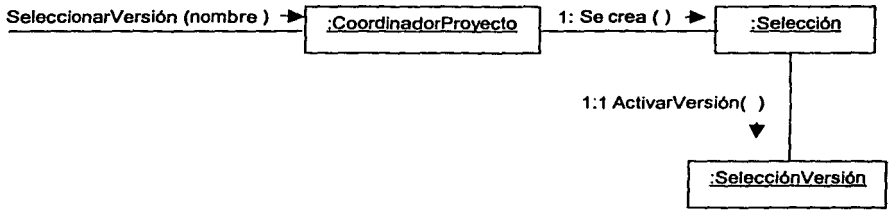

Figura 3.15. Diagrama de colaboración SeleccionarVersion

El diagrama de la Figura 3.15 se interpreta de la forma siguiente:

1. El mensaje SeleccionarVersión se envía a una instancia de CoordinadorProyecto. La instancia corresponde al mensaje y crea una selección.
2. El objeto Selección envía el mensaje ActivarVersión a la instancia SelecciónVersión. La instancia corresponde al mensaje creando una versión activa del eje activo.

Diagrama de Colaboración para la operación ConsultarCaracterísticas

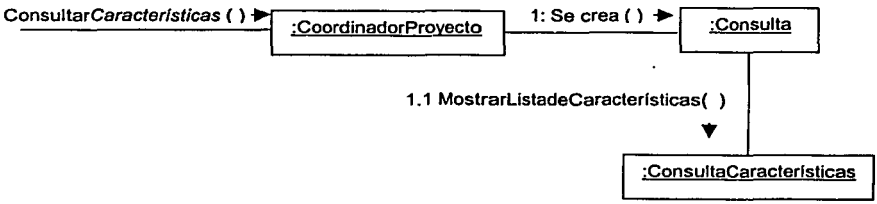

Figura 3.16. Diagrama de colaboración *ConsultarCaracterísticas*

El diagrama de la Figura 3.16 se interpreta de la forma siguiente:

1. El mensaje ConsultarCaracterísticas se envía a una instancia de CoordinadorProyecto. La instancia corresponde al mensaje y crea una consulta.
2. El objeto Consulta envía el mensaje MostrarListadeCaracterísticas a la instancia ConsultaCaracterísticas. La instancia corresponde al mensaje mostrando la lista de características que corresponde al eje activo.

Diagrama de Colaboración para la operación VerificarDimensiones

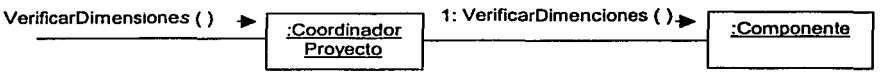

Figura 3.17. Diagrama de colaboración *VerificarDimensiones*

El diagrama de la Figura 3.17 se interpreta de la forma siguiente:

1. El mensaje VerificarDimensiones es enviado a una instancia de CoordinadorProyecto. La instancia corresponde al mensaje y lo manda a una instancia Componente que hace la verificación.

Diagrama de Colaboración para la operación DibujarModelo3D

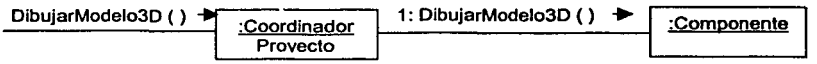

Figura 3.18. Diagrama de colaboración *DibujarModelo3D*

El diagrama de la Figura 3.18 se interpreta de la forma siguiente:

1. El mensaje DibujarModelo3D es enviado a una instancia de CoordinadorProyecto. La instancia corresponde al mensaje y lo manda a una instancia de Componente.

Diagrama de Colaboración para la operación GuardarModelo3D

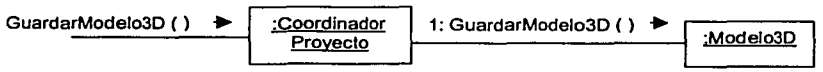

Figura 3.19. Diagrama de colaboración *GuardarModelo3D*

El diagrama de la Figura 3.19 se interpreta de la forma siguiente:

1. El mensaje GuardarModleo3D es enviado a una instancia de CordinadorProyecto. La instancia corresponde al mensaje y lo manda a una instancia de Modelo3D que crea el archivo .

3.1.10 Diagramas de Clases

En el Diagrama de Clases se muestran las clases que participarán en la construcción del sistema de cómputo. Para construirlo es necesario identificar las clases analizando los diagramas de Colaboración. Posteriormente se dibujan en el diagrama y se agregan los atributos (los atributos se obtienen del Modelo Conceptual). Después se agregan los nombres de los métodos analizando los diagramas de colaboración y se agregan las líneas de navegabilidad (Ver Anexo A.10).

En la figura 3.20 se muestra el diagrama de clases obtenido para la *Interfaz:*[2]

[2] Nota: Las operaciones ConsultarEjes y SeleccionarEje en el diagrama de clases se definen como ConsultarComponentes y SeleccionarComponente respectivamente.

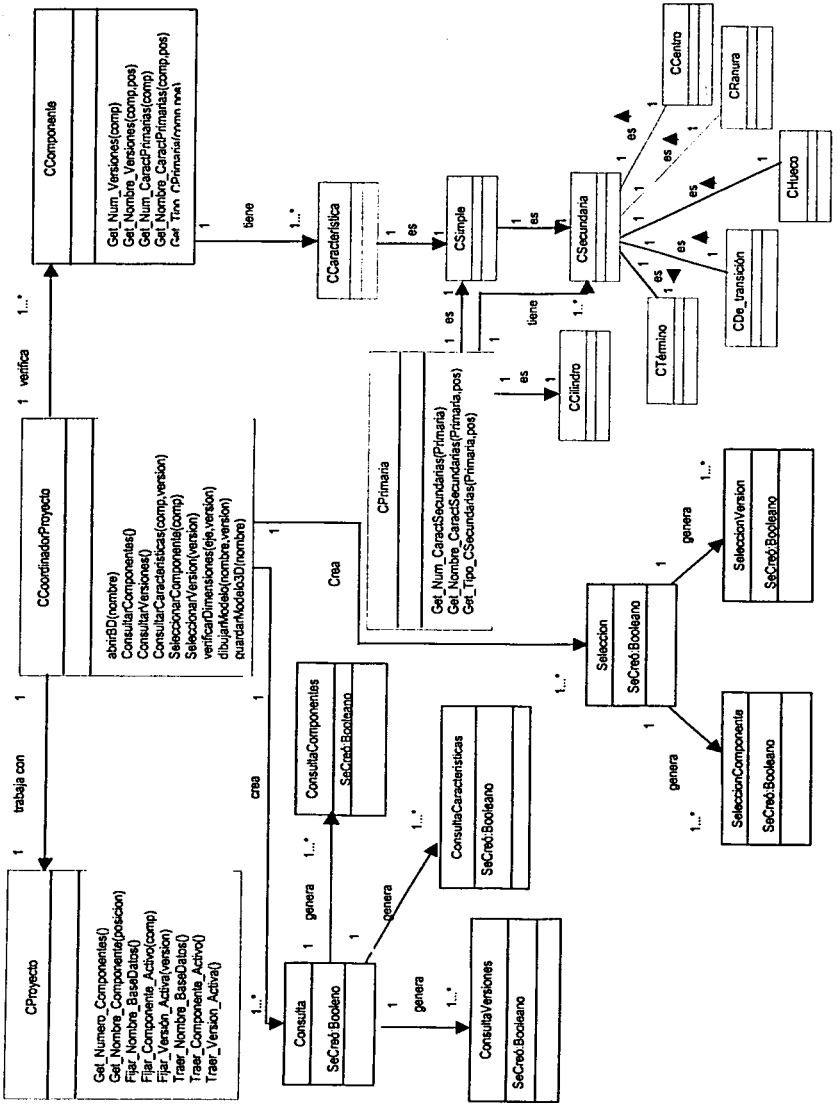

Figura 3.20. Diagrama de Clases propuesto

Para explicar el Diagrama de Clases mostrado en la figura 3.20 se ha tomado como ejemplo una parte de él (Figura 3.21):

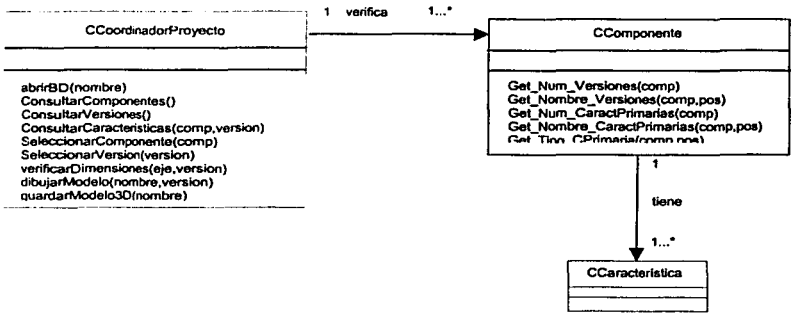

Figura 3.21. Diagrama parcial del Diagrama de Clases propuesto

En la figura 3.21 la clase CCoordinadorProyecto tiene varios métodos tales como: abrirBD, consultarComponentes, seleccionarComponente, etc. y es la clase principal que controlar el sistema. La clase CCoordinadorProyecto manda un mensaje a una ó muchas instancias de la clase CComponente para que ésta verifique u obtenga el número de versiones de un componente (eje) ó el número de Características Primarias, etc.

La clase CComponente utiliza la clase Característica para poder obtener el número de las Características Primarias ó el tipo de cada una de ellas.

CAPÍTULO 4
IMPLEMENTACIÓN DE LA INTERFAZ

El capítulo anterior habla sobre el análisis y el diseño de un modelo para la *Interfaz*. En este capítulo se da una breve explicación de las características de las herramientas usadas para la construcción de la *Interfaz*.

4.1 Herramientas usadas

Las herramientas usadas para el desarrollo de la *Interfaz* son: ObjectStore para construir la base de datos o MP, Visual C++ 6 para crear el código necesario que permita lograr los objetivos de la *Interfaz* y SolidWorks para crear el Modelo 3D. A continuación se describen estas herramientas.

4.1.1 ObjectStore

ObjectStore es un sistema potente para crear Bases de Datos Orientadas a Objetos. Las Bases de Datos pueden ser programadas en C++, Java o ActiveX, siendo estos lenguajes de programación orientados a objetos. Utilizando cualquiera de los lenguajes mencionados se pueden almacenar los objetos y tener acceso a la información contenida en una Base de Datos.

ObjectStore tiene una arquitectura heterogénea, por lo que puede encontrarse en diferentes sistemas operativos (servidores), puede trabajar en red y permite el acceso de clientes en diferentes plataformas como se observa en la figura 4.1:

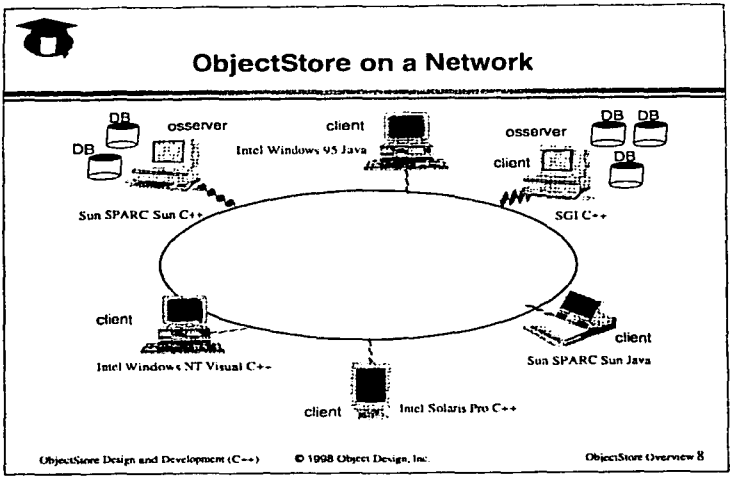

Figura 4.1. ObjetcStore en red y sus clientes

Cada Base de Datos creada se almacena como un archivo al que hay que especificarle una ruta de acceso y un nombre. Si se desea leer, modificar o agregar información en la base de datos se debe utilizar un conjunto de instrucciones u operaciones propias de ObjectStore. A este conjunto de instrucciones se le da el nombre de transacción.

Como ejemplo de dos transacciones diferentes se presenta la figura 4.2:

```
# include <ostore/store.hh>
int main ()
{
        objectstore : : inicialize( );
        os_database* firstDB = os_database : : open("flights.db",0,0664);

        OS_BEGIN_TXN(t1,0,os_transaction::update)
        {
                //En esta parte de la transacción se debe incluir el código que
                //para crear nueva información.
        }

        OS_END_TXN(t1)

        OS_BEGIN_TXN(t2,0,os_transaction::read_only)
        {
                //En esta parte de la transacción se incluye el código que
                //permita leer la información.
        }

        OS_END_TXN(t2)

        FirstDB -> close();
}
```

Figura 4.2. Uso de transacciones.

El nombre de la transacción en el BEGIN y END debe ser igual y debe ser único dentro del contexto en el cual es usada la transacción.

Algunas de las herramientas más destacadas con que cuenta ObjectStore son las siguientes:

- Inspector de ObjectStore. Es una herramienta que permite examinar una Base de Datos. A través de ella se puede observar y modificar el contenido de una Base de Datos, ver su estructura y ver su esquema en diferentes notaciones.
- Diseñador de Bases de Datos y un Asistente de ayuda para crear el diseño.
- Herramienta Gráfica para el análisis de la Base de Datos.

4.1.2 Visual C++ 6

El lenguaje C++ fue desarrollado por Bjarne Stroustrup, de los laboratorios Bell, en los años ochenta [Deitel H., Deitel P.,1994]. C++ es una mejora o extensión del lenguaje C, la diferencia principal entre C y C++ es que C es un lenguaje estructurado, mientras que C++ es un lenguaje orientado a objetos, por lo tanto mediante los objetos modela la realidad y da solución a un determinado problema.

C++ se desarrolló originalmente para resolver simulaciones muy rigurosas por lo que, debido a su potencialidad y a la de cualquier otro lenguaje orientado a objetos, crear software con este tipo de lenguajes reduce considerablemente el tiempo de desarrollo.

Visual C++ 6 es una herramienta para crear aplicaciones gráficas para Windows 95/98 o Windows NT. Una aplicación gráfica implica el uso de ventanas, botones, diálogos, mensajes, etc., llamados controles, que se pueden manipular mediante la programación orientada a objetos utilizando el lenguaje C++.

Visual C++ 6 se encuentra en tres configuraciones diferentes: Ediciones Básicas, Profesionales y para Desarrollo.

La Edición Básica permite dominar fácilmente el lenguaje C++ y contiene casi todas las prestaciones de la Edición Profesional.

La Edición Profesional ofrece servicios y controles para plataformas Win32, incluyendo Windows 95/98 y NT.

La edición para Desarrollo ofrece todos los servicios de la Edición Profesional y, además, permite a los desarrolladores crear aplicaciones cliente/servidor para Internet e incluso Intranet. También permite trabajar con bases de datos utilizando SQL [Pappas C., Murria W., 1999].

El hardware y software recomendados para trabajar con Visual C++ 6 son los siguientes:

PC con procesador Pentium, a velocidad de 200MHz
32 MB de memoria RAM
1 GB de espacio en disco duro
Monitor Super VGA
Sistema operativo: Windows 95/98 o Windows NT.

4.1.3 SolidWorks 99

SolidWorks es un modelador de sólidos cuyo fin es permitir el diseño de planos, modelos en tres dimensiones o ensambles que se almacenan en archivos para que, posteriormente, se puedan editar o imprimir. También a partir de dichos archivos se puede analizar la información para generar reportes, generar código, llevar a cabo un análisis de costos y tener acceso a otros sistemas que SolidWorks permite como a sistemas de simulación, STEP, IGES, CAM, etc.

Con SolidWorks se puede dar a los modelos sólidos color, textura, dimensiones, acabados, etc. Además tiene una API (Interfaz de Programación de Aplicaciones) que contiene las funciones de SolidWorks: crear un círculo, crear una línea, hacer un corte, etc[1]. Estas funciones se pueden activar desde Visual Basic, C, C++ o desde archivos de macros de SolidWorks para ir construyendo modelos sólidos a través de un programa.

Para crear un modelo sólido de un eje de transmisión usando un programa, lo primero es lograr establecer una conexión entre SolidWorks y, en este caso, Visual C++. Después se llama a las funciones que tiene SolidWorks usando la notación especificada en el API de SolidWorks para lograr que lo que se programe en Visual C++ se vea en SolidWorks. Por ejemplo, si se llama a la función createline2(p1,p2) donde p1 es el punto inicial de la línea que se desea crear y p2 el punto final, SolidWorks desplegará una línea que tenga como punto inicial p1 y como punto final p2. Algunas funciones llegan a ser obsoletas entre las diferentes versiones de SolidWorks, por ejemplo, una función que forma parte del API de la versión 98 plus puede no funcionar en la versión 99. Cabe aclara que en el desarrollo de esta tesis se ha utilizado la versión 99 de SolidWorks.

[1] Consultar el Anexo B para mayor información sobre las funciones de SolidWorks 99

SolidWorks trabaja con tres tipos de archivos: archivos para dibujos, para piezas y para ensambles. La *Interfaz* únicamente utiliza archivos para dibujar piezas y estos archivos tienen un formato o extensión .sldprt.

4.2 Descripción de la Interfaz

La *Interfaz* tiene que leer, del Modelo del Producto (MP), las características primarias (cilindros), las características secundarias (chaflanes, radios, huecos, etc.) y las dimensiones que definen al eje del cual se quiere obtener el Modelo 3D.

Al ejecutarse el programa el sistema verifica que el modelador de sólidos (SolidWorks) esté disponible, es decir, que haya sido ejecutada dicha aplicación. Si no es así, el sistema manda un mensaje al usuario indicando que no puede establecer comunicación con SolidWorks y, posteriormente lo abrirá (figura 4.3).

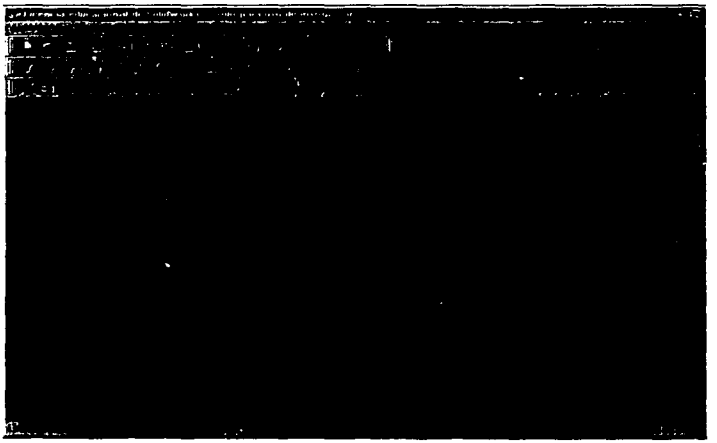

Figura 4.3. Modelador de Sólidos *"SoliWorks"*

Al verificar primeramente que exista conexión con SolidWorks, el programa se asegura de que va a poder realizar su objetivo (crear el Modelo 3D de un eje), de lo contrario no podrá ejecutarse el programa.

Ventana Principal de la Interfaz

La ventana principal contiene (Ver figura 4.4):

- Un menú con las siguientes opciones:
  - Modelo del Producto
    - Abrir MP
    - Salir
  - Ver
    - Barra de herramientas
    - Barra de estado
  - Consultar
    - Ejes
    - Versiones
    - Características
- Una barra de herramientas que contiene algunas opciones del menú para una mayor rapidez en su ejecución.
- Una caja de texto, en donde se puede observar el nombre de la base de datos (MP) una vez que se haya abierto ésta.
- Un botón para consultar los ejes definidos en la base de datos (MP) que se encuentre abierta, con el fin de seleccionar uno de ellos. Si aún no se ha abierto ninguna base de datos, el botón se encontrará desactivado.

- Un botón para consultar e indicar una versión del eje ya seleccionado. Si aún no se ha seleccionado algún eje, el botón se encontrará.

- Un botón para consultar las características primarias y secundarias que componen al eje seleccionado. Si aún no se ha seleccionado una versión del eje, el botón se encontrará desactivado. Además, si no se han especificado características primarias ni características secundarias del eje, la consulta estará vacía.

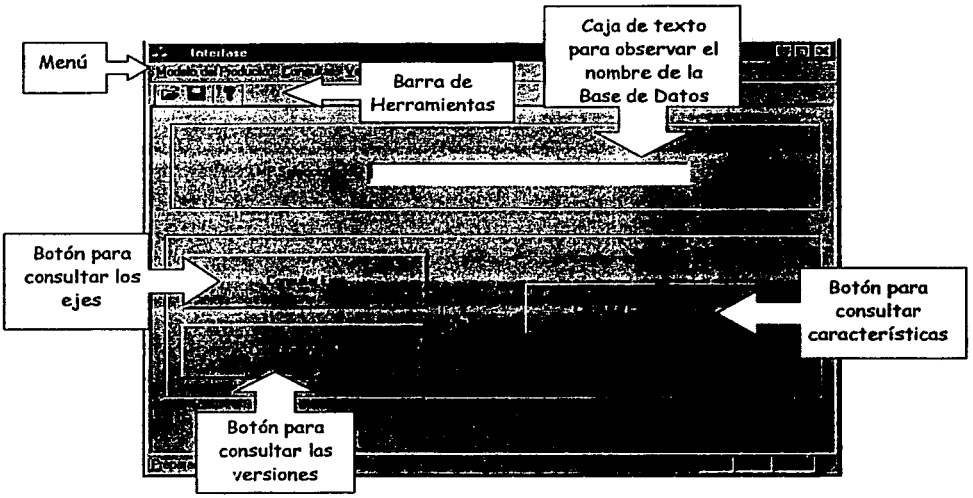

Figura 4.4. Ventana Principal

Al hacer clic en el botón *Abrir MP* del menú *Modelo del Producto*, se abrirá una ventana de navegación que permitirá al usuario seleccionar el archivo de la base de datos donde se encuentra almacenado el *Eje* que desea dibujar. Lo anterior se ilustra en la figura 4.5:

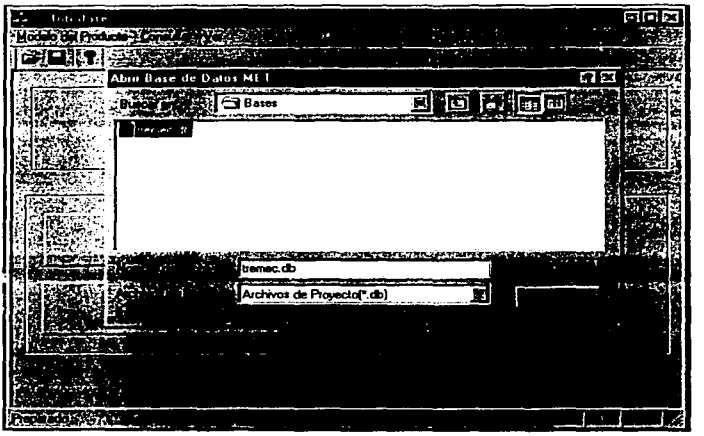

Figura 4.5. Ventana de navegación

Una vez que el usuario haya abierto un MP, y se haga clic en el botón *Consultar Ejes, se* mostrará la ventana de la figura 4.6, para que el usuario pueda seleccionar un *Eje*.

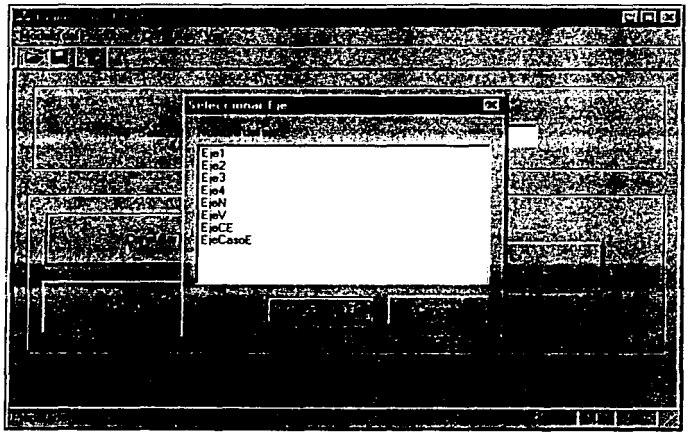

Figura 4.6. Ventana para la consulta de ejes

Después de haber seleccionado un *Eje* y hacer clic en el botón *Consultar Versiones,* el sistema mostrará la ventana de la figura 4.7 que permitirá al usuario seleccionar la versión que desee del *Eje*. Nótese que las versiones se denotan con la letra mayúscula "D" seguida de un número consecutivo, comenzando desde cero y después del nombre dado al eje y un subguión (*Eje1_D0, Eje1_D1, etc.*). Siempre que se defina un eje la versión por default es la cero. Las diferentes versiones de un solo eje solamente tienen de diferencia las dimensiones de sus características.

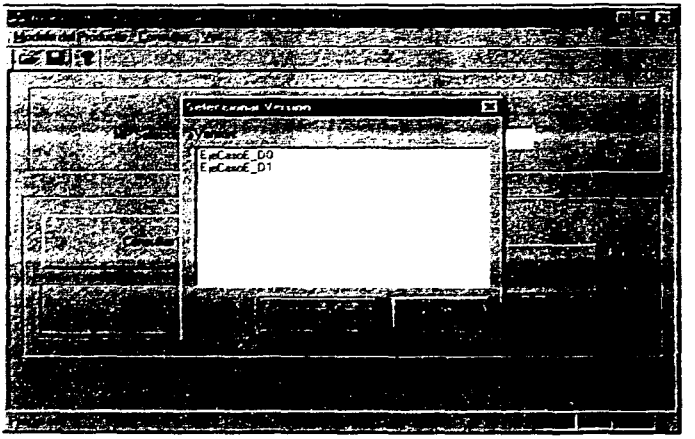

Figura 4.7. Ventana para la consulta de versiones

Para que el usuario se cerciore de que desea crear el *Modelo 3D* del eje y versión seleccionados debe hacer clic en el botón *Consultar Características*, esto mostrará la ventana de la figura 4.8:

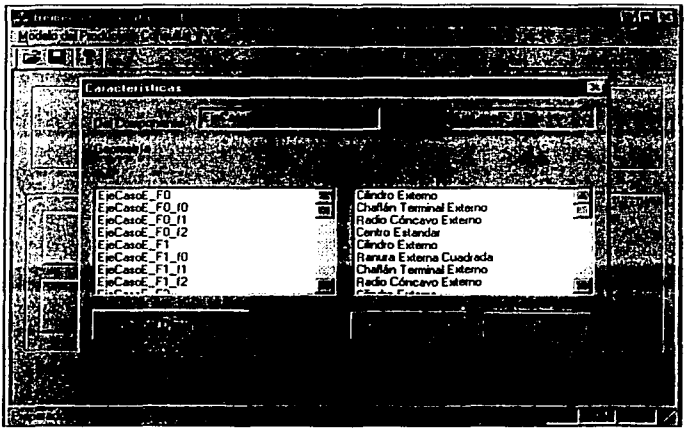

Figura 4.8. Ventana para consultar las características

Ahora el sistema está listo para crear el Modelo 3D al hacer clic en el botón Crear Modelo 3D de la misma ventana.

Una vez terminado el Modelo, el sistema pregunta si desea guardar dicho Modelo, si la opción fue afirmativa entonces le pedirá un nombre para el Modelo y una ruta específica para guardar el archivo.

CAPÍTULO 5
CASO DE ESTUDIO

En el presente capítulo se muestran algunos otros programas que forman parte de SADET, como el MP [Mirón, 2001] y el Modelador de celdas de manufactura ó MM [Ayala, 2001], con el objetivo de explicar el funcionamiento de éstos ya que son necesarios para poder crear el modelo de un eje de transmisión. Además se muestra cómo se debe introducir la información en el MP tomando como ejemplo del caso de estudio un eje proporcionado por un colaborador industrial. Posteriormente se muestran los pasos que se deben seguir para comprobar que es posible crear el modelo 3D.

5.1 Presentación del Caso de Estudio

Como caso de estudio se utiliza una flecha secundaria de una transmisión comercial diseñada y fabricada por un colaborador industrial (Figura 5.1).

Figura 5.1 Flecha secundaria de una transmisión comercial

En la figura 5.2 se presenta la ventana principal del *Modelador de ejes de transmisión (met)* [Mirón, 2001] que permite introducir información sobre el eje en la base de datos (MP) a través de menús, íconos y ventanas:

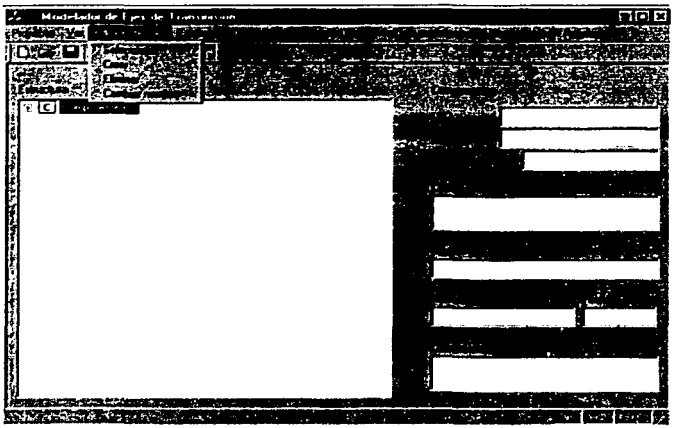

Figura 5.2. Ventana principal del Modelador de ejes de transmisión (met)

Además *SADET* cuenta con otro sistema de base de datos llamado Modelo de Manufactura (MM) [Ayala, 2001] que almacena la información sobre las fábricas y herramientas adecuadas para que sea posible la manufactura de dicho eje; es decir, dependiendo de las características de cada eje que se desee manufacturar, se analizarán los diferentes tipos de fábricas y herramientas que estén representados en la base de datos (MM), con el objetivo de determinar cuál de ellas es la adecuada para manufacturar el eje deseado. Para poder ingresar datos al MM existe el programa llamado *Modelador de celdas de manufactura* (mcm) [Ayala, 2001]. En la figura 5.3 se muestra la ventana principal del mcm:

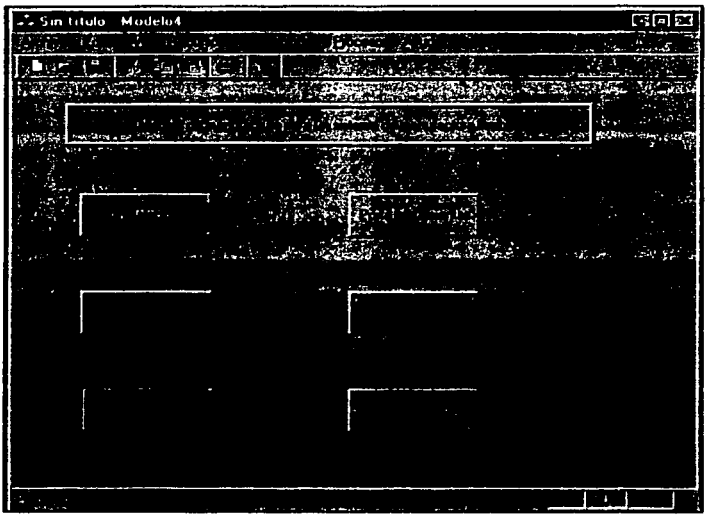

Figura 5.3. Ventana principal del Modelador de celdas de manufactura (mcm)

Como parte del rediseño se requiere obtener el Modelo 3D del eje, por lo que se ha desarrollado el trabajo propuesto *Interfaz* que requiere la información contenida en el *Modelo del Producto*. En la figura 5.4 se muestra la ventana principal:

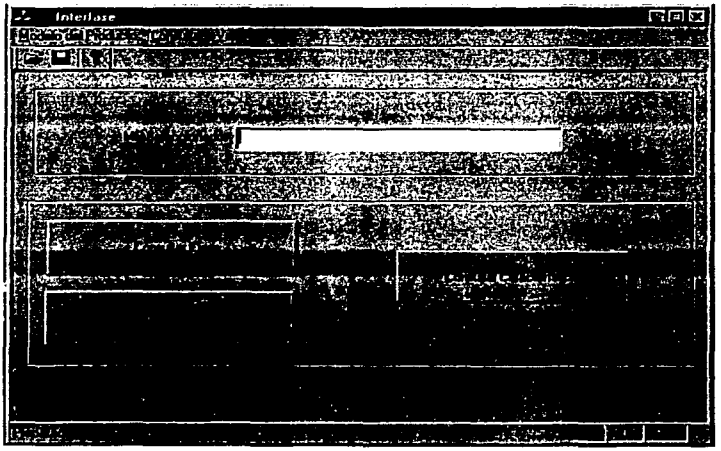

Figura 5.4. Ventana principal del sistema *Interfaz (int)*

5.2 Desarrollo del Caso de Estudio

Para obtener el Modelo 3D, es indispensable haber utilizado el *Modelador de ejes de transmisión* con el fin de guardar la información del eje y sus características en el *Modelo del Producto.* A continuación, en la figura 5.5 se muestra la ventana que permite crear un eje en el MP:

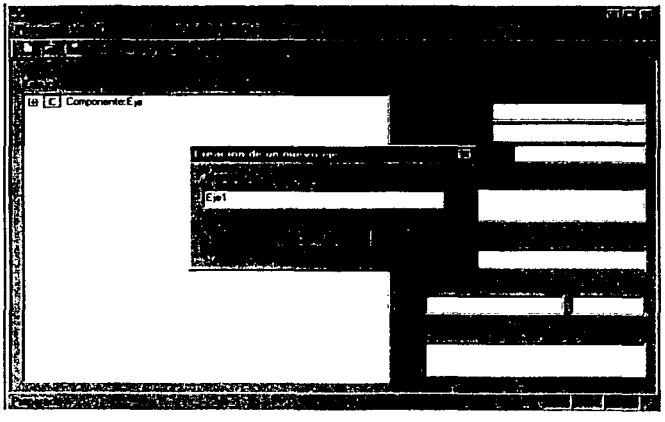

Figura 5.5. Ventana que permite crear un eje en el MP

Esta ventana se abre al hacer clic en el menú *Edición de Ejes – Crear* y en ella se debe indicar el nombre que se le quiere dar al eje que se va a rediseñar (en este caso llamado *Eje1*). Siempre que se crea un eje se crea también una versión denominada *"D0"*, si se desea crear otra versión del eje entonces se hace

manualmente de forma similar a la creación de un eje y se denominarán como *"D1", "D2"*, etc (*Eje1_D0, Eje1_D1, etc.*). Las diferentes versiones de un eje cuentan con las mismas características, únicamente cambian los valores de las dimensiones.

Después se deben crear las características primarias (cilindros) que formarán al eje a través del menú *Estructura –Características Primarias – Crear.* Cada característica primaria queda registrada con la letra "F" (mayúscula) y un número consecutivo, comenzando desde el cero, después del nombre del eje y un guión bajo (*Eje1_F0, Eje1_F1, etc.*). A continuación se crean las características secundarias utilizando ventanas como la mostrada en la figura 5.6:

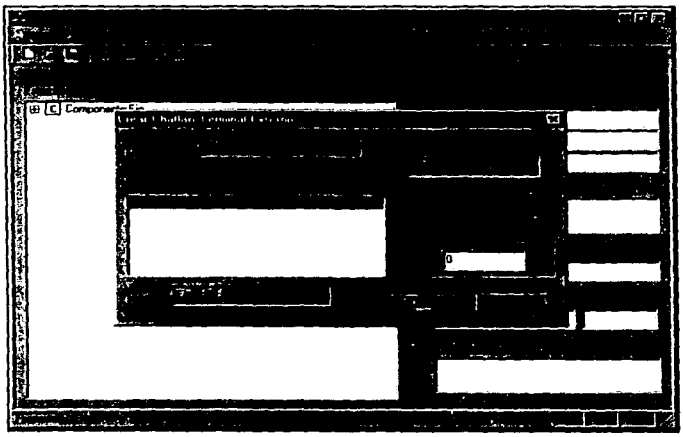

Figura 5.6. Ventana que permite crear las características de un eje en el MP

Esta figura muestra una ventana donde se observa la existencia de un eje llamado *Eje1* y cinco cilindros, a través de ella se puede crear la característica secundaria Chaflán Terminal Externo mediante el menú *Estructura –Características Secundarias –Crear -Chaflán Terminal Externo*, siempre y cuando se indique cuál característica primaria llevará dicho Chaflán. De igual forma se crean todas las características secundarias que formarán al eje.

Por último deben indicarse las dimensiones de cada característica, utilizando una ventana como la mostrada en la figura 5.7:

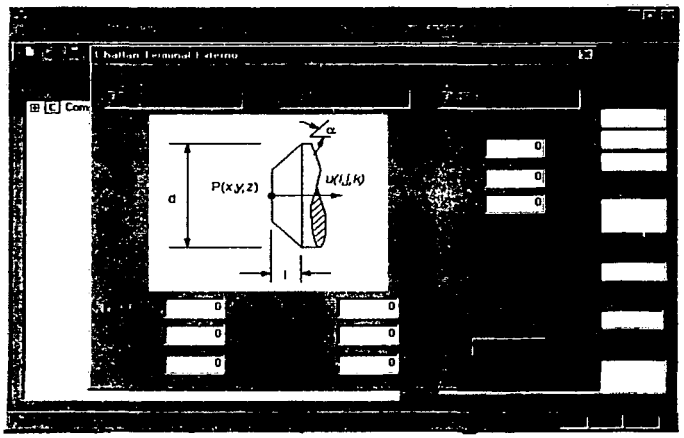

Figura 5.7. Ventana que permite introducir las dimensiones de las características de un eje

Es importante que los datos registrados en el MP estén correctos y completos para que pueda crearse el Modelo 3D.

Una vez hecho esto es posible utilizar la *Interfaz* para crear el Modelo 3D en el modelador SolidWorks siguiendo los siguientes 5 pasos: (1) figura 5.8, (2) figura 5.9, (3) figura 5.10, (4) y (5) figura 5.11.

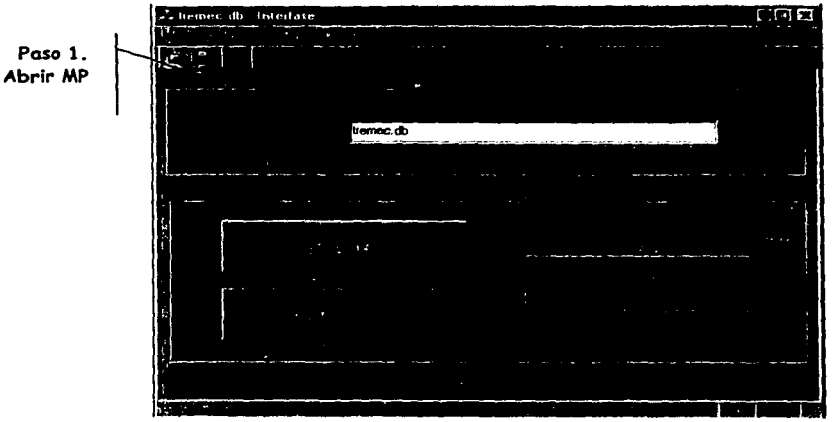

Figura 5.8. Ventana que muestra el paso no. 1 para obtener el Modelo 3D

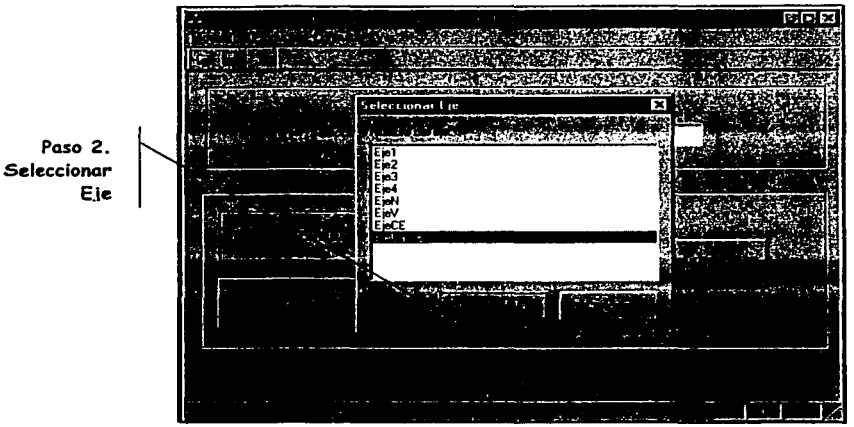

Figura 5.9. Ventana que muestra el paso no. 2 para obtener el Modelo 3D

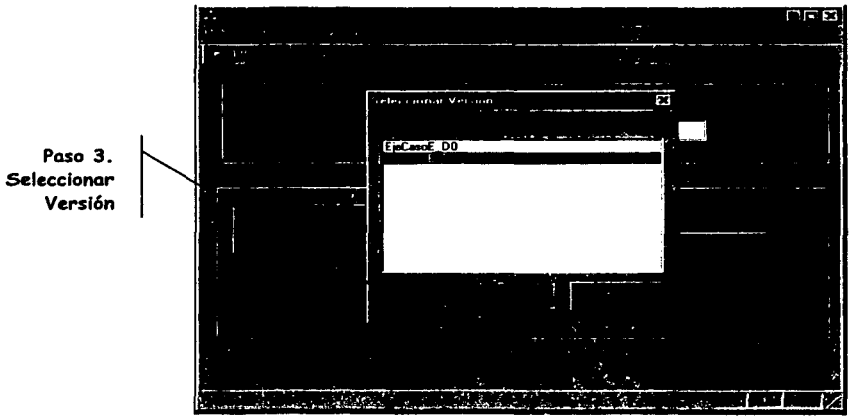

Figura 5.10. Ventana que muestra el paso no. 3 para obtener el Modelo 3D

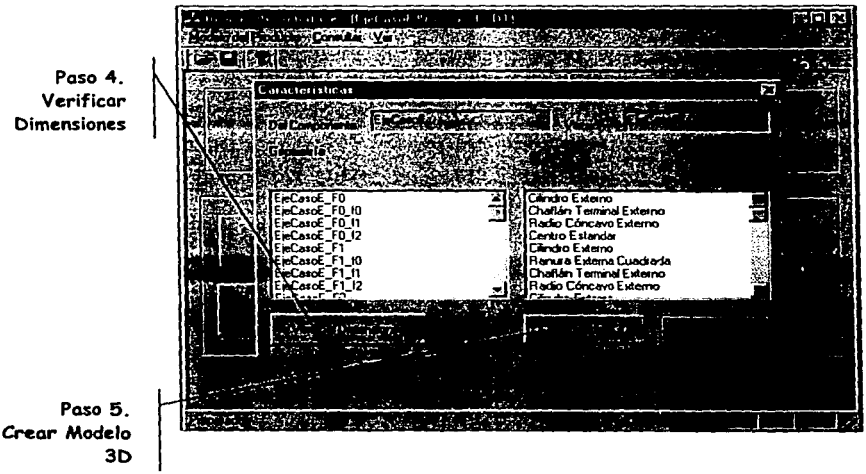

Figura 5.11. Ventana que muestra el paso 4 y 5 para obtener el Modelo 3D

El Modelo 3D resultante de este proceso, que se genera utilizando la información del MP en el modelador de sólidos (SolidWorks), es el que a continuación se muestra en las figuras 5.12 y 5.13:

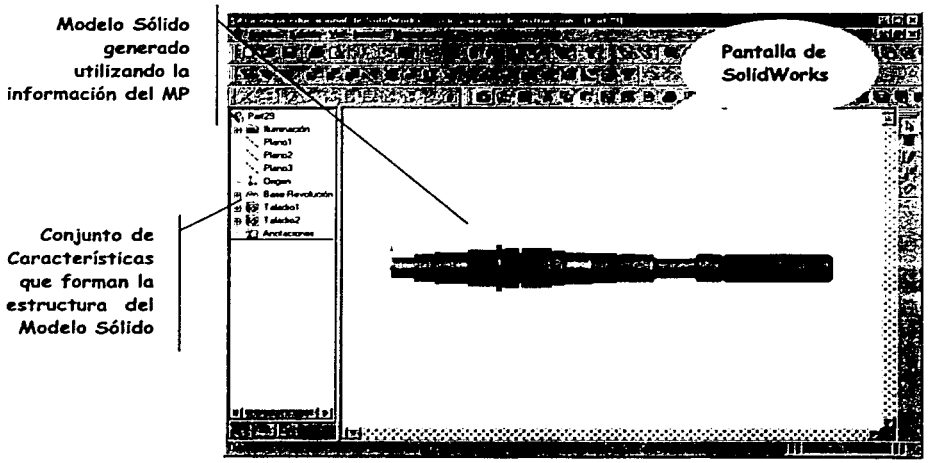

Figura 5.12. Modelo 3D del Caso de Estudio

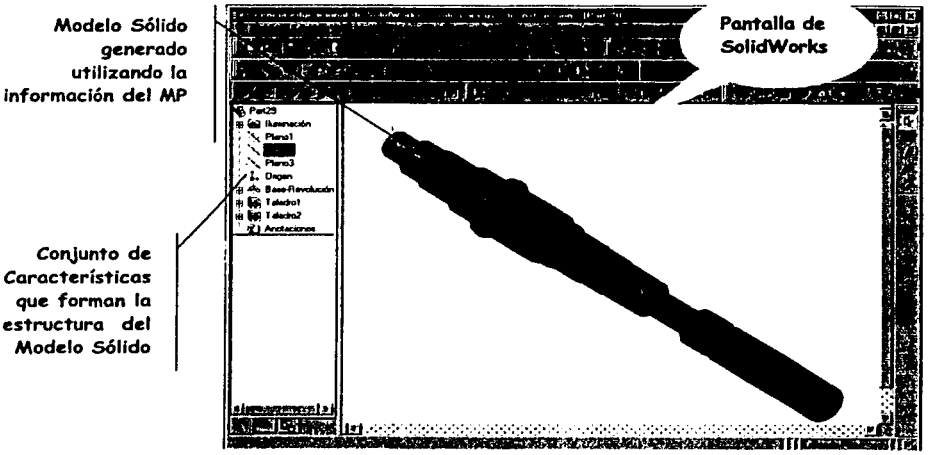

Figura 5.13. Modelo 3D del Caso de Estudio (vista isométrica)

En las figuras anteriores se puede observar que el modelador de sólidos (SolidWorks) presenta el modelo del eje tomado como caso de estudio en una pantalla dividida en dos partes. Del lado derecho se observa el modelo del eje como un sólido y a la izquierda se observa la estructura de las características (features) que lo componen. Dicha estructura se presenta desglosada y aparece de acuerdo al orden en que se dibuja cada característica.

CAPÍTULO 6
CONCLUSIONES Y TRABAJOS FUTUROS

6.1 Conclusiones

El desarrollo, documentación y prueba de la Interfaz se ha llevado a cabo de manera satisfactoria. La documentación se puede consultar en el capítulo 3 de este trabajo.

La Interfaz lee y convierte la información del MP al lenguaje de programación llamado Visual C++, proceso esencial en la realización de este trabajo.

La Interfaz indica al Modelador de Sólidos, en este caso SolidWorks, que comience a crear el modelo del eje activo mediante la función llamada DibujarModelo().

La Interfaz guarda el modelo creado en SolidWorks únicamente en el caso de que el usuario así lo requiera.

La metodología utilizada en el desarrollo de la *Interfaz* permite extender el código para poder representar, en un modelo 3D, características adicionales a las definidas actualmente en el MP. Para incluir otras características primeramente deben estar estructuradas en el MP, después se deben indicar en el modelo conceptual del UML para que posteriormente se consideren como una clase y pueda generarse el modelo sólido.

Con el uso de la herramienta UML en el desarrollo del análisis y el diseño de la *Interfaz* se ha podido documentar el proceso llevado a cabo para la construcción de la misma. Esta documentación es muy valiosa ya que ha proporcionado un

mejor entendimiento del problema debido al uso de diagramas y, consecuentemente, ha proporcionado una solución. Además permite a cualquier persona ajena al diseño del sistema comprender su desarrollo con el objetivo de auxiliar a los desarrolladores de algunos trabajos futuros que se pretenda construir.

Utilizando el análisis y diseño orientado a objetos y documentándolos con UML en el proceso de desarrollo de la *Interfaz*, se aprendió a modelar sistemas orientados a objetos y se logró obtener un modelo que cumple con los requerimientos planteados para el sistema.

Empleando modelos de información (MP y MM) no solo se logran analizar las características de los productos y de las herramientas necesarias para su manufactura, la *Interfaz* ha demostrado que se puede representar gráficamente dicha información. Por lo tanto la *Interfaz* es parte importante que complementa el proyecto *SADET* permitiendo visualizar la información contenida en el MP.

El API (Interfaz de Programación de Aplicaciones) de SolidWorks contiene funciones que definen la creación de una línea, de un arco, de chaflanes, etc., utilizadas para crear planos, piezas o ensambles. Estas funciones facilitaron en gran mediada la programación para construir los modelos 3D.

Siendo Solidworks un sistema CAD que facilita el diseño mecánico (planos, piezas y ensambles) es programado en diferentes partes del mundo, sin embargo, la mayoría de los desarrolladores utilizan el lenguaje de programación Visual Basic, que no es orientado a objetos, por lo que programar utilizando Visual C++ ha sido un logro importante. Respecto a la *Interfaz* desarrollada, la experiencia obtenida en el manejo del API de SolidWorks es un antecedente importante para futuras aplicaciones con las cuales se desee utilizar otros sistemas CAD.

El proyecto *SADET* es un sistema CAE basado en modelos de información. Actualmente algunos de los sistemas que lo forman ya están terminados como el modelo de manufactura (MM) [Ayala, 2001], el modelo del producto (MP) [Mirón,

2001], el modelador de celdas de manufactura (mcm) [Ayala, 2001] y el modelador de ejes de transmisión (met) [Mirón, 2001]. Los sistemas restantes se encuentran en desarrollo.

6.2 Trabajos Futuros

Tomando como base el sistema *Interfaz* se pretende ampliarlo para que pueda representar el modelo sólido de cualquier pieza rotacional maquinada, piezas prismáticas y partes de mayor complejidad como moldes de inyección. Este trabajo se desarrollará como parte de tres investigaciones doctorales y una tesis de maestría.

Tomando como base el sistema *Interfaz* se pretende implementar un sistema que haga lo contrario, es decir, que represente en un modelo del producto el conjunto de características que conforman un modelo 3D.

ANEXO A
LENGUAJE UNIFICADO PARA LA CONSTRUCCIÓN DE MODELOS: UML

Lenguaje Unificado de Construcción de Modelos es la traducción al español de las siglas UML. Fue creado en 1994 por Grady Booch y Jim Rumbaugh y se define como un lenguaje o notación estándar usado para construir modelos orientados a objetos, "permite especificar, visualizar y construir los artefactos de los sistemas de software" [Larman Craig, 1999]. Tiene un vocabulario y unas reglas que nos ayudan a representar los sistemas de software de una forma conceptual y física. Con el UML especificamos, visualizamos, construimos y documentamos los componentes de dichos sistemas. Como su mayor parte se compone de esquemas o diagramas, el UML es una herramienta poderosa que permite entender la problemática de un sistema y realizar un diseño óptimo de éste antes de empezar su construcción.

Es importante especificar que el UML no define un proceso de desarrollo, es decir, el programador debe seleccionar un proceso de desarrollo conveniente antes de comenzar a trabajar con el UML. El UML solo estandariza la notación o la forma de crear y leer modelos, pero no nos dice qué modelos crear ni cuándo crearlos. El proceso de desarrollo bien estructurado es el que nos indica qué componentes debe tener el sistema.

El UML es un lenguaje gráfico que ayuda a una mejor interpretación y entendimiento de un sistema de software ya que detrás de los símbolos hay una semántica bien definida. Por ejemplo, en la presentación final del proyecto asistirán personas totalmente ajenas al desarrollo de software y será difícil para ellas lograr

una comprensión clara del trabajo, o bien para los propios desarrolladores es necesario crear un modelo específico que les ayude a resolver el problema.

Una vez terminado el proceso de desarrollo a través de un modelo UML, el modelo puede ser directamente transferido a cualquier lenguaje de programación como Java o C++; también se puede transferir a tablas en una base de datos relacional o a una base de datos orientada a objetos. Durante este proceso de desarrollo los conceptos que se comprenden mejor si se representan gráficamente están hechos en UML, mientras que los que se comprenden mejor textualmente se representan en un lenguaje de programación.

A continuación se describe la notación UML:

Anexo A.1. *ACTORES*

Un actor es un agente externo al sistema que por lo general inicia un proceso. Este proceso comienza cuando el sistema es estimulado por un actor con el objetivo de recibir una respuesta de dicho sistema, hasta completar una tarea o proceso deseado. Comúnmente los actores son personas que utilizan el sistema, pero es posible que un actor sea otro sistema de cómputo o algún aparato eléctrico o mecánico.

Un actor es algo externo al sistema cuya función es estimularlo con el objeto de empezar un proceso, los actores pueden ser:

- Papeles que desempeñan las personas
- Sistemas de cómputo
- Aparatos mecánicos o eléctricos

En UML un actor se representa con el esquema mostrado en la figura 1:

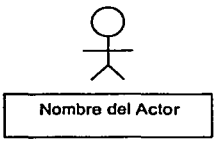

Figura 1. Notación UML de un actor

Anexo A.2. *CASOS DE USO*

Para comprender los requerimientos de un sistema se puede emplear una técnica llamada casos de uso. "El caso de uso es un documento narrativo que describe la secuencia de eventos de un actor (agente externo) que utiliza un sistema para completar un proceso" [Larman Craig, 1999].

Un caso de uso es la descripción de un proceso que debe efectuar el sistema y se compone de varios pasos individuales. Una forma para establecer los casos de uso es identificar primero a los actores y a partir de ellos definir los procesos en que participan, siendo estos procesos los casos de uso. Para mostrar la representación de un caso de uso (figura 2) se toma como ejemplo el sistema de una máquina registradora de productos de una tienda comercial, el actor es un cajero:

Figura 2. Notación UML de un Caso de Uso

Anexo A.3. *DIAGRAMA DE CASOS DE USO*

El diagrama de casos de uso es una representación de los pasos a seguir de un actor, estos pasos se llaman eventos. Cuando un actor ejecuta un evento sobre un sistema, el sistema manda una respuesta, que depende del evento aplicado, el actor vuelve a aplicar un evento y el sistema contesta, y así sucesivamente con el fin de completar un proceso. Un diagrama de caso de uso se construye como se muestra en la figura 3:

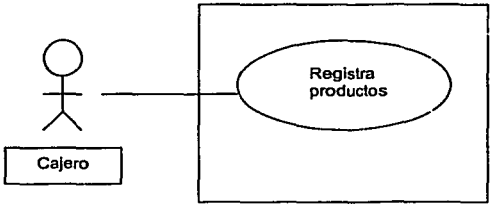

Figura 3. Notación UML de un Diagrama de Casos de Uso

Anexo A.4. *CASOS ESENCIALES EXPANDIDOS DE USO*

Los casos esenciales expandidos de uso son los casos de uso identificados anteriormente pero expresados en una forma desglosada. En UML se describen los pasos para completar un proceso (en este ejemplo el proceso es del registro para la venta de un producto) numerándolos en orden ascendente, en el lado izquierdo de una línea vertical divisora se escribe la acción de los actores y en el derecho la respuesta del sistema (ver figura 4).

Las relaciones del modelo conceptual se leen de izquierda a derecha y de arriba hacia abajo, la notación tiene el significado siguiente:

1 = uno
1 ... * = uno ó muchos

y se lee:

- Una tienda almacena uno o varios productos
- Una tienda almacena una caja registradora (se está hablando de una tienda pequeña)
- Uno o muchos productos son registrados en una caja registradora
- Un cajero registra un producto

Anexo A.6 *DIAGRAMAS DE SECUENCIA*

Un diagrama de secuencia describe a detalle los pasos que el actor debe realizar en el sistema para obtener la respuesta deseada. Este diagrama no describe cómo se lleva a cabo cada paso, sólo ilustra la secuencia de los pasos que deben realizarse para cada caso de uso (ver figura 6).

Acción de los Actores	Respuesta del Sistema
1. Inicia cuando un cliente llega a la caja para realizar el pago se sus productos.	
2. El cajero le indica al sistema que debe comenzar la captura de algunos productos.	
	3. El sistema está listo y pide los datos necesarios (tipo de producto, precio, etc.).

Figura 4. Notación UML de Casos Esenciales Expandidos de Uso

Anexo A.5. *MODELO CONCEPTUAL*

Un paso esencial en el análisis orientado a objetos consiste en descomponer el problema a resolver en conceptos u objetos individuales. Estos objetos o conceptos son representados en un diagrama llamado *modelo conceptual.* En UML, un modelo conceptual no incluye la representación de las operaciones del sistema, solamente muestra los conceptos, las asociaciones que hay entre los conceptos y los atributos de éstos. Siguiendo el ejemplo de la máquina registradora de productos, el modelo conceptual se representa como en la figura 5:

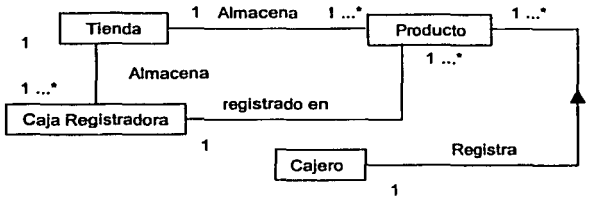

Figura 5. Notación UML de un Modelo Conceptual

Las relaciones del modelo conceptual se leen de izquierda a derecha y de arriba hacia abajo, la notación tiene el significado siguiente:

1 = uno
1 ... * = uno ó muchos

y se lee:

- Una tienda almacena uno o varios productos
- Una tienda almacena una caja registradora (se está hablando de una tienda pequeña)
- Uno o muchos productos son registrados en una caja registradora
- Un cajero registra un producto

Anexo A.6 *DIAGRAMAS DE SECUENCIA*

Un diagrama de secuencia describe a detalle los pasos que el actor debe realizar en el sistema para obtener la respuesta deseada. Este diagrama no describe cómo se lleva a cabo cada paso, sólo ilustra la secuencia de los pasos que deben realizarse para cada caso de uso (ver figura 6).

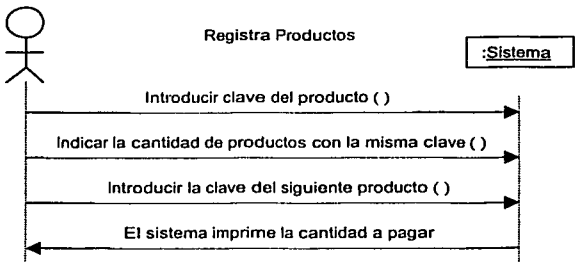

Figura 6. Notación UML de un Diagrama de Secuencia

Anexo A.7. *CONTRATOS*

Un contrato es un documento que describe qué sucederá en el sistema con cada operación descrita en el diagrama de secuencia. El contrato detalla cada operación, para poder comprender el comportamiento del sistema. En la figura 7 se muestra un ejemplo:

Contrato	
Nombre:	Introducir clave del producto ( )
Responsabilidades:	Registrar la venta de un producto. Imprimir el precio del producto.
Tipo:	Sistema
Excepciones:	Si la clave no es válida indicar que se cometió un error.
Precondiciones:	El sistema conoce la clave del producto
Poscondiciones:	- Si se trata de una nueva venta, se crea una venta. - Si se trata de una nueva venta, la nueva venta se asocia a la terminal (caja registradora), etc.

Figura 7. Notación UML de un Contrato

A continuación se describe cada sección de un contrato:

- Nombre: Se refiere al nombre de la operación.
- Responsabilidades: La acción que debe ejecutar la operación.
- Tipo: Se refiere al nombre del tipo (concepto, clase de software, etc.).
- Excepciones: Respuesta no esperada que puede dar el sistema.
- Precondiciones: Suposiciones sobre el estado del sistema antes de que se ejecute la operación.
- Poscondiciones: Es el estado del sistema que generalmente cambia después de aplicar la operación.

Anexo A.8. *CASOS REALES EXPANDIDOS DE USO*

Un caso real de uso describe el diseño concreto del caso de uso indicando cómo trabaja la interacción de bajo nivel (código) con la interfaz gráfica (ventanas que el usuario utiliza para trabajar con el sistema). Es similar a los *Casos Esenciales Expandidos de Uso,* pero haciendo referencia a las ventana que el usuario utilizará. Continuando con el ejemplo de la máquina registradora de productos, podemos observar la ventana principal del sistema en la figura 8 y en la figura 9 el caso real de uso correspondiente:

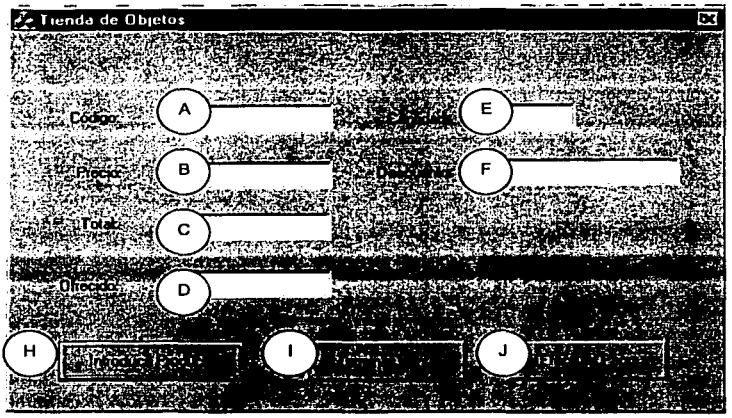

Figura 8. Ventana principal del sistema para una máquina registradora de productos

Acción de los Actores	Respuesta del Sistema
1. Inicia cuando un cliente llega a la caja para realizar el pago se sus productos.	
2. El cajero teclea el código de cada producto en A de la Ventana 1. Si hay más de un producto puede teclearla cantidad en E. Se oprime H después de capturar cada producto.	
	3. Calcula y presenta en C el total de la venta.

Figura 9. Notación UML de Casos Esenciales Expandidos de Uso

Anexo A.9. *DIAGRAMAS DE COLABORACIÓN*

El objetivo de los diagramas de colaboración es explicar cómo trabajan los objetos u conceptos definidos en el modelo conceptual con los mensajes para cumplir con sus tareas. Por lo tanto no es posible realizar estos diagramas si no se han elaborado previamente el modelo conceptual ni los contratos. El formato para representar un objeto es utilizar el nombre del objeto enmarcado con un recuadro; los mensajes se representan a través de líneas que apuntan en dirección del objeto que debe recibir dicho mensaje, tal como se muestra en la figura 10 y 11:

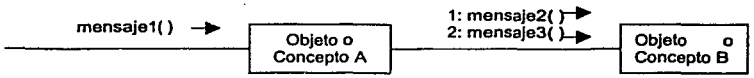

Figura 10. Diagrama de Colaboración

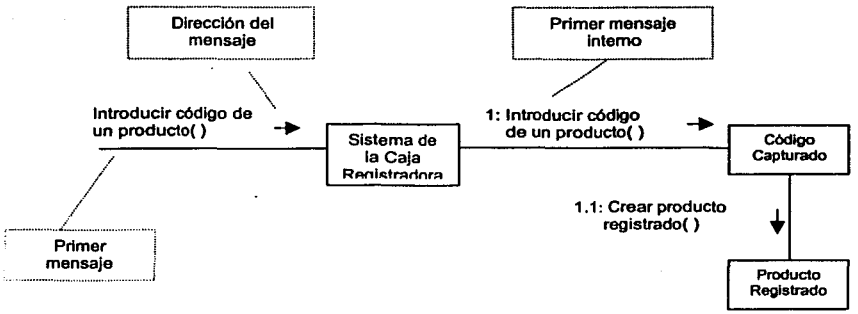

Figura 11 . Ejemplo de un Diagrama de Colaboración

1. El mensaje *Introducir código de un producto* se envía al objeto de *Sistema de la caja registradora*. El objeto corresponde al mensaje.
2. El objeto *Sistema de la Caja Registradora* envía el mensaje Introducir código de un producto al objeto *Código Capturado*.
3. El objeto *Código Capturado* crea un objeto *Producto Registrado*.

Anexo A.10. *DIAGRAMAS DE CLASES*

Como su nombre lo indica, este tipo de diagramas muestra las clases y sus especificaciones que van a participar para la construcción del software o sistema de cómputo. Gracias a la elaboración de los diagramas de colaboración se pueden definir las clases de la siguiente manera:

1. Primeramente se identifican las clases del sofware basándose en los objetos de los diagramas de colaboración. Se debe recordar que los objetos no pueden existir si no se ha creado una clase que defina su estructura, así que si sabemos qué objetos son necesarios sabremos también las clases necesitamos definir.
2. Las clases deben dibujarse en un diagrama similar al diagrama de colaboración.
3. Se dibujan los atributos de cada clase
4. Se agregan los métodos de cada clase.
5. Se debe indicar el tipo de dato a los que pertenece cada atributo y método.
6. Se agregan las asociaciones necesarias (uno a muchos, uno a uno, etc.)
7. Se agrega la dirección de las asociaciones.

En la figura 12 se muestra el diagrama de clases parcial del ejemplo tomado:

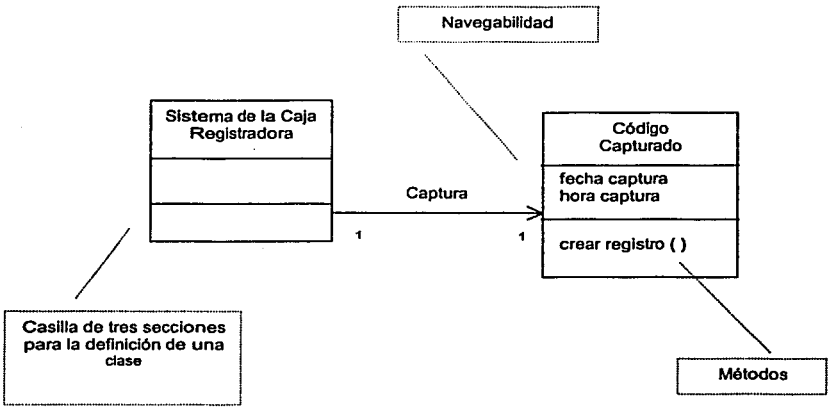

Figura 12. Ejemplo de un Diagrama de Clases

ANEXO B
FUNCIONES DEL API DE SOLIDWORKS 99

Existen alrededor de 300 funciones nuevas para SolidWorks 99. A continuación se listan algunas de ellas:

ModelDoc::CreateArc2	Esta función crea un arco basado en un punto central, un punto inicial, un punto final y una dirección.
ModelDoc::CreateCircle2	Esta función crea un círculo basado en un punto central y un punto en el círculo.
ModelDoc::CreateCircleByRadius2	Esta función crea un círculo basado en un punto central y un radio específico.
ModelDoc::CreateEllipse2	Esta función crea una elipse basada en un punto central y los puntos que especifican el mayor y el menor eje.
ModelDoc::CreateLine2	Esta función crea una línea 3D.
ModelDoc::CreatePoint2	Esta función crea un punto.
ModelDoc::Save2	Esta función salva el documento.
ModelDoc::SaveAs2	Esta función salva el documento con un nombre diferente.
SketchArc::GetCenterPoint	Esta función obtiene las coordenadas del punto central de un arco.
SketchArc::GetEndPoint	Esta función obtiene las coordenadas del punto final de un arco.
SketchArc::GetStartPoint	Esta función obtiene las coordenadas del punto inicial de un arco.
ModelDoc::GetType	Esta función obtiene el tipo de documento activo (dibujo, pieza o ensamble).
GetActiveDoc	Esta función activa un documento para trabajar en él.
ModelDoc::CreateTangentArc	Esta función dibuja un arco tangente a una línea.
	Esta función se utiliza para seleccionar cualquier línea, cara, ejes, etc.
ModelDoc::HoleWizard	Esta función invoca el asistente para crear hoyos en una pieza.
ModelDoc::InsertSketch	Esta función crea un croquis para poder comenzar a dibujar.
ModelDoc::FeatureRevolve2	Esta función crea un sólido de revolución.

La mayoría de estas funciones descritas deben contar con cierta información (datos) requerida para poder ser invocadas y así realizar su tarea. Estos datos pueden ser de diferente tipo (enteros, booleanos, reales, cadenas, etc.) de acuerdo a la definición de cada función, por ejemplo, si se desea dibujar una línea la función que debe llamarse es la siguiente:

ModelDoc.CreateLine2 (xStart, yStart, zStart, xEnd, yEnd, zEnd)
en donde:

xStart	es el valor X del punto inicial de la línea. Tipo: double
yStart	es el valor Y del punto inicial de la línea. Tipo: double
zStart	es el valor Z del punto inicial de la línea. Tipo: double
xEnd	es el valor X del punto final de la línea. Tipo: double
yEnd	es el valor Y del punto final de la línea. Tipo: double
zEnd	es el valor Z del punto final de la línea. Tipo: double

Otro ejemplo es el de la función que guarda un documento con un nombre dado por el usuario:

ModelDoc.SaveAs2 (newName, unused, saveAsCopy, silent)
en donde:

newName	es el nuevo nombre del documento.
unused	actualmente no se usa este valor. Se debe poner 0.
SaveAsCopy	toma el valor TRUE para salvar el documento con el nuevo nombre de archivo sin remplazar archivo ya existen, es decir, habrá dos archivos con la misma información pero con nombres diferentes.
silent	toma el valor TRUE si se quiere evitar diálogos de error y advertencia.

ANEXO C

IMPLEMENTACIÓN DE LA INTERFAZ EN VISUAL C++

```
/////////////////////////////////////////////////////////////////////////////
Clase: CCoordinadorProyecto
Descripción: es una clase coordinadora de aplicaciones y tiene la función de servir como intermediario
entre la capa de presentación (interfaz de usuario) y las clases del dominio.
/////////////////////////////////////////////////////////////////////////////

// CCoordinadorProyecto.h : interface for the CCoordinadorProyecto class.
class CCoordinadorProyecto
{
public:
        CCoordinadorProyecto();
        virtual ~CCoordinadorProyecto();
        CString VersionEje;
        int Cilindro0;
        int Cilindro1;
// Operaciones
public:
        bool AbrirBD(const char * cpszPath_Proyecto,const char * cpszNombre_Proyecto);
        bool SeleccionarComponente(const char * Comp_Activo);
        bool ConsultarComponentes(CListBox * CompenLista);
        bool ConsultarVersiones(CListBox * VersenLista);
        bool SeleccionarVersion(const char * Ver_Activa);
        bool ConsultarCaracteristicas(const char * Comp_Activo,const char * Ver_Activa,CListBox
        *Caract,CListBox *Tipo);
        bool VerificarDimensiones(const char * Comp_Activo,const char * Ver_Activa);
        bool DibujarModelo(const char * Comp_Activo,const char * Ver_Activa);
        bool GuardarModelo(CString nombreArchivo);
};
```

A continuación se presenta solamente el código de la función DibujarModelo ya que el código completo es extenso.

```
/*****************************************************************************
function:       DibujarModelo
input:          Componente Activo y Versión Activa
La función dibuja un componente de acuerdo a la información tomada del MP.
*****************************************************************************/
bool CCoordinadorProyecto::DibujarModelo(const char * Comp_Activo,const char * Ver_Activa)
{
        dModelDoc = CMETApp::GetSldWorks()->NewPart ();
        ModelDoc.AttachDispatch(dModelDoc); //Obtiene elmiembro ModeloDoc
        ModelDoc.InsertSketch(); //Inserta un Sketch (Croquis)
        CCoordinadorProyecto c_Coordinador;
        //Determinar el nombre de la Version
        csComponente = Comp_Activo;
        csVersion = Ver_Activa;
        iLonComponente = csComponente.GetLength();
        iLonVersion = csVersion.GetLength();
        iDesde = iLonVersion-iLonComponente;
        VersionEje = csVersion.Right(iDesde);

        iNum_CPrimarias=c_Coordinador.el_componente->Get_Num_CaractPrimarias(Comp_Activo);
        for(int y=0; y<iNum_CPrimarias; y++)
       {
csCPrimaria=c_Coordinador.el_componente->Get_Nombre_CaractPrimarias(Comp_Activo,y);
csTipoCPrimaria = c_Coordinador.el_componente->Get_TipoCPrimaria(Comp_Activo,y);
//CCilindro
if(csTipoCPrimaria == "CCilindro")
{
        CString namecil = csCPrimaria + VersionEje;
        L = (c_Coordinador.la_df_cil_ext->Get_L(Comp_Activo, Ver_Activa, namecil))/1000;
        XCil = (c_Coordinador.la_df_cil_ext->Get_Punto_X(Comp_Activo, Ver_Activa, namecil))/1000;
        YCil = (c_Coordinador.la_df_cil_ext->Get_Punto_Y(Comp_Activo, Ver_Activa, namecil))/1000;
        DCil = (c_Coordinador.la_df_cil_ext->Get_D(Comp_Activo, Ver_Activa, namecil))/1000;
        TCil = TCil + L;
```

```
        iciclo = 0;
        Izquierda = false;
        Derecha = false;
}//CCilindro
      iNum_CSecundarias=c_Coordinador.la_cprimaria->Get_Num_CaractSecundarias(csCPrimaria);
      iContador = iContador + 1;
            if (iNum_CSecundarias == 0)
            {
                  ModelDoc.CreateLine2 (LineX1,LineY1,0,XCil,DCil/2,0);
                  ModelDoc.CreateLine2 (XCil,DCil/2,0,XCil+L,DCil/2,0);
                  LineX1 = XCil+L;
                  LineY1 = DCil/2;
                  Derecha = true;
            }

  /**********Inicio Ciclo para dibujar CSecundarias **********/
        label1:
        for(int x=0; x<iNum_CSecundarias; x++)
        {
      csCSecundaria = c_Coordinador.la_cprimaria->Get_Nombre_CaractSecundarias(csCPrimaria,x);
      csTipoCSecundaria = c_Coordinador.la_cprimaria->Get_TipoCSecundaria(csCPrimaria,x);

//Dibujar Característica Secundaria: "Centro"
if(csTipoCSecundaria == "CCentro")
{
  if(iciclo == 1)
  {
    //Recuperar inf. de la Característica Sedundaria Centro
   CString nameCentro = csCSecundaria + VersionEje;
   DCentro = (c_Coordinador.la_df_centro->Get_d(Comp_Activo, Ver_Activa, nameCentro))/1000;
   LCentro = (c_Coordinador.la_df_centro->Get_L(Comp_Activo, Ver_Activa, nameCentro))/1000;
   RCentro = (c_Coordinador.la_df_centro->Get_r(Comp_Activo, Ver_Activa, nameCentro))/1000;

   if (y == 0)
  Cilindro1 = 1;
```

```
    if (y == iNum_CPrimarias - 1)
    CilindroN = 1;
  }
}//if Característica Secundaria: "Centro"

//Dibujar Característica Secundaria: "Ranura"
  if(csTipoCSecundaria == "CRanura")
  {
    if (iciclo == 1)
    {
        //Recuperar inf. de la Característica Sedundaria Ranura
        CString namer = csCSecundaria + VersionEje;
        Lran = (c_Coordinador.la_df_ranura->Get_L(Comp_Activo, Ver_Activa, namer))/1000;
        float Rran = (c_Coordinador.la_df_ranura->Get_r(Comp_Activo, Ver_Activa, namer))/1000;
        float Dran = (c_Coordinador.la_df_ranura->Get_de(Comp_Activo, Ver_Activa, namer))/1000;
        Xran = (c_Coordinador.la_df_ranura->Get_Punto_X(Comp_Activo, Ver_Activa, namer))/1000;
        float Iran = c_Coordinador.la_df_ranura->Get_di(Comp_Activo, Ver_Activa, namer);

        float Y1 = Dran/2;
        float Y2 = (Dran/2) - Rran;
        float X1 = Xran + Lran;

        //Dibujar ranura externa cuadrada
        ModelDoc.CreateLine2 ( LineX1,LineY1,0,Xran,Y1,0 );//crea línea
        ModelDoc.CreateLine2 (Xran,Y1,0,Xran,Y2,0);
        ModelDoc.CreateLine2 (Xran,Y2,0,X1,Y2,0);
        ModelDoc.CreateLine2 (X1,Y2,0,X1,Y1,0);
        LineX1 = X1;
        LineY1 = Y1;
        Ran = true;
  }
}

//Dibujar Característica Secundaria: "Radio Convexo Externo"
if(csTipoCSecundaria == "Conv_Ext")
{
```

```
    CString nameConv_Ext = csCSecundaria + VersionEje;
    float Dconv_ext=(c_Coordinador.la_df_radio_convexo->Get_d(Comp_Activo,Ver_Activa,
    nameConv_Ext)) /1000;
    float Rconv_ext=(c_Coordinador.la_df_radio_convexo->Get_r(Comp_Activo,Ver_Activa,
   nameConv_Ext))/1000;
   float Xconv_ext = (c_Coordinador.la_df_radio_convexo->Get_Punto_X(Comp_Activo, Ver_Activa,
   nameConv_Ext))/1000;
   int Iconv_ext = c_Coordinador.la_df_radio_convexo->Get_Vector_I(Comp_Activo, Ver_Activa,
   nameConv_Ext);

   float YConv1 = (Dconv_ext / 2) - Rconv_ext;
   float YConv2 = Dconv_ext / 2;

   if(iciclo == 0)
   {
       if(Iconv_ext == -1)
       {
           float XConv2 = Xconv_ext + Rconv_ext;
           ModelDoc.CreateLine2(LineX1,LineY1,0,LineX1,YConv1,0);
           ModelDoc.CreateTangentArc(Xconv_ext,YConv1,0,XConv2,YConv2,0);
           LineX1 = XConv2;
           LineY1 = YConv2;
           Izquierda = true;
       }
}
   if(iciclo == 1)
   {
       if (Iconv_ext == 1)
       {
               Derecha = true;
               float XConv2 = Xconv_ext - Rconv_ext;
               ModelDoc.CreateLine2 (LineX1,LineY1,0,XConv2,YConv2,0);
               ModelDoc.CreateArc2 (XConv2,YConv1,0,XConv2,YConv2,0,Xconv_ext,YConv1,0,+1);
              LineX1 = Xconv_ext;
              LineY1 = YConv1;
       }
```

```
  }
}//fin Dibujar Característica Secundaria: "Radio Convexo Externo"

//Dibujar Característica Secundaria: "Chaflán de Transición Externo"
  if(csTipoCSecundaria == "Chaf_Trans")
  {
        CString nameChaf_Trans = csCSecundaria + VersionEje;
        float Dchaf_trans = (c_Coordinador.la_df_chaflan_trans->Get_d(Comp_Activo, Ver_Activa,
        nameChaf_Trans))/1000;
        float Lchaf_trans = (c_Coordinador.la_df_chaflan_trans->Get_l(Comp_Activo, Ver_Activa,
        nameChaf_Trans))/1000;
        flota Xchaf_trans=(c_Coordinador.la_df_chaflan_trans->Get_Punto_X(Comp_Activo,
         Ver_Activa, nameChaf_Trans))/1000;
        float ALFAtrans = c_Coordinador.la_df_chaflan_trans->Get_alfa(Comp_Activo, Ver_Activa,
         nameChaf_Trans);
        float Ichaf_trans = c_Coordinador.la_df_chaflan_trans->Get_Vector_I(Comp_Activo,
        Ver_Activa, nameChaf_Trans);
       //Convierte el ángulo a radianes
       float GAMA = (ALFAtrans * pi) / 180;
       float g = Lchaf_trans / cos(GAMA);
       float h = g * sin(GAMA);
       float Y1 = Dchaf_trans / 2;
       float Y2 = (Dchaf_trans / 2) + h;

       if(iciclo == 0)
       {
          if (Ichaf_trans == -1)
          {
             float X2 = Xchaf_trans + Lchaf_trans;
             ModelDoc.CreateLine2(LineX1,LineY1,0,Xchaf_trans,Y2,0);
             ModelDoc.CreateLine2 (Xchaf_trans,Y2,0,X2,Y1,0);
             LineX1 = X2;
             LineY1 = Y1;
             Izquierda = true;
          }
       }
```

```
        if(iciclo == 1)
        {
                if (lchaf_trans == 1)
                {
                   Derecha = true;
                   float XX2 = Xchaf_trans - Lchaf_trans;
                   ModelDoc.CreateLine2 (LineX1,LineY1,0,XX2,Y1,0);
                   ModelDoc.CreateLine2 (XX2,Y1,0,Xchaf_trans,Y2,0);
                   LineX1 = Xchaf_trans;
                  LineY1 = Y2;
                }
        }
}//fin Dibujar Caracteristica Secundaria: "Chaflán de Transición Externo"

//Dibujar Característica Secundaria: "Undercut"
if(csTipoCSecundaria == "Undercut")
{
  CString nameUnder = csCSecundaria + VersionEje;
  float Dunder= (c_Coordinador.la_df_undercut->Get_d(Comp_Activo, Ver_Activa, nameUnder))/1000;
  float Runder = (c_Coordinador.la_df_undercut->Get_r(Comp_Activo, Ver_Activa, nameUnder))/1000;
  float Xunder=(c_Coordinador.la_df_undercut->Get_Punto_X(Comp_Activo,Ver_Activa,
  nameUnder))/1000;
  int lunder = c_Coordinador.la_df_undercut->Get_Vector_I(Comp_Activo, Ver_Activa, nameUnder);
  float Pyi = Dunder / 2;
  float Pyf = (Dunder / 2) + Runder;
  if (iciclo == 0)
  {
        if (lunder == -1)
        {
           float Pxi = Xunder + Runder;
           float PX = Xunder + (2 * Runder);
           ModelDoc.CreateLine2 (LineX1,LineY1,0,Xunder,Pyi,0);
           ModelDoc.CreateArc2 (Pxi,Pyi,0,Xunder,Pyi,0,PX,Pyi,0,+1);
           LineX1 = PX;
           LineY1 = Pyi;
```

```
        Izquierda = true;
        }
}
if (iciclo == 1)
{
   if (Iunder == 1)
   {
      Derecha = true;
      float Pxi = Xunder - (2 * Runder);
      float PY = Pxi+Runder;
      ModelDoc.CreateLine2 (LineX1,LineY1,0,Pxi,Pyi,0);
      ModelDoc.CreateArc2 (PY ,Pyi,0, Pxi,Pyi,0,Xunder,Pyi,0,1);
      LineX1 = Xunder;
      LineY1 = Pyi;
  }
 }
}// fin Dibujar Característica Secundaria: "Undercut"

//Dibujar Característica Secundaria: "Radio Cóncavo Extemo"
if(csTipoCSecundaria == "Conc_Ext")
{
  CString nameConc = csCSecundaria + VersionEje;
  Flota Dconc = (c_Coordinador.la_df_radio_concavo->Get_d(Comp_Activo,Ver_Activa,
  nameConc))/1000;
  float RConc = (c_Coordinador.la_df_radio_concavo->Get_r(Comp_Activo,Ver_Activa,
  nameConc))/1000;
  float XConc = (c_Coordinador.la_df_radio_concavo->Get_Punto_X(Comp_Activo, Ver_Activa,
  nameConc))/1000;
  int   IConc = c_Coordinador.la_df_radio_concavo->Get_Vector_I(Comp_Activo,Ver_Activa,
  nameConc);
  if (iciclo == 0)
  {
    if (IConc == -1)
    {
       float PX1 = XConc + RConc;
       float PY1 = DConc / 2;
```

```
        float PY2 = (DConc / 2) + RConc;

        ModelDoc.CreateLine2 (LineX1,LineY1,0,XConc,PY2,0);
        ModelDoc.CreateArc2(PX1,PY2,0,XConc,PY2,0,PX1,PY1,0,+1);
        LineX1 = PX1;
        LineY1 = PY1;
        Izquierda = true;
      }
}
if (iciclo == 1)
{
  if (IConc == 1)
  {
    Derecha = true;
    float PX1 = XConc - RConc;
    float PY1 = DConc / 2;
   float PY2 = (DConc / 2) + RConc;
   ModelDoc.CreateLine2 (LineX1,LineY1,0,PX1,PY1,0);
   ModelDoc.CreateArc2 (PX1,PY1+RConc,0,PX1,PY1,0,XConc,PY2,0,+1);
   LineX1 = XConc;
   LineY1 = PY2;
  }
 }
}       //Fin Dibujar Característica Secundaria: "Radio Cóncavo Externo"

if(csTipoCSecundaria == "Chaf_Term")
{
  CString nameChaf = csCSecundaria + VersionEje;
  float Dchaf = (c_Coordinador.la_df_chaflan_terminal->Get_d(Comp_Activo, Ver_Activa,
  nameChaf))/1000;
  float Lchaf = (c_Coordinador.la_df_chaflan_terminal->Get_l(Comp_Activo,Ver_Activa,
  nameChaf))/1000;
  float Xchaf = (c_Coordinador.la_df_chaflan_terminal->Get_Punto_X(Comp_Activo, Ver_Activa,
  nameChaf))/1000;
  float ALFA = c_Coordinador.la_df_chaflan_terminal->Get_alfa(Comp_Activo, Ver_Activa,
  nameChaf);
```

```
  int Ichaf = c_Coordinador.la_df_chaflan_terminal->Get_Vector_I(Comp_Activo, Ver_Activa,
  nameChaf);
 //Convierte el ángulo a radianes
 float BETA = (ALFA * pi) / 180;
 float c = Lchaf / cos(BETA);
 float b = c * sin(BETA);
 float P2Y = (Dchaf / 2) - b;
 float P1Y = Dchaf / 2;
 float P1X = Xchaf + Lchaf;
 if(iciclo == 0)
{
  if (Ichaf == -1)
  {
    ModelDoc.CreateLine2 (LineX1,LineY1,0,Xchaf,P2Y,0);
    ModelDoc.CreateLine2 (Xchaf,P2Y,0,P1X,P1Y,0);
    LineX1 = P1X;
    LineY1 = P1Y;
    Izquierda = true;
 }
}

    if (iciclo == 1)
   {
        if (Ichaf == 1)
        {
          Derecha = true;
           float P2X = Xchaf - Lchaf;
           ModelDoc.CreateLine2 (LineX1,LineY1,0,P2X,P1Y,0);
           ModelDoc.CreateLine2 (P2X,P1Y,0,Xchaf,P2Y,0);
           LineX1 = Xchaf;
           LineY1 = P2Y;
        }
    }
}
if (iciclo == 0)
{
```

```
   if (Izquierda == false)
  {
        ModelDoc.CreateLine2 (LineX1,LineY1,0,LineX1,DCil/2,0);
        LineX1 = LineX1;
        LineY1 = DCil/2;
  }
}
iContador = iContador + 1;
}//fin for CSecundrias
if (iciclo == 0 )
{
        iciclo = 1;
        goto label1;
}
if (Derecha == false)
{
   ModelDoc.CreateLine2 (LineX1,LineY1,0,XCil+L,DCil/2,0);
   LineX1 = XCil+L;
   LineY1 = DCil/2;
 }

}//fin for CPrimarias
ModelDoc.CreateLine2 (LineX1,LineY1,0,TCil,0,0);
double point2[3];
point2[0] = TCil ;
point2[1] = 0;
point2[2] = 0;
ModelDoc.CreateCenterLine(vArray, vArray2);
double a= 2*pi;
double b= pi/2;
double c= 3*b;
long r = ModelDoc.FeatureRevolve2 ( a, TRUE, 360, 0,swAutoCloseSketch);
if (r == 0)
{
   if (Cilindro1 == 1)
   {
```

```
        ModelDoc.SelectByID ("", "FACE", 0,0,0);
        ModelDoc.HoleWizard(LCentro,swEndCondBlind,FALSE,FALSE,8,RCentro,LCentro,DCentro,
         1.431169986635, 2.059488517353,-1,-1,-1,-1,-1,-1,-1);
   }
   if (Cilindro1 == 2)
  {
        ModelDoc.SelectByID ("", "FACE", 0,0,0);
        ModelDoc.HoleWizard(LHueco,swEndCondBlind,FALSE,FALSE,5,DHueco,LHueco,2.0594885
         17353,-1,-1,-1,-1,-1,-1,-1,-1,-1);
  }
if (CilindroN == 1)
{
   ModelDoc.SelectByID ("", "FACE", TCil,0,0);
 ModelDoc.HoleWizard(LCentro,swEndCondBlind,FALSE,FALSE,8,RCentro,LCentro,DCentro,1.43116
9986635, 2.059488517353,-1,-1,-1,-1,-1,-1,-1,-1);
}
if (CilindroN == 2)
{
        ModelDoc.SelectByID ("", "FACE", TCil,0,0);
        ModelDoc.HoleWizard(LHueco,swEndCondBlind,FALSE,FALSE,5,DHueco,LHueco,2.0594885
        17353,-1,-1,-1,-1,-1,-1,-1,-1,-1);
  }

 }
 return(TRUE);
}
```

REFERENCIAS BIBLIOGRÁFICAS

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

Booch, Grady, 1996. *Análisis y Diseño Orientado a Objetos con Aplicaciones.* Addison-Wesley/Díaz de Santos, Segunda Edición.

Deitel, H. M., Deitel, P. J., 1995. *Como programar e C/C++.* Prentice Hall Hispanoamericana, S.A. Segunda Edición.

ObjectStore Design and Development for C++

Tutorial SolidWorks Versión 99.

Krause, F., Kimura, F., Kjelberg, T., Lu, S.C. , 1993. *AutoCad Release 12 Tutorial.*

Borja, V., Mirón, H., 1999. *Especificación de un Modelo de Producto para Ejes de Transmisión.*

Borja, V., 1997. *Redesign Supported by Data Models with Particular Reference to Reverse Engineering,* PhD Thesis, Department of Manufacturing Engineering, Loughborough University.

Borja V., López M., Valeriano G., González L., Santillán S., Bell F. , 1997. *Diseño para manufactura asistido por computadora: el Agente para torneado*, Memorias del II Congreso de la Sociedad Mexicana de Ingenieros Mecánicos, SOMIM.

Ayala, Ruiz, Álvaro, 2001. Modelo de Manufactura de una Celda de Producción. UNAM.

Mirón González Hugo, 2001. Modelo del Producto. UNAM.

Molina, A., 1995. A Manufacturing Model to Support Data-Driven Applications for Design and Manufacture., PhD Thesis, Loughborough, England: Loughborough University of Technology.

Costa, C. A., 2000. Product Range Models in Injection Monld Tool Design. PhD Thesis. Loughborough, England: Loughborough University.

Canciglieri-Jnr., O, 1999. Product Model Based Translation Mechanism to Support Multiple Viewpoints in the Design for Manufacture of Injection Monlded Products. PhD Thesis. Loughborough, England: Loughborough University.

Dorador, J. M., 2001. Product and Process Information Interactions in Assembly Decision Support Systems. PhD Thesis. Loughborough, England: Loughborough University.

Pappas, C., Murria W., 1999. Visual C++ 6.0 *Manual de Referencia*. McGraw-Hill.

Dahl, O., Dijkstra, E., and Hoare, 1972. *Structured Programming*. London, England.

ISO, 1994. 10303 Industrial Automation Systems-Product data representation and exchange.