UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA
DIVISIÓN DE ESTUDIOS DE POSGRADO

T E S I S

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA MODELADOR DE EJES DE TRANSMISIÓN

PRESENTADA POR:
HUGO G. MIRÓN GONZÁLEZ, ING.

PARA OBTENER EL GRADO DE:
MAESTRO EN INGENIERÍA MECÁNICA
(ÁREA DISEÑO)

DIRIGIDA POR.
DR VICENTE BORJA RAMÍREZ

CIUDAD UNIVERSITARIA NOVIEMBRE, 2001

RESUMEN

La implantación de la ingeniería concurrente, requiere de nuevos sistemas CAE (*Computer Aided Engineering*). A este respecto, se exploran aplicaciones de modelos (representaciones computacionales) de productos y capacidades de manufactura. En la UNAM, en colaboración con el CONACYT y dos empresas del área metalmecánica, se desarrolla un sistema CAE basado en estos modelos: SADET (Sistema Auxiliar para el Diseño de Ejes de Transmisión). El modelo del producto (MP) de SADET contiene la información necesaria para asistir el rediseño y la planeación de la producción de piezas rotacionales maquinadas. El modelador de ejes de transmisión (MET) es capaz de almacenar, recuperar y editar información de la base de datos, es decir la interfaz de usuario para poblar el MP.

El objetivo de la presente tesis es especificar la estructura básica del MP (Modelo de Producto) y desarrollar la aplicación MET (Modelador de Ejes de Transmisión), los cuales son dos de las partes principales del sistema SADET. En primer lugar se resumen algunos conceptos sobre el modelado computacional de productos, y se plantean los objetivos y alcances para el desarrollo de la investigación. Posteriormente se especifica la información contenida en el MP y se presentan las estructuras de datos principales que componen al modelo. El MP propuesto, se basa en investigación madura en el área, es resultado de una revisión y simplificación del modelo propuesto por (Borja, 1997), para el modelado visual se utiliza el Lenguaje Unificado para Modelado o UML, ya que el MP es orientado a objeto. Para su implementación se utiliza una base de datos orientada a objeto (*ObjectStore*), y *Visual* C++ en un ambiente *Windows* 98.

Una vez definido el MP, se describe la metodología para desarrollar el MET y se lleva a cabo su implementación. Por último se ilustra la aplicación del MP y el MET para representar la estructura física de un eje de transmisión. El caso de estudio verifica la utilidad del modelo propuesto y la herramienta realizada, representando un componente real y utilizando *ObjectStore Inspector* para verificar la integridad de la información en la base de datos

AGRADECIMIENTOS

A mis padres por su infinito amor, ejemplo y apoyo.

Gracias chiquita por tu incondicional apoyo y ayuda para por fin terminar mi Tesis, pero principalmente por tu gran corazón.

A mi hermana y mi familia por su cariño y por estar siempre conmigo.

A mi director de Tesis, por su guía y por invertir su tiempo en la enseñanza.

TABLA DE CONTENIDO

RESUMEN

AGRADECIMIENTOS

1. INTRODUCCIÓN 1

2. ANTECEDENTES

2.1 Introducción 3

2.2 Ingeniería Concurrente 3

2.2.1 Requerimientos de la IC 5

2.2.2 Soporte requerido para la IC 6

2.3 Modelado del producto 7

2.3.1 Tipos de modelos del producto 9

2.4 Diseño basado en modelos de información 9

2.5 Trabajo previo 10

3. OBJETIVOS Y ALCANCES DE LA INVESTIGACIÓN

3.1 Introducción 12

3.2 Metas y objetivos 12

3.3 Alcances de la investigación 12

4. REVISIÓN Y MODELADO EN UML DEL MP

4.1 Introducción 14

4.2 Requerimientos del MP y su metodología de diseño 14

4.3 Estructura principal del MP 16

4.4 Modelado de la categoría de información "Estructura del Producto" 17

4.4.1 La taxonomía característica 18

4.5 ¿Cómo definir entidades de diseño 19

4.6 Modelado de la categoría de información "Características de Diseño" 20

4.6.1 Taxonomía de descripción física para representar partes rotacionales maquinadas 22

4.6.2 Representación de las clases de la taxonomía de descripción física 22

5. DEFINICIÓN Y ALCANCE DEL SISTEMA A DESARROLLAR

5.1 Introducción 25

5 2 Metodología para el desarrollo 25

5.3 Modelador de ejes de transmisión 26

5.3.1 Elementos de la implementación experimental 28

5.4 Recursos necesarios 28

6. IMPLEMENTACIÓN DE LA APLICACIÓN PROTOTIPO

6 1 Introducción 30

6.2 Implementación del MP 30

6 3 Implementación del Modelador de Ejes de Transmisión 32

6.3.1 Modelo conceptual del MET 32

6.3 2 Comportamiento del MET 34

6.3.3 Desarrollo de la aplicación MET 36

7. CASO DE ESTUDIO

7.1 Introducción 47

7.2 Objetivos del caso de estudio 47

7 3 El componente para el caso de estudio 47

7.4 Uso del MET a través del caso de estudio 49

8. CONCLUSIONES 55

REFERENCIAS 57

APÉNDICE A EL MP PROPUESTO POR BORJA 59

APÉNDICE B LENGUAJE UNIFICADO PARA MODELADO 70

APÉNDICE C MODELO IDEF0 DE INGENIERÍA INVERSA 76

APÉNDICE D DETALLES DE CLASES DEL MP 86

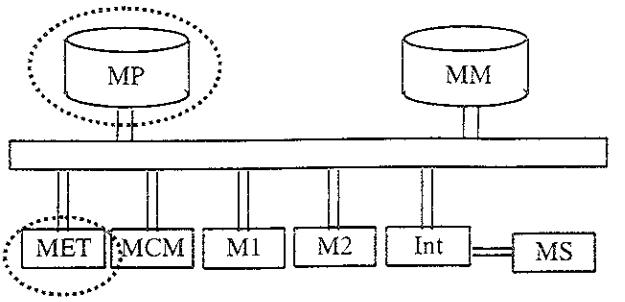
CAPÍTULO 1

INTRODUCCIÓN

Debido a la constante competencia de las empresas a nivel mundial, éstas se han visto en la necesidad de buscar nuevos métodos para mejorar la calidad de sus productos, así como disminuir los tiempos y costos necesarios para su producción. La Ingeniería Concurrente (IC), es un método que nos permite lograr una ventaja competitiva a través del desarrollo de productos de alta calidad en menor tiempo. Ésta propone considerar todos los aspectos del ciclo de vida del producto tan pronto como sea posible en el proceso de su diseño.

La IC requiere de un ambiente eficiente para controlar la información, de tal forma que es necesario integrar las diferentes herramientas de cómputo que se usan durante la realización de productos para lograr así un soporte integral que asista eficazmente a equipos de diseño. Para lograr este soporte, se exploran los sistemas CAE (*Computer Aided Engineering*) basados en modelos de información, es decir en representaciones computacionales en forma de información y conocimiento (detalles adicionales de cómo se puede usar o aplicar la información) dentro de una base de datos.

El trabajo reportado en esta tesis es parte de una investigación realizada en la Facultad de Ingeniería de la UNAM (proyecto PAPIIT IN110398). El objetivo de dicha investigación es explorar la aplicación de modelos de información para asistir el rediseño de componentes. La parte fundamental del proyecto es el desarrollo del Sistema Auxiliar para el Diseño de Ejes de Transmisión (SADET). En la figura 1.1 se muestra un esquema completo de dicho sistema en donde se remarcan los elementos desarrollados en esta tesis. Este sistema CAE soporta el diseño de ejes desde diferentes perspectivas y el diseño del proceso de manufactura simultáneamente. SADET está integrado por diversos programas que interactúan con los mismos modelos de información, cada programa realiza en forma independiente una tarea específica del diseño, pero a través de un ambiente de integración, pueden pedir y proporcionar información a los otros programas del sistema y modelos de información.



MP Modelo de producto
MET Modelador de ejes de transmisión
M1 Módulo 1 de diseño para manufactura
Int Interface a modelador de sólidos
MM Modelo de manufactura
MCM Modelador de celdas de manufactura
M2 Módulo 2 de diseño para manufactura
MS Modelador de sólidos

Figura 1.1 *Arquitectura del sistema SADET*

El objetivo de la presente tesis es especificar la estructura básica del MP (Modelo de Producto) y desarrollar la aplicación MET (Modelador de Ejes de Transmisión), los cuales son dos de las partes principales del sistema SADET.

En el Capítulo 2 se hace una revisión sobre conceptos relativos al modelado de productos, posteriormente en el Capítulo 3 se plantean los objetivos y alcances de la investigación. A continuación en el Capítulo 4 se describe el MP propuesto y su modelo en UML (*Unified Modeling Language*), comenzando por definir su marco de referencia. En los Capítulos 5 y 6 se define y se implementa la aplicación prototipo iniciando con el MP y posteriormente el Modelador de Ejes de Transmisión. Por último se ejemplifica la representación de un producto utilizando el MP (Capítulo 7) y se presentan las conclusiones así como una breve descripción del trabajo a realizar en el futuro sobre el proyecto (Capítulo 8).

CAPÍTULO 2

ANTECEDENTES

2.1 Introducción

En este capítulo se reporta una breve revisión bibliográfica sobre los principales temas relacionados con esta tesis. En la Sección 2.2 se define Ingeniería Concurrente, así como sus requerimientos y el soporte requerido para llevarla a cabo. Las Secciónes 2.3 y 2.4 presentan conceptos acerca de modelos de información del producto y sus diferentes tipos.

Por último, en la Sección 2.5, se describe el trabajo previo sobre el que se basa la presente tesis.

2.2 Ingeniería Concurrente

La Ingeniería Concurrente (IC), es un método que nos permite lograr una ventaja competitiva a través del desarrollo de productos con alta calidad en menos tiempo. Existen varias definiciones de IC, a continuación se presentan algunas de las principales:

- Aparicio, 1996. Es una filosofía en la cuál todas las actividades del ciclo de vida del producto son consideradas concurrentemente en la fase de diseño. Esto se lleva acabo por un equipo multidisciplinario de expertos en las diversas etapas del ciclo de vida del producto, y tiene como objetivo el identificar y prevenir problemas en cada una de ellas. Como resultado se obtiene un incremento en la calidad, así como la disminución en el tiempo de desarrollo y el costo del producto.

- Ellis, 1992. Es un ambiente de diseño en ingeniería en el cual la tecnología de diseño asistido por computadora es usada para evaluar e incrementar la calidad del producto, no solo durante la actividad de diseño, sino a través de todo su ciclo de vida.

- Cals, 1991. Es un método sistemático para crear el diseño de un producto, considera todos los elementos del ciclo de vida de dicho producto, desde su concepción hasta su distribución. La IC define simultáneamente el producto, su proceso de manufactura, y todos los demás procesos requeridos del ciclo de vida, tales como soporte logístico, etc.

- Winner, 1998. Es un método sistemático para la integración del diseño concurrente de productos y sus procesos relacionados, incluyendo manufactura y soporte. Este método está destinado a causar que los desarrolladores consideren todos los elementos del ciclo de vida del producto desde su concepción hasta su venta, incluyendo calidad, costo, planeación y requerimientos de usuarios.

Como se observa, existen diversas definiciones, se puede resaltar que todas ellas coinciden en que la IC corresponde al proceso de diseño, y promueve actividades en paralelo o concurrentes en el desarrollo de productos. En la figura 2.1, se compara un proceso secuencial tradicional y el enfoque de la Ingeniería Concurrente.

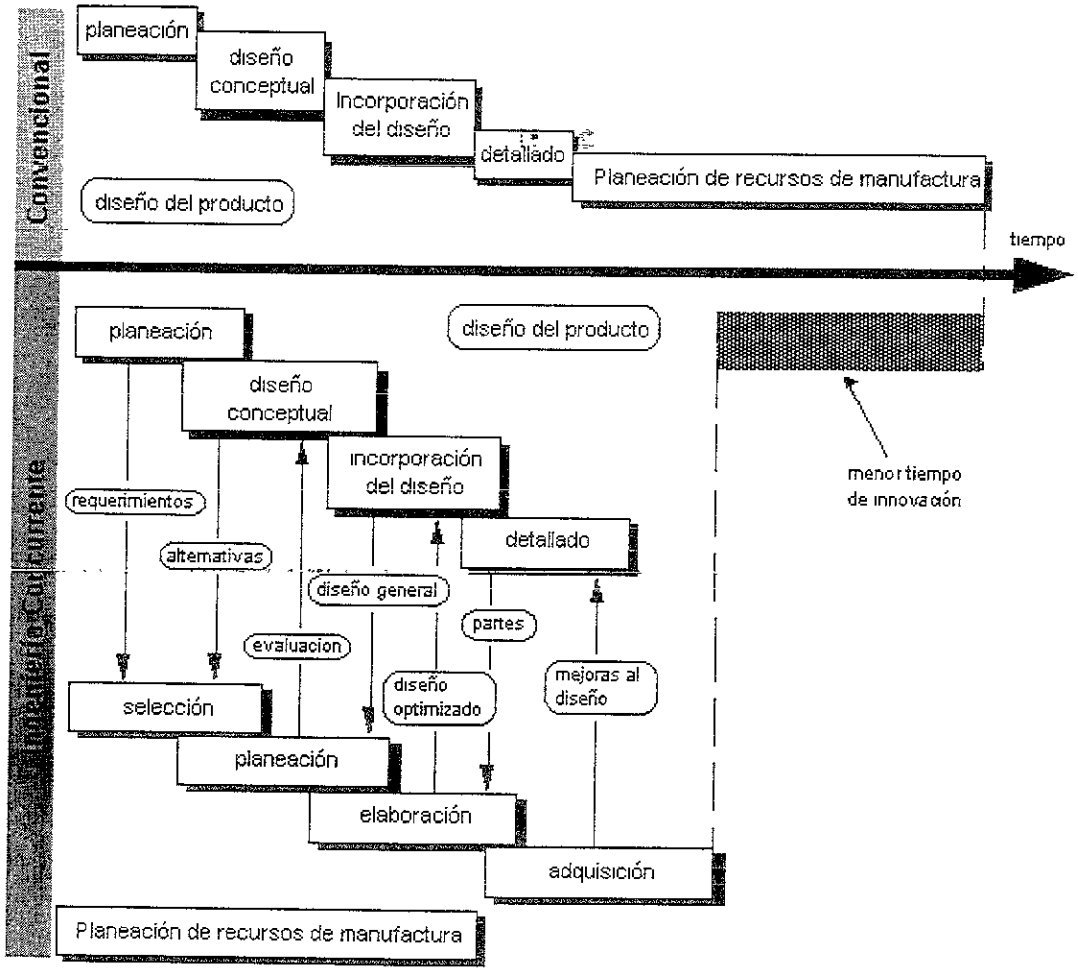


Figura 2.1 *Ingeniería Concurrente*
(Fuente: Weck,1991. Production Engineering the Competitive Edge)

El enfoque o la propuesta de la IC es el de considerar todos los aspectos del ciclo de vida del producto en forma simultánea. Al utilizar la IC se incrementa la calidad de los productos y se mejoran sus procesos debido a que los diseños cubren todos los requerimientos del cliente.

Por otro lado, se pueden lograr importantes reducciones del costo evitando cambios en etapas posteriores al diseño y simplificando la producción. También se reduce el tiempo en el proceso de diseño, desarrollo y comercialización del producto, debido a las actividades paralelas y a la integración del diseño del producto y su proceso de manufactura.

2.2.1 Requerimientos de la IC

La IC se debe caracterizar por la aplicación de principios que introducen cambios culturales, humanos y de organización dentro de las empresas, a través del uso de metodologías formales, en algunos casos, soportados por la tecnología de información (Molina et al., 1995). Dichos principios pueden ser clasificados como:

- Principios de organismo: nuevas estructuras de una organización, trabajo en equipo, disciplina, enfoque a la actitud de los clientes.
- Principios de mejora del proceso: desarrollo y mejora continua de las actividades del ciclo de vida del producto, de tal manera que sean integradas y se dé la concurrencia siempre que sea posible.
- Principios de gestión de información: altos niveles de información e integración, así como gestión de recursos corporativos (tecnología de información, etc.).

El trabajo en equipo es uno de los elementos principales en la estructura organisacional de la IC, esto hace posible la concurrencia de actividades. Los equipos se conforman por especialistas de campos complementarios, clientes y proveedores. La comunicación y organización eficientes dentro del equipo son sumamente importantes para tomar decisiones.

Para mejorar las prácticas de diseño, la IC hace uso de métodos formales (QFD, DFM, DFA, etc.) y de herramientas (computadoras/redes, CAD, CAE, simulación) enfocadas a las diferentes etapas del ciclo de vida del producto. La IC requiere de un ambiente eficiente para controlar la información, ya que siempre debe estar disponible para todos los miembros del equipo. El uso de

la tecnología de información asiste en el control de la información involucrada en el proceso de diseño.

2.2.2 Soporte requerido por la IC

Las investigaciones actuales exploran los sistemas CAE basados en los modelos de información, dichos sistemas capturan diversos elementos del diseño de productos posibilitando la IC (Prasad, 1996). Estos sistemas soportan el diseño de productos desde diferentes perspectivas y el diseño del proceso de manufactura simultáneamente. Los sistemas CAE son integrados por diversos programas conectados entre sí que interactúan con los mismos modelos de información. Cada programa es lo bastante independiente para realizar una tarea específica del diseño, pero a través de un ambiente de integración, pueden pedir y proporcionar información a los otros programas y modelos de información.

El soporte CAE requerido para la IC se puede dividir en dos tipos: independiente e integrado. Los sistemas de soporte independiente son la primera generación de las aplicaciones de IC. Están propuestos para productos específicos, pero no soportan el trabajo en equipo. Estas aplicaciones asisten el proceso de diseño del producto concurrentemente considerando diferentes aspectos de su ciclo de vida. Las estructuras de soporte integrado pueden ser consideradas como aplicaciones de segunda generación. Estas promueven el trabajo en equipo usando modelos de información y aplicaciones integradas de ingeniería.

Molina et al. (1995) identifica cuatro importantes requerimientos tecnológicos para el soporte computacional de la Ingeniería Concurrente:

a) Metodologías de modelado. Metodologías para modelar el ciclo de vida del producto, procesos de diseño y manufactura, comportamiento de la empresa, organización, comportamiento humano.

b) Soporte a decisiones asistido por computadora. Aplicaciones que permitan interactuar a grupos y equipos, colaborar y tomar decisiones durante la realización del producto. La tendencia actual es la de desarrollar aplicaciones que puedan compartir información a través de modelos y otras aplicaciones.

c) Arquitectura de información. Arquitecturas computacionales de información abiertas o distribuidas, para proveer diferentes servicios y llevar a cabo la integración y la comunicación.

d) Estructuras para el desarrollo de un ambiente de IC. Existe una necesidad de estructuras que organicen por sí solas el proceso de desarrollo de sistemas de IC. Tales estructuras de desarrollo sirven como referencia a modelos para establecer la aplicación tiempo-orden de la gente. Métodos y herramientas para definir, desarrollar e implementar el ambiente de IC a un nivel deseado de integración y automatización.

2.3 Modelado del producto

En sus primeras etapas, las computadoras asistieron a los diseñadores resolviendo ecuaciones complejas, posteriormente eran capaces de representar productos usando dibujos en dos y tres dimensiones. Actualmente los avances en las bases de datos facilitan el manejo de grandes cantidades de información, dichos avances han sido integrados a sistemas CAD. De esta forma el modelado del producto se ha desarrollado como consecuencia de cambios en las prácticas de diseño y la tecnología que lo soporta.

En nuestros días, el modelado del producto es fundamental en el desarrollo de sistemas CAE. En concreto, se refiere a las actividades para representar y utilizar información relacionada a productos, a través de todo su ciclo de vida, desde la planeación inicial del producto hasta su venta (Kimura & Suzuki, 1989).

El modelado del producto consiste de dos aspectos interrelacionados: modelos del producto y fases de desarrollo del producto (fases involucradas en la realización de productos, incluyendo planeación del producto, diseño y manufactura). Un *modelo del producto* es una representación de toda la información relevante concerniente a un producto dado durante su ciclo de vida (Krause et al., 1993). Éste llena todos los requerimientos de información para aplicaciones específicas, y para sectores relacionados de ingeniería. En el contexto de sistemas CAE, los modelos del producto se refieren a una base de datos que contiene información y conocimiento (detalles adicionales de cómo se puede usar o aplicar la información) del producto, su controlador y algoritmos de acceso.

El modelo del producto es determinado por su estructura y su contenido específico. La estructura depende de la naturaleza del producto, de las herramientas para modelar la información y de los esquemas necesarios para implementar la base de datos. El contenido es determinado por el producto en particular.

Las etapas de desarrollo del producto representan procesos de modelado del producto. Esos procesos son conjuntos de actividades.

Existen varias propuestas acerca de la información que deben incluir los modelos del producto. La información se determina por los procesos de desarrollo del producto que serán soportados por el modelo. Se pueden identificar tres grupos de información del producto según propuestas de diversos autores:

a) Información relacionada con la descripción del producto, por ejemplo: estructura, información económica, geometría (Bauert, 1989), tecnología, funcionalidad, comportamiento (Kimura & Suzuki, 1989).

b) Información relacionada con el ciclo de vida del producto, por ejemplo: métodos usados por el diseñador (Bauert, 1989); requerimientos, decisiones y justificación para determinar la descripción del producto (Kimura & Suzuki, 1989).

c) Información relacionada con el ambiente del producto, por ejemplo: información de compañías participantes (Bauert, 1989), aspectos inovativos de diseño y manufactura (Mantyla, 1989), comportamiento de los usuarios, estrategias de los competidores (Krause et al., 1993).

Un modelo del producto puede estar estructurado en submodelos, creando lo que se llaman modelos parciales los cuales contienen información relacionada a tareas específicas de las fases de desarrollo del producto. Si están bien organizados, los modelos parciales se pueden compartir por varios productos, aunque el contenido siempre dependerá de cada producto.

2.3.1 Tipos de modelos del producto

Krause et al. (1993) ha identificado varios tipos de modelos del producto, dependiendo de la información que almacenan en sus estructuras. Estos tipos son:

- Modelos del producto orientados a la estructura, se basan en la estructura física del producto.
- Modelos orientados a la geometría, se dedican a capturar la geometría y una representación sólida del producto.
- Modelos orientados a las características, éstos almacenan información en forma de características.
- Modelos del producto basados en el conocimiento, destinados a capturar conocimiento.
- Modelos del producto integrados, estos almacenan información de todos los tipos antes descritos.

El proceso de desarrollo de un producto está soportado por un modelo que posibilita la gestión integrada y la representación neutral de su información y de su conocimiento genérico, dicho conocimiento comprende la historia del producto, principios de desarrollo, modelos de requerimientos tecnológicos y de clientes y modelos de fallas. La representación del conocimiento genérico del producto toma en consideración los diferentes estados de su vida. De esta forma, los modelos del producto proveen un soporte integrado para el desarrollo de productos.

2.4 Diseño basado en modelos de información

La figura 2.2 ilustra el concepto de diseño basado en modelos de información, este promueve el uso combinado de modelos del producto, modelos de manufactura y aplicaciones para asistir a equipos de diseño a realizar actividades particulares que generen, recuperen y modifiquen información contenida en dichos modelos de información, todo esto bajo un mismo ambiente de integración (ej. red computacional).

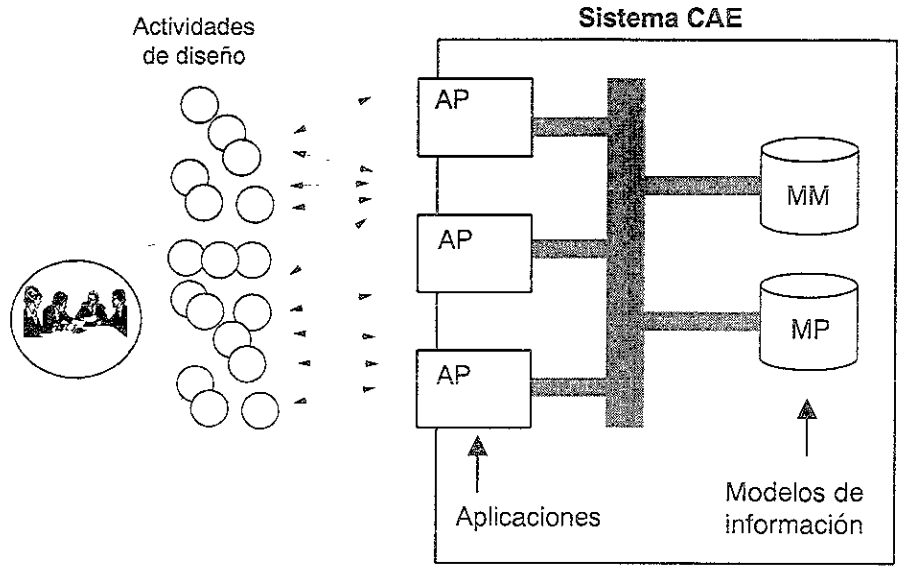


Figura 2.2 *Diseño basado en modelos de información*

2.5 Trabajo previo

La presente tesis se basa en la investigación doctoral de Borja (1997) titulada "*Redesign Supported by Data Models with Particular Reference to Reverse Engineering*" realizada en la Universidad de Loughborough, Reino Unido.

La meta de dicha investigación fue la de explorar la especificación y definición de sistemas CAE basados en modelos de información para posibilitar el rediseño concurrente y, en particular, plantear un enfoque novedoso a la ingeniería inversa. Esto con el fin de identificar futuros lineamientos para la realización y aplicación de esta tecnología en las prácticas de diseño concurrente.

En la investigación de Borja se identificaron tres objetivos principales:

- Establecer un modelo del proceso de ingeniería inversa de productos, en el que se considerará el concepto de diseño basado en el manejo de información.
- Diseñar un sistema CAE que soportara el modelo del proceso.
- Explorar la aplicación del modelo del proceso y del sistema CAE a través de un caso de estudio industrial.

Para alcanzar los objetivos planteados se requirió trabajar en diversas áreas. En principio se realizó una investigación de los métodos contemporáneos de realización de productos, diseño concurrente y procesos de rediseño, técnicas de diseño, ingeniería inversa, etc. También se llevó a cabo la revisión literaria del soporte computacional moderno para ingeniería inversa.

A continuación se propuso un proceso original de ingeniería inversa. Y a su vez se definió el concepto de un sistema CAE manipulador de información para posibilitar dicho proceso de ingeniería inversa. El concepto se fundamentó en el modelado de información, y en particular en el proyecto MOSES (1992-1995).

El siguiente paso fue desarrollar un modelo de información que fuera capaz de capturar la información requerida para soportar el desarrollo de un producto rediseñado, éste modelo se basó en el paradigma orientado a objetos.

Posteriormente se definió un modelo de información de manufactura capaz de representar varias instalaciones de producción y que soportara los aspectos de manufactura de la ingeniería inversa. Esos aspectos comprenden sugerencias para definir las características del substituto, y la selección de procesos y recursos para su manufactura.

Teniendo todo lo anterior se realizó una implementación prototipo del concepto del sistema CAE diseñado anteriormente, con el fin de soportar un caso de estudio en colaboración con la industria. En el caso de estudio definido se investigaron aspectos representativos del proceso de ingeniería inversa y del concepto del sistema usando la implementación experimental del sistema CAE en un problema real. Con esto se llevó a cabo la exploración del modelo del proceso así como del sistema CAE y se alcanzaron las metas y objetivos establecidos en la investigación.

La presente tesis se basa en una parte del trabajo descrito anteriormente, en particular en el desarrollo del modelo de información del producto. En el siguiente capítulo se identifican las metas y objetivos del trabajo reportado en esta tesis y se delimita la investigación requerida para llevarlos a cabo.

CAPÍTULO 3

OBJETIVOS Y ALCANCES DE LA INVESTIGACIÓN

3.1 Introducción

A continuación se identifican las metas y objetivos del trabajo de tesis, así como las actividades realizadas para llevarlos a cabo. De esta forma queda establecido el alcance de dicha investigación.

3.2 Metas y objetivos

La meta de esta investigación es revisar y aplicar la tecnología desarrollada anteriormente, en este caso el MP propuesto por Borja (1997), y utilizarla –con cambios propuestos– para desarrollar un programa o aplicación que asista a los diseñadores en la extracción de información de productos existentes. Tanto el programa como el MP se integrarán a un sistema CAE para asistir rediseño concurrente.

En el presente trabajo se han identificado tres objetivos principales que son:

- Revisar y modelar en UML el modelo de información del producto (MP) propuesto por Borja (1997). Dicho modelo es parte de un sistema CAE para asistir el proceso de ingeniería inversa.

- Implementar el MP modelado en UML empleando una base de datos orientada a objetos.

- Desarrollar una aplicación (Modelador de Ejes de Transmisión) que sea capaz de almacenar, recuperar y editar información de la base de datos y probarla a través de un caso de estudio.

3.3 Alcances de la investigación

Para alcanzar los objetivos propuestos en la sección anterior, se requieren actividades en diversas áreas, como son:

a) Investigación de métodos para el desarrollo de productos, en particular Ingeniería Concurrente, sus requerimientos y el soporte necesario para implementarla (Capítulo 2).

b) Revisión bibliográfica sobre los modelos de información y el modelado del producto (Capítulo 2). Abarcando la investigación de herramientas y lenguajes de modelado, como Booch y posteriormente UML (Apéndice B). Esto con el fin de poder comprender el trabajo realizado anteriormente por Borja y sobre el cuál se basa esta tesis.

c) Revisar el modelo de información del producto (MP) propuesto por Borja (1997), y proponer cambios o simplificaciones con el fin de adecuarlo para capturar la información necesaria para modelar ejes de transmisión (Capítulo 4).

d) Modelar en UML el nuevo MP propuesto (Capítulo 4).

e) Definir el caso de estudio y probar el funcionamiento de la aplicación a través de éste. (Capítulo 7). Plantear las conclusiones del trabajo realizado (Capítulo 8).

f) Definir el alcance del sistema a desarrollar, los recursos necesarios y el software requerido para su implementación. Con base en dichos recursos definir las especificaciones del programa (Capítulo 5).

g) Implementar el MP y el programa Modelador de Ejes de Transmisión (MET) (Capítulo 6).

En los siguientes capítulos se reporta el trabajo descrito anteriormente y que se ha realizado para alcanzar las metas y objetivos establecidos en esta tesis.

CAPÍTULO 4

REVISIÓN Y MODELADO EN UML DEL MP

4.1 Introducción

En el capítulo anterior se plantearon los objetivos y alcances de la investigación, ahora se presenta una modificación al MP propuesto por Borja (1997) para crear un nuevo modelo simplificado que sea capaz de capturar la información necesaria para modelar ejes de transmisión. Esta nueva adaptación del modelo del producto formará parte de un sistema CAE para asistir el proceso de rediseño concurrente.

En la primera parte se definen los requerimientos y el procedimiento de construcción del MP, posteriormente se detalla cada una de las categorías de información definidas y las modificaciones realizadas en cada una de ellas.

4.2 Requerimientos del MP y su metodología de diseño

El objetivo al desarrollar un modelo de información del producto es el de posibilitar la creación de instancias, como por ejemplo modelos del producto (MP´s), los cuales representan productos y sus rediseños. El modelo del producto soporta aplicaciones para asistir el diseño para manufactura o diseño para "X".

Para la creación de la estructura del modelo de información se tomaron como base conceptos genéricos, de tal forma que pudiera ser aplicado a una gran variedad de productos electro-mecánicos. Sin embargo, las especificaciones de las clases se definieron considerando componentes rotacionales (ejes de transmisión) manufacturados por procesos de maquinado, en particular por torneado.

Originalmente el proceso utilizado por Borja para la definición del MP fue:

a) La definición de los requerimientos de información usando un modelo funcional (se utilizó la técnica de modelado IDEF0).

b) La generación de un esquema conceptual (categorías de información) y estructural como un modelo semántico empleando el paradigma orientado a objetos.

c) La documentación detallada de los atributos, su dominio y restricción en un lenguaje de definición de información (ddl) usado para la implementación.

Para definir el nuevo MP simplificado se partió del modelo semántico realizado por Borja (utilizando la técnica de Booch) y se siguieron los siguientes pasos:

a) Aplicación de restricciones necesarias para cumplir solo con los objetivos planteados, (se presentarán a lo largo del desarrollo del MP en las secciones 4.3, 4.4 y 4.6).

b) Generación del modelo semántico empleando ahora UML como herramienta de modelado.

c) Implementación del MP simplificado empleando una base de datos orientada a objetos (Capítulo 6).

Los diagramas de clases en UML se crearon de tal forma que fueran flexibles, simples de aprender y de usar. Además, éstos debían ser eficientes en la comunicación de ideas. Considerando los elementos esenciales del proceso de ingeniería inversa, se seleccionaron los nodos relevantes del modelo IDEF0 propuesto por Borja, que son: Determinar Especificaciones, Análisis de la Información del Producto de Referencia, Rediseñar Entidad, Seleccionar Instalación y Especificar Información de Manufactura. Los diagramas de definición de dichos nodos se presentan en el Apéndice C.

El modelo de información del producto definido por Borja (1997) organiza la información requerida para dicho modelo en seis categorías de información, esto como resultado del análisis de las limitaciones o restricciones de los nodos anteriormente descritos. Estas categorías son:

1. Estructura del producto. Modela la estructura física de un producto, representa la organización de sus elementos y como están asociados entre ellos.
2. Especificaciones. Describe el conjunto de características que se desean de un producto.
3. Conceptos. Incluye la información que describe la funcionalidad de los productos y su estructura funcional.

4. Características de Diseño. Modela la forma, dimensiones, tolerancias, acabado superficial, materiales, entre otras propiedades.
5. Información de Manufactura. Modela la definición de productos desde el punto de vista de sus aspectos de manufactura.
6. Propiedades. Representa las características cuantificables después de la manufactura del producto.

4.3 Estructura principal del MP

Como se mencionó anteriormente, la estructura del MP se basa en el paradigma orientado a objetos y provee el contexto para las estructuras de información las cuales representan las categorías de información enunciadas anteriormente. Como primera restricción sólo se utilizarán dos de las categorías de información para definir la estructura y contenidos del MP. Las categorías *estructura del producto* y *características de diseño* representan la necesidad de información para soportar ingeniería inversa considerando el método de manejo de información, y en particular el uso combinado de un MP y un modelo de manufactura (MM).

La notación en UML representa las clases por rectángulos con tres divisiones internas, estos compartimientos alojan el nombre de la clase, sus atributos y sus funciones. Las asociaciones semánticas son indicadas por líneas las cuales tienen algún elemento en sus extremos dependiendo del tipo de asociación. También se indica la cardinalidad de las asociaciones por medio de números. En el Apéndice B se presenta una breve descripción del lenguaje de modelado UML.

Es importante mencionar que los diagramas de clases mostrados en este capítulo están simplificados, ya que no se incluyen atributos ni funciones en los mismos. Los diagramas completos se presentan en la implementación (Capítulo 6).

El MP y su estructura se desarrollaron usando el UML como una herramienta de modelado semántico para el diseño y documentación de los esquemas. La figura 4.1 muestra el diagrama de clases de la estructura principal del MP y la relación entre las clases.

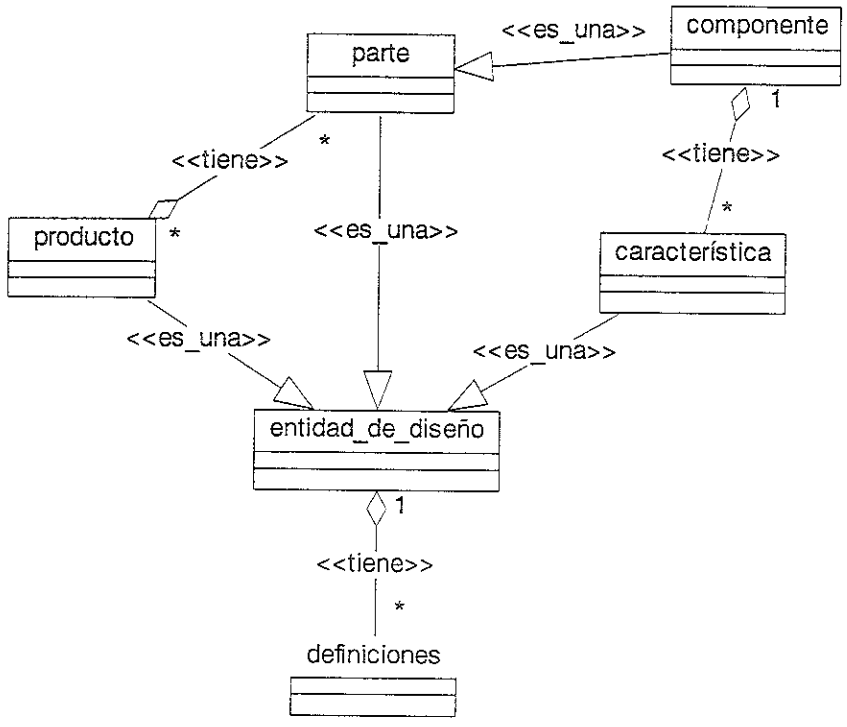


Figura 4.1 *Estructura principal del MP*

La estructura principal del MP se centra alrededor de la estructura física del producto y sobre las etapas de su ciclo de vida. Para el caso del MP simplificado una segunda restricción es que solo se basa en una de las tres etapas del ciclo de vida del producto definidas originalmente (Apéndice A). Esta es: el producto como es diseñado y planeado para manufactura. Dicha etapa será modelada por la clase *definiciones*. De esta forma podemos comenzar a notar las diferencias entre el esquema mostrado en la figura anterior y el de la figura A.1.

4.4 Modelado de la categoría de información "Estructura del Producto"

La categoría de información *estructura del producto*, la cual es la primera de tres categorías requeridas para el MP, es representada por las subclases de *entidad_de_diseño* y sus clases asociadas (parte superior del MP que se muestra en la figura 4.1).

Como se ilustra en el diagrama de clases (figura 4.1), el elemento núcleo del MP es la clase *entidad_de_diseño*. Una entidad de diseño es un objeto físico o elemento el cuál corresponde a la solución desarrollada a través de actividades de diseño con el objetivo de satisfacer un conjunto

de requerimientos. A través de sus subclases y sus asociaciones, se puede representar la categoría de información *estructura del producto*, de productos de referencia y substitutos[i].

La estructura de un producto está formada por entidades descompuestas en sus partes constitutivas, hasta llegar a un punto indivisible o al llegar al nivel deseado. La estructura es comúnmente representada usando diagramas de árbol jerárquicos, los cuales pueden modelar partes o productos completos: un producto puede ser subdividido en un gran número de partes y éstas a su vez pueden ser descompuestas en componentes.

Un *componente* está definido como un elemento indivisible físicamente, y está formado por *características* las cuales son formas básicas puestas juntas por razones funcionales o de manufactura. Aún cuando las *características* son partes de la estructura de un *componente*, no pueden estar solas físicamente, siempre deben asociarse con otras para constituir componentes o características más complejas.

Productos, *partes* y *características* son entidades de diseño particulares. Las asociaciones *tiene* en el diagrama de la figura 4.1 representan la estructura de productos: un *producto* tiene *partes*, un *componente* tiene *características*.

4.4.1 La taxonomía *característica*

Una gran variedad de taxonomías pueden ser asociadas a los elementos de la estructura del MP, a continuación se presenta en la figura 4.2 la taxonomía *característica* para representar partes rotacionales producidas por operaciones de maquinado y en particular por torneado.

Considerando que el lector está familiarizado con la mayoría de los nombres seleccionados para las clases, solo dos características secundarias serán explicadas: *término* y *de_transición*. 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 otra característica primaria.

[i] Un producto está en un proceso de rediseño y eventualmente será manufacturado "producto substituto" o ya existe "producto de referencia".

Cada clase de la taxonomía *característica* intenta ser lo suficientemente genérica para permitir diferentes definiciones. Es importante resaltar que la taxonomía presenta solo la estructura, otra información, como por ejemplo la geometría se representa usando otras clases.

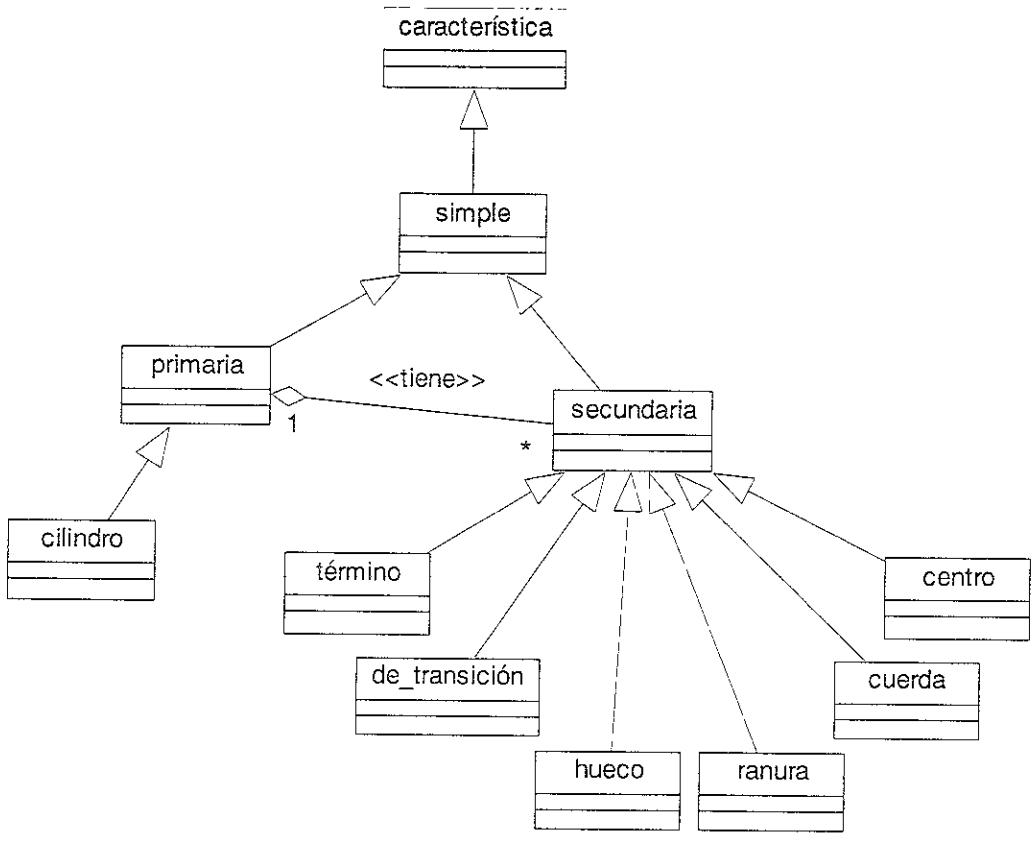


Figura 4.2 *Taxonomía característica para representar partes rotacionales maquinadas*

4.5 ¿Cómo definir entidades de diseño?

Existen diferentes tipos de definiciones que pueden ser usadas para la definición de productos, incluyendo dibujos, modelos geométricos sólidos, representaciones de frontera, patrones de esfuerzo, diagramas de bloques, etc. En este trabajo dicha definición se realiza desde dos perspectivas de diseño para soportar ingeniería inversa: su aspecto físico (características de diseño) y sus aspectos de manufactura (información de manufactura). Estos corresponden a dos categorías de información requeridas por el MP y son modeladas usando dos clases como se muestra en la figura 4.3. Las clases son *descripción_física* e *información_de_manufactura*.

La cardinalidad uno a uno la cual asocia la clase *definiciones* con las otras, implica que una combinación particular de instancias de las dos subclases determina una definición específica de la entidad de diseño. Consecuentemente, si cualquiera de estos cambia, se creará una definición diferente de la misma entidad de diseño. De esta forma se pueden representar soluciones alternativas o versiones de diseño diferentes de la misma entidad.

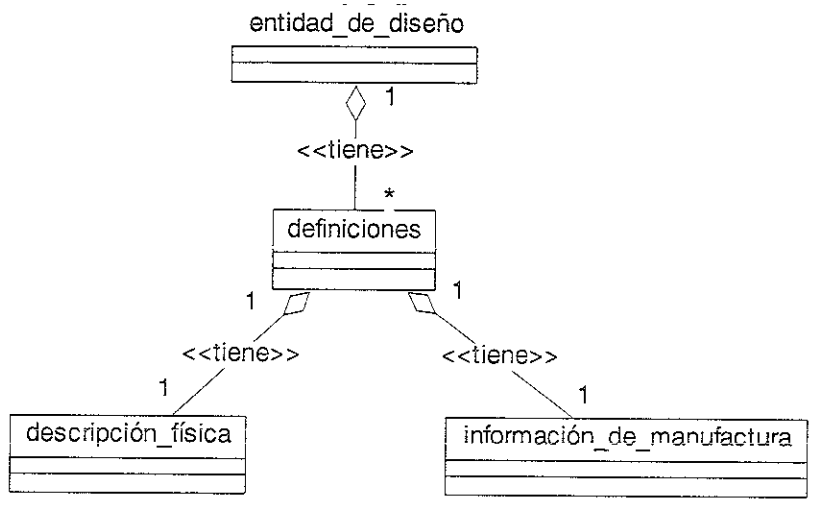


Figura 4.3 *Clases de definición del MP*

La clase *definiciones* es usada para representar una versión particular de la entidad y es capaz de soportar el proceso de rediseño de una entidad substituta, describiendo la entidad desde diferentes perspectivas o usando una variedad de tamaños. Otras alternativas de solución relacionan etapas distintivas del proceso, incluyendo diseño conceptual y diseño de detalle. Estas pueden ser modeladas como una entidad de diseño y pueden tener varias *definiciones*.

4.6 Modelado de la categoría de información "Características de Diseño"

Las actividades principales de una persona mientras se encuentra diseñando una entidad de diseño son las de definir sus características. Estas características son: estructura, forma, dimensiones, tolerancias, acabado superficial, materiales y proceso de manufactura. Las características de diseño determinan la funcionalidad y todas las propiedades de entidades a través de las etapas de su ciclo de vida.

En el MP, las características de diseño están modeladas dentro de dos diferentes categorías de información. La estructura del producto ha sido discutida anteriormente y el proceso de

manufactura es analizado como parte de la información de manufactura (Apéndice A). La forma, dimensiones, tolerancias, acabado superficial y materiales son referidos como atributos físicos. Estos están modelados con la clase *descripción_física*.

Ésta es una metaclase que está asociada a diferentes taxonomías de clases de descripción física las cuales representan una variedad de productos, partes y características. Los atributos físicos son modelados con la estructura de información que se muestra en la figura 4.4. El propósito de los atributos *vector* y *punto* es el de localizar la posición y orientación de la entidad en el espacio. El atributo *material* representa las propiedades de los materiales de la entidad.

El método para modelar atributos físicos, excepto materiales, emplea una combinación de representaciones visuales (archivos) de 2D y/o 3D los cuales definen parámetros con una notación comúnmente utilizada en dibujos mecánicos, e información detallada acerca de los parámetros contenidos dentro del MP. La representación visual no está considerada como parte del MP, pero se mantiene una referencia hacia ella.

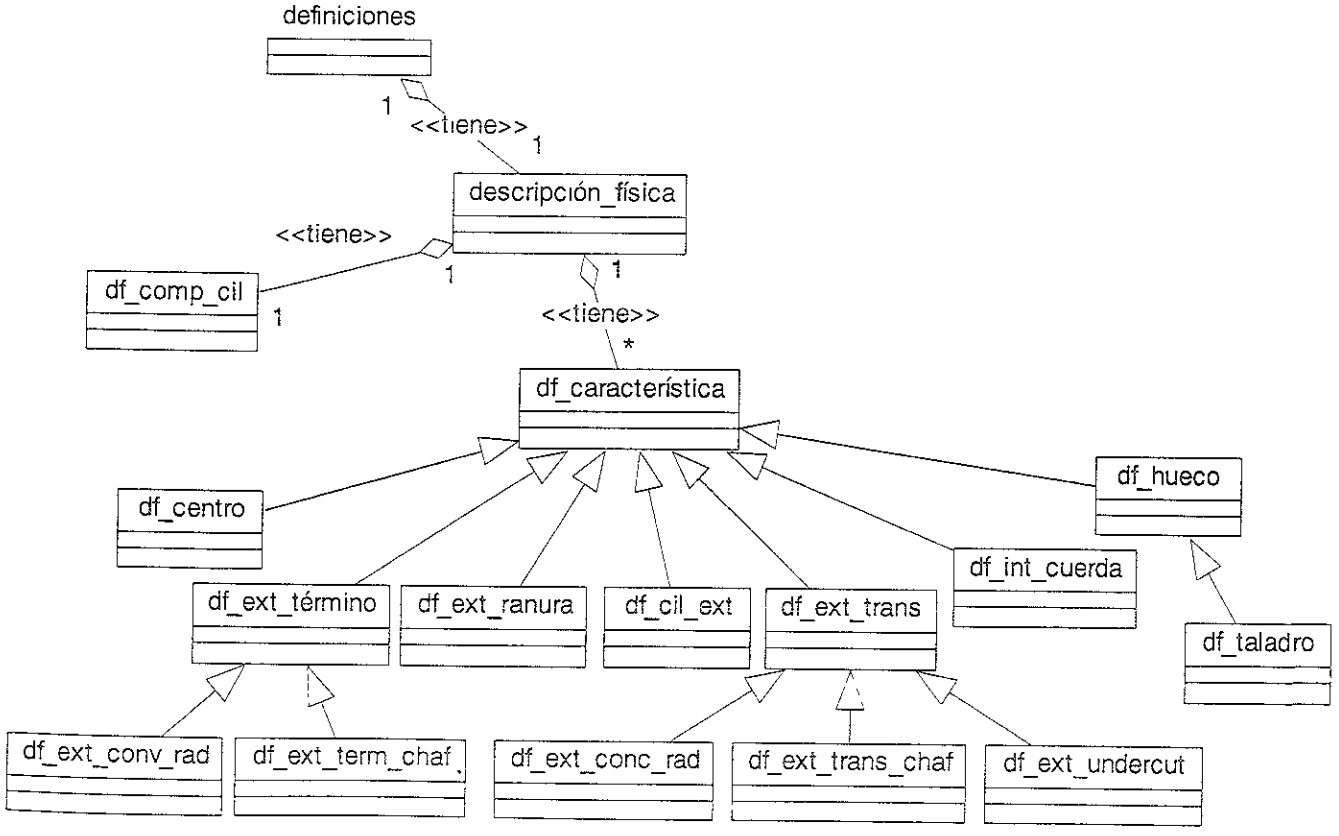


Figura 4.4 *Taxonomía descripción_física para representar partes rotacionales maquinadas*

El modelo hace posible una descripción flexible, fácil de entender y estructurada de los atributos físicos de componentes cuando son usados en diferentes niveles de la estructura del producto. Información esencial es incluida en los niveles superiores de la estructura (productos y partes) y la información detallada de cualquier entidad está disponible en los niveles inferiores (características).

4.6.1 Taxonomía de descripción física para representar partes rotacionales maquinadas

La figura 4.4 muestra la taxonomía para representar las características de diseño de partes rotacionales producidas con procesos de torneado. La taxonomía no pretende ser exhaustiva, pero es desarrollada a detalle.

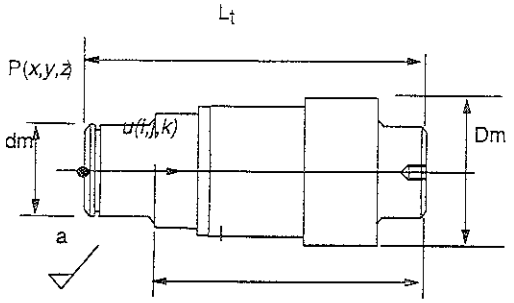
Dentro de la taxonomía, se incluye una clase para describir componentes cilíndricos, por ejemplo *df_comp_cil*. Esta clase pretende ser el elemento superior de la taxonomía la cual comprende descripciones particulares para cilindros, ejes, flechas y componentes de este tipo.

Las demás clases de la taxonomía contienen información sobre la descripción física de las características definidas inicialmente en la estructura de un componente. En el Capítulo 7 se muestra un ejemplo concreto de cómo se utiliza esta taxonomía.

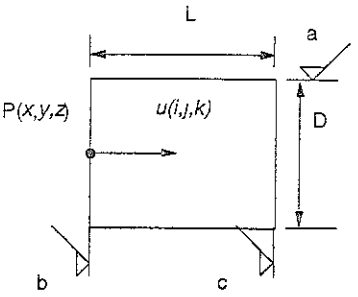
4.6.2 Representación de las clases de la taxonomía de descripción física

A continuación se muestra una representación simplificada de la información de las instancias creadas para describir componentes. Note que las instancias incluyen parámetros predefinidos. Los valores de algunos de sus atributos están en conjunto mientras que otros están siendo habitados durante el proceso de ingeniería inversa.

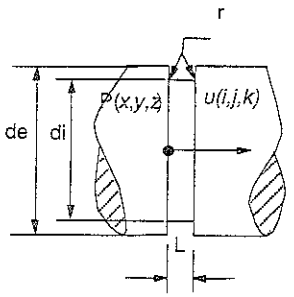
Los diagramas de la figura 4.5 corresponden a la representación gráfica de las características dentro del MP.



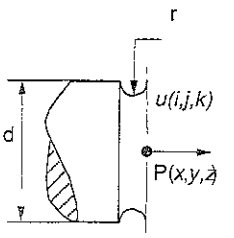
df_comp_cil



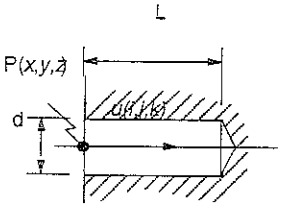
df_cil_ext



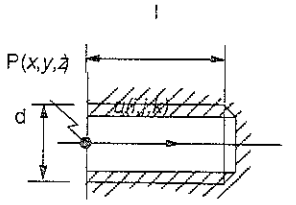
df_ext_ranura



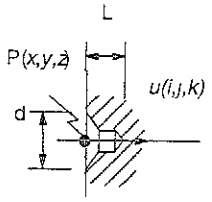
df_ext_undercut



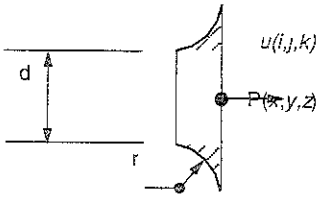
df_taladro



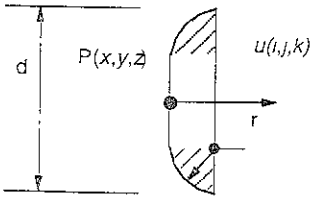
df_int_cuerda



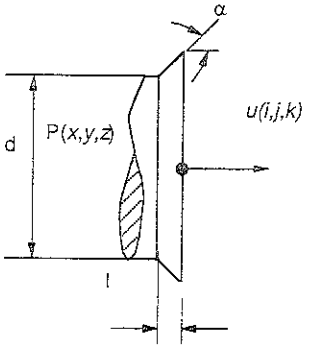
df_centro



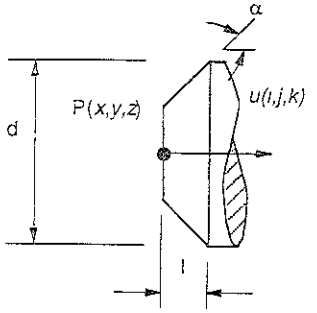
df_ext_conc_rad



df_ext_conv_rad



df_ext_trans_chaf



df_ext_term_chaf

Figura 4.5 *Representación de las instancias de la clase descripción_física (Borja, 1997)*

Es importante resaltar que la información detallada se encuentra en el nivel característica, mientras la representación del componente solo incluye parámetros generales como se muestra en la clase *df_comp_cil*, esto lo podemos observar en la siguiente figura 4.6. De esta forma es como se modelan las características de diseño dentro del MP.

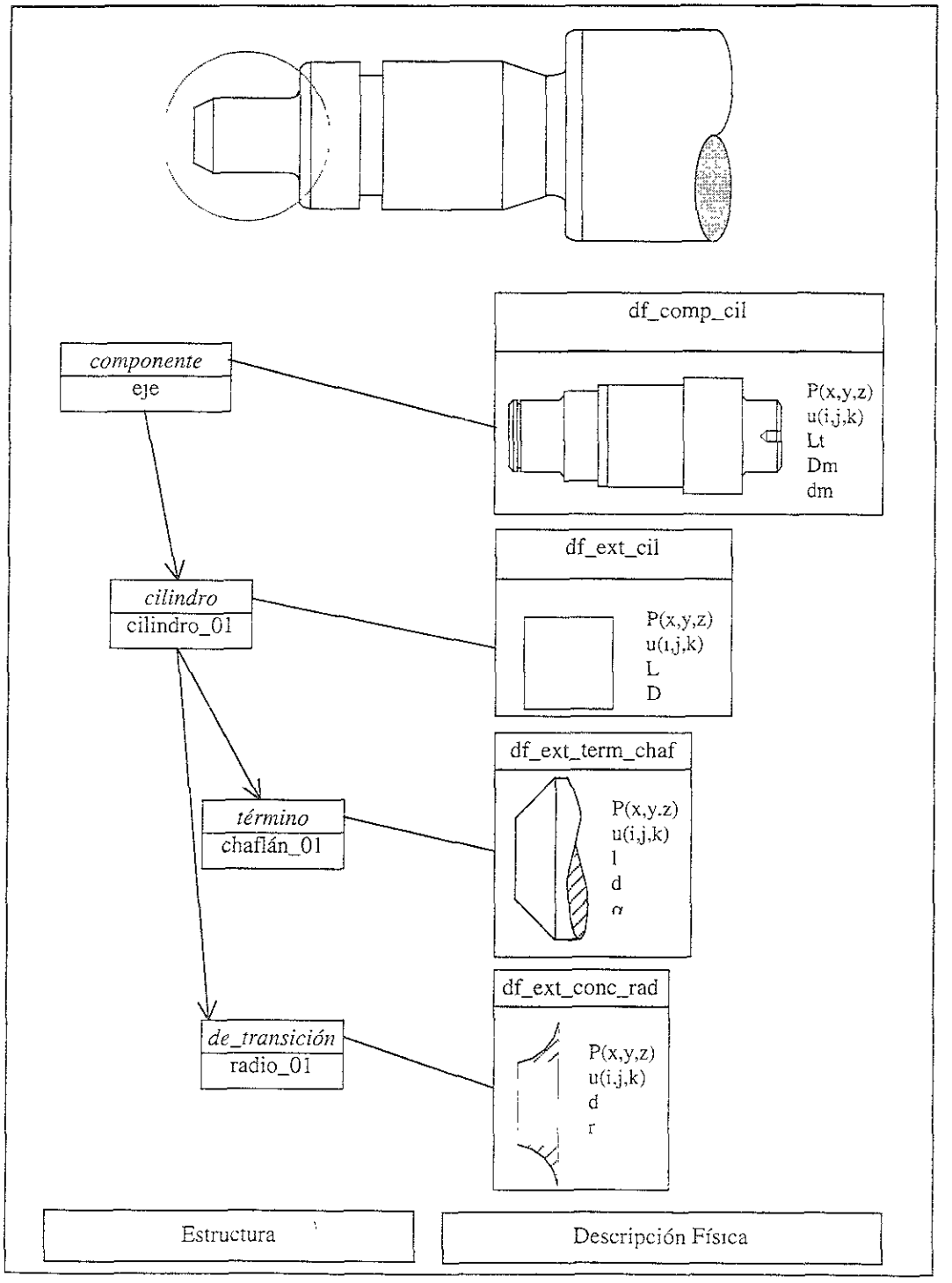


Figura 4.6 *Representación de un eje*

CAPÍTULO 5

DEFINICIÓN Y ALCANCE DEL SISTEMA A DESARROLLAR

5.1 Introducción

En el capítulo 4 se presentó el modelo del producto que forma parte del sistema SADET para asistir el proceso de rediseño concurrente. A continuación, en la primera parte de este capítulo se presenta la metodología que se utilizó para implementar el sistema, posteriormente en la Sección 5.3 se describe el programa Modelador de Ejes de Transmisión (MET). Por último en la Sección 5.4 se comenta sobre los recursos disponibles para la realización de la aplicación (MET). Los detalles de la implementación prototipo se describen en el Capítulo 6.

5.2 Método para el desarrollo

Para el desarrollo del sistema se utilizó el proceso que se muestra en la figura 5.1 y que a continuación se describe:

1. FASE INICIAL. En esta se determinaron los requerimientos del sistema (Capítulos 4 y 5), se incluye una definición del proyecto (Capítulo 3) y la determinación de requerimientos de hardware y software.
2. FASE DE PLANEACIÓN. Incluye la planeación de las actividades necesarias (Capítulo 3) y el establecimiento de los recursos requeridos para desarrollar el sistema.
3. FASE DE ELABORACIÓN. Comprende la especificación de las características del sistema y el diseño de su arquitectura (Capítulo 6).
4. FASE DE CONSTRUCCIÓN. Consiste en construir el producto como resultado de una serie de iteraciones (Capítulo 6), es decir existe la posibilidad de regresar a las primeras fases del proceso de desarrollo y realizar cambios en el diseño conceptual del sistema hasta que estemos satisfechos con el resultado obtenido.
5. INTEGRACIÓN DE DATOS. Por último se realizan pruebas integrando datos reales al sistema (Capítulo 7).

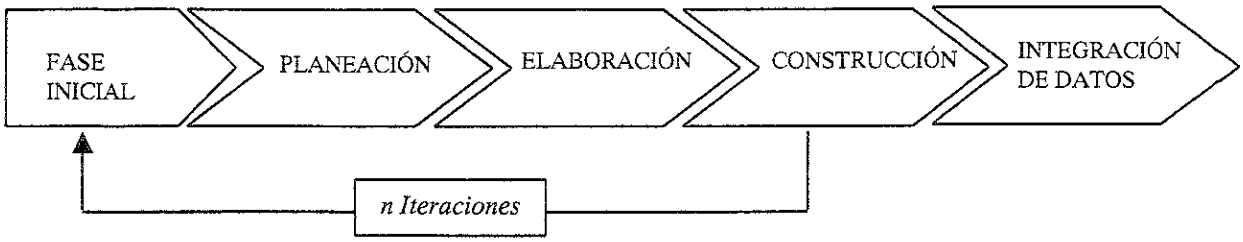


Figura 5.1 *Proceso de desarrollo del sistema*

5.3 Modelador de Ejes de Transmisión

El asistente Modelador de Ejes de Transmisión (MET) es una aplicación desarrollada para capturar información de componentes existentes. La implementación del MET asiste en la documentación de la estructura y geometría de ejes de transmisión y en general de partes rotacionales manufacturadas por procesos de torneado, creando modelos del producto y proporcionando una interface para poblar o llenar dichos modelos. El asistente MET también auxilia en el análisis de la información de productos de referencia.

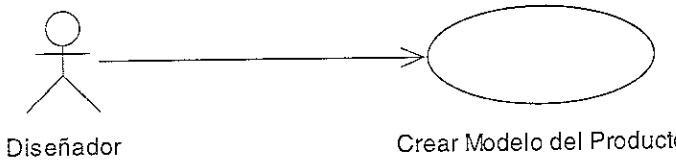
Se utilizará un diagrama de casos de uso como una técnica para captura de requisitos, estos diagramas tiene los siguientes elementos:

- Actores. Un actor representa cualquier cosa fuera del sistema que interactúa con él, pueden ser personas, otros sistemas, máquinas, etc.
- Casos de uso. Describe una secuencia de acciones realizadas por el sistema que conducen a un resultado de valor para el usuario, describe la funcionalidad del sistema independientemente de la implementación.
- Relaciones. Relación de comunicación entre elementos, se denota por medio de una flecha y es posible entre actor y caso de uso o entre casos de uso.

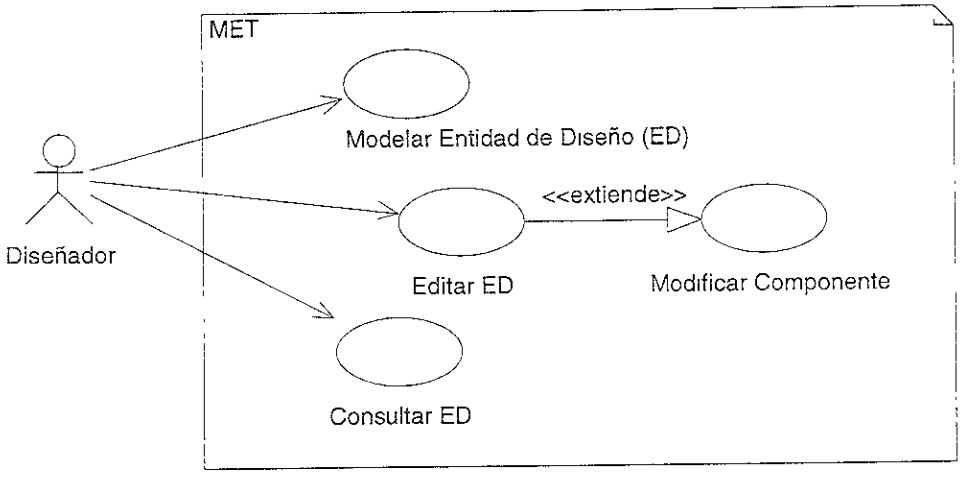
A continuación se resumen los requerimientos básicos de la aplicación, la terminología usada corresponde a los términos usados para la definición del MP. En la figura 5.2(a) se describen los actores que intervienen en el sistema, en la figura 5.2(b) se muestra la interacción del actor con la función global del sistema, un diseñador utiliza el sistema para crear un modelo del producto. En la figura 5.2(c) se detalla la funcionalidad del sistema incorporando mas casos de uso, el diseñador modela, edita y consulta entidades de diseño; el caso de uso *modificar componente* es un caso particular de edición de entidades de diseño.

Actor	Descripción
Diseñador o Usuario en General	Es la persona encargada de ingresar toda la información correspondiente al eje de transmisión. De momento consideraremos que dicha operación la realizará un *Diseñador.*

(a)



(b)



(c)

Figura 5.2 *Diagrama de Casos de uso para el MET*

Una característica importante de la aplicación, es que ésta soporta la documentación de componentes de una forma flexible, dándole al diseñador la opción de plantear o recuperar la estructura de partes antes de definir sus dimensiones, lo cual resulta conveniente para la conceptualización de productos. Otra característica es que facilita la recuperación de información y la posibilidad de editar componentes, de esta forma siguiendo las recomendaciones de otros programas del sistema se pueden crear nuevos componentes.

5.3.1 Elementos de la implementación experimental

Como se ha mencionado anteriormente, tanto el MP como la aplicación MET son parte de un sistema CAE para asistir el proceso de ingeniería inversa. En la figura 1.1 se muestra un esquema completo de dicho sistema en donde se remarcan los elementos desarrollados en esta tesis. A continuación se presenta en la tabla 5.1 la implementación de los elementos individualmente.

Tabla 5.1 *Implementación de los elementos MP y MET del sistema CAE*

Elemento	Parte	*	Alcance de la implementación
Modelo de Información del Producto		P	◦ Se implementa el modelo con las restricciones impuestas.
	Estructura	I	◦ Implementación completa para representar la estructura de componentes.
	Descripción física	I	◦ Se implementan las descripciones requeridas para representar ejes de transmisión y en general componentes rotacionales.
	Información de Manufactura	X	◦ No se implementó pero se deja una referencia para una futura implementación.
Ambiente de Especificaciones	Modelador de ejes de transmisión (MET)	I	◦ La implementación soporta la documentación de la estructura de componentes y sus características de diseño, también pretende contener a futuro una representación gráfica de los ejes.

* Niveles de implementación: I. Implementado, P. Parcialmente implementado, X. No implementado

5.4 Recursos necesarios

Uno de los principales objetivos de este trabajo es el modelar el MP propuesto por Borja (1997) en UML para posteriormente implementarlo empleando una base de datos orientada a objetos, de tal forma que como primer recurso debíamos contar con dicha base de datos, es importante mencionar que al principio no se contó con capacitación para utilizar la base de datos y también que el autor fue el primero del grupo de investigación en trabajar con ella. Al principio se utilizó *ObjectStore PSE Pro for C++* para implementar el MP, este es un programa que nos permite almacenar y recuperar objetos en su formato nativo C++, su desventaja es la capacidad de la base de datos que se puede crear, la ventaja es que una vez implementado el MP en este programa se

puede emigrar fácilmente hacia un miembro mas poderosos de su familia como es *ObjectStore 5.1*, sin necesidad de muchos cambios en el código. La familia *ObjectStore* cuenta con un conjunto de herramientas para el rápido desarrollo de bases de datos, entre estas, *Database Designer*. En este programa se diseña la base de datos utilizando como notación UML, y posteriormente el mismo programa crea los componentes básicos para una aplicación en *C++*.

Una vez implementado el MP en *ObjectStore* se hizo necesario desarrollar una aplicación (MET) que utilizara dicha base de datos, para lo cuál se utilizó *Microsoft Visual C++ 6.0*, una herramienta muy poderosa para el desarrollo de aplicaciones para Windows 32bits. Cabe destacar que *ObjectStore* también cuenta con un asistente para desarrollar aplicaciones junto *con Visual C++*, con lo que se vincula la aplicación con la base de datos.

Como plataforma computacional, se usó un sistema muy común en nuestros días, una computadora Pentium II 450Mhz con 64Mb en RAM y el sistema operativo Windows 98. La implementación fue realizada por el autor.

CAPÍTULO 6

IMPLEMENTACIÓN DE LA APLICACIÓN PROTOTIPO

6.1 Introducción

En este capítulo se describe la implementación experimental de los elementos del sistema descritos en el Capítulo 5. Como se explicó en el Capítulo 3, el proyecto consiste en la implementación de la base de datos del Modelo de Información del Producto (MP) y de un programa Modelador de Ejes de Transmisión (MET) que sea capaz de almacenar, recuperar y editar la información de la base de datos. En la Sección 6.2 se describe el desarrollo del MP y posteriormente en la Sección 6.3 el de la aplicación MET.

6.2 Implementación del MP

Los esquemas del marco de referencia (Sección 4.3), taxonomía *característica* (Sección 4.4.1), *características de diseño* (Sección 4.6) y taxonomía de *descripción_física* (Sección 4.6.1) se implementaron como se especificó en la tabla 5.1. Para ello se utilizó *Database Designer* de *ObjecStore*.

Database Designer nos ofrece un ambiente amigable para el desarrollo de bases de datos. Podemos usarlo con las notaciones de modelado OMT (*Object Management Team*) o UML. Una vez establecido el modelo completo de la base de datos el programa crea un archivo (.db) que contiene los componentes básicos para una aplicación en *C++* que utilize *ObjectStore 5.1*.

ObjectStore también cuenta con un asistente para el desarrollo de aplicaciones en *Visual C++*, el cuál utilizamos junto con el archivo generado por *Database Designer* para crear el proyecto que nos permitirá ligar la base de datos con un programa (MET) que nos proporcione una interfaz con el usuario.

La figura 6.1 presenta la implementación del modelo correspondiente al marco de referencia del MP junto con la taxonomía *característica*. Cabe mencionar que a diferencia de todos los diagramas UML presentados anteriormente, en los que se muestran en este capítulo se incluyen, en lo posible, los atributos y funciones de las clases, ya que estos fueron los diagramas utilizados para la programación de la base de datos.

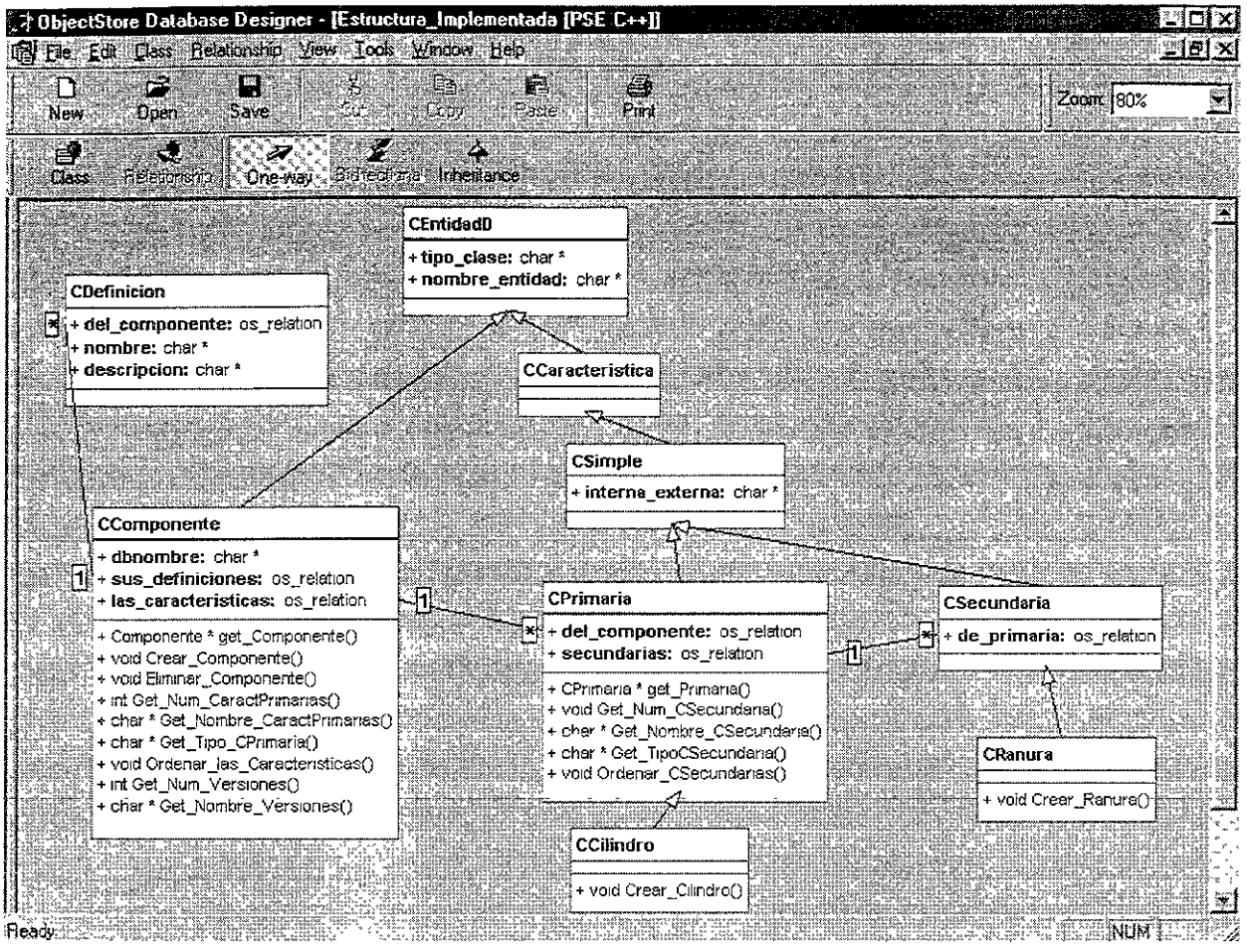


Figura 6.1 *Implementación en Database Designer del Marco de referencia del MP*

Un aspecto importante en la reciente implementación del MP, es que todas las clases del modelo formarán parte de un mismo proyecto en Visual C++, es decir que se modelan todos los esquemas de forma conjunta, aunque por razones de espacio en las figuras se muestren en forma separada. En la figura 6.2 se muestra la implementación de la categoría de información *características de diseño*, es decir la taxonomía de *descripción_física*.

De esta forma queda definida la arquitectura de la base de datos, en el Apéndice D se presentan algunos ejemplos de las definiciónes de clases del MP, así como el archivo *schema.osg*, el cual es un esquema de la base de datos, en éste archivo se listan los tipos de clases que pueden ser almacenados en la base de datos y de cuales se pueden formar colecciones o arreglos de varios elementos del mismo tipo

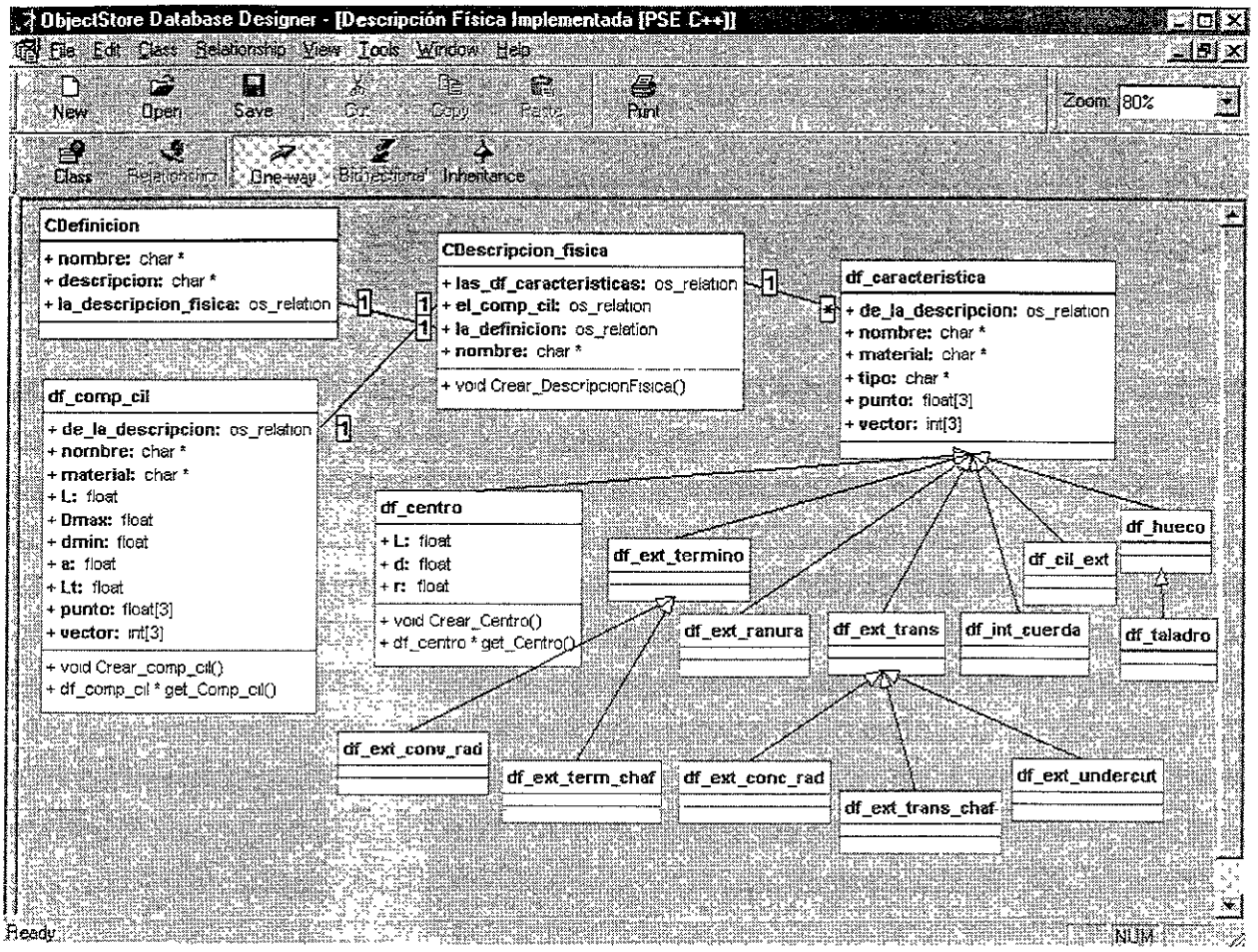


Figura 6.2 *Implementación en Database Designer de la taxonomía descripción_física*

6.3 Implementación del Modelador de Ejes de Transmisión

6.3.1 Modelo conceptual del MET

Un modelo conceptual tiene como objetivo explicar los conceptos significativos en el dominio del problema. Cabe resaltar que una cualidad esencial que debe ofrecer un modelo conceptual es que representa cosas del mundo real, no componentes de software. Comenzamos por identificar los conceptos, que se muestran a continuación, considerando que el lector está familiarizado con la mayoría de los conceptos seleccionados , éstos no serán explicados.

Depto. de Diseño	ModeloUnaEntidad	Modificar Componente	Estructura de ED
Estación de Diseño	DiseñoUnaEntidad	Entidad de Diseño (ED)	Producto
Diseñador	Modelar ED	Definición de ED	Parte
Diseño	Editar ED	Descripción física de ED	Componente
Modelo	Consultar ED	Inf. Manufactura de ED	Característica

Una vez identificados los conceptos buscaremos sus asociaciones y atributos. Un buen modelo conceptual capta las abstracciones esenciales y la información indispensable para comprender el dominio dentro del contexto de los requerimientos; nos facilita además conocer el dominio, sus conceptos, su terminología y sus relaciones.

A continuación se muestra en la figura 6.3 el Modelo Conceptual del dominio del MET.

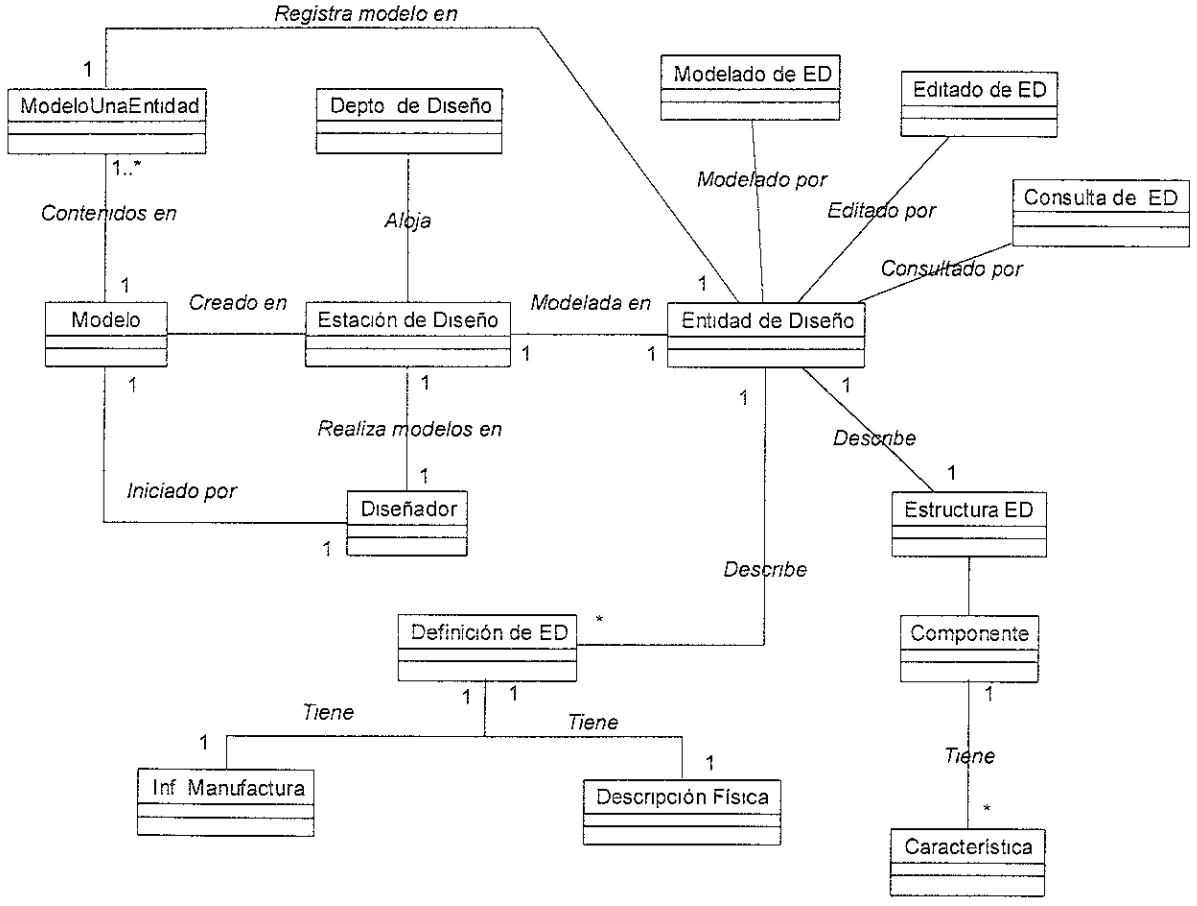


Figura 6.3 *Modelo conceptual del dominio del MET*

La parte central del modelo es el concepto de *entidad de diseño*, ésta es modelada en una *estación de diseño*, la cual se encuentra físicamente en el *departamento de diseño* y es utilizada por un *diseñador* para crear *modelos de entidades de diseño*, es decir modelos de productos.

Los conceptos que se muestran a la derecha de *entidad de diseño* se refieren a las operaciones que se pueden realizar sobre la entidad y por último los conceptos de la parte inferior del diagrama nos representan la estructura y descripción física de la entidad de diseño, lo que significa que incluimos al MP en el dominio del MET.

6.3.2 Comportamiento del MET

Antes de iniciar el diseño lógico de cómo funciona una aplicación de software, es necesario investigar y definir su comportamiento como una "caja negra". El *comportamiento del sistema* es una descripción de lo que hace, sin explicar la manera en que lo hace. Los objetos de un sistema interactúan para realizar colectivamente los servicios ofrecidos por las aplicaciones.

Para describir el comportamiento del sistema utilizamos los diagramas de secuencia (en UML) del sistema, éstos muestran la secuencia de interaciones entre objetos durante un escenario concreto, cada objeto está representado por una barra vertical, donde el tiempo transcurre de arriba abajo. A continuación se muestran en la figura 6.4 los diagramas correspondientes a los casos de uso y sus escenarios principales mostrados en la tabla 6.1.

Tabla 6.1 *Casos de uso y sus principales escenarios*

Caso de Uso	Escenarios principales
Modelar Entidad de Diseño	Modelar Estructura Modelar Geometría
Editar Entidad de Diseño	Eliminar Entidad de Diseño Copiar Componente
Consultar Entidad de Diseño	Consultar Entidad de Diseño
Modificar Componente	Modificar Componente

El primer renglón de la tabla 6.1 nos dice que para el caso de uso modelar entidad de diseño existen dos escenarios posibles, el modelar la estructura de la entidad de diseño y el modelar su geometría, éstos escenarios se representan en los dos primeros diagramas de la figura 6.4. Los diagramas nos describen la interacción entre el diseñador y el sistema, en el primer diagrama el diseñador desea comenzar a modelar la estructura de una entidad de diseño para lo cual el sistema muestra una lista de características primarias, el diseñador selecciona una y se manda el mensaje al sistema para crear ese tipo de característica, posteriormente el sistema muestra una lista de los tipos de características secundarias que se pueden asociar a la característica primaria, nuevamente se envía el mensaje de crear ese tipo de característica secundaria y por último el sistema guarda la información. De esta forma es como el diseñador va definiendo la estructura de la entidad de diseño.

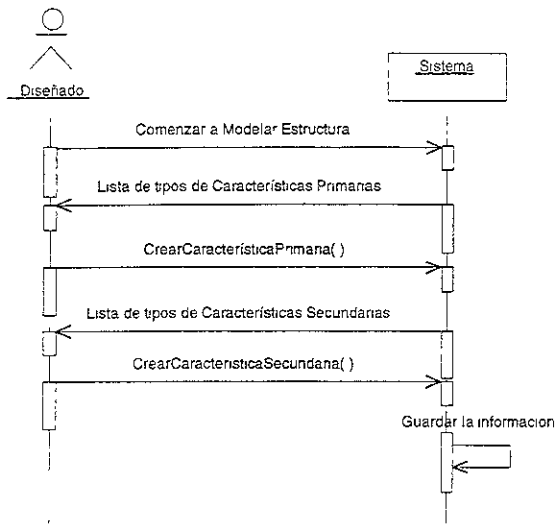


Diagrama de secuencia del escenario Modelar Estructura

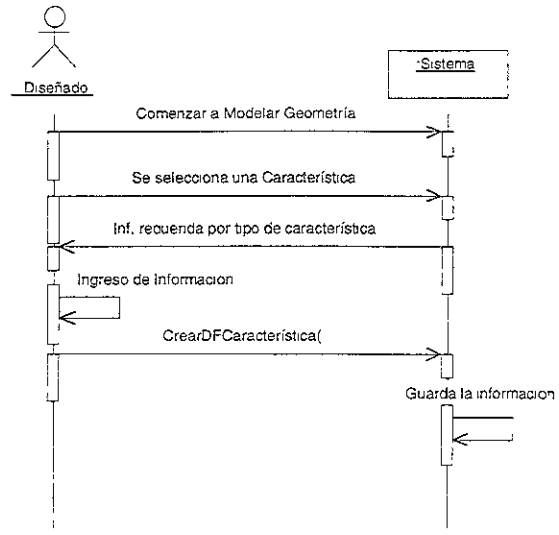


Diagrama de secuencia del escenario Modelar Geometría

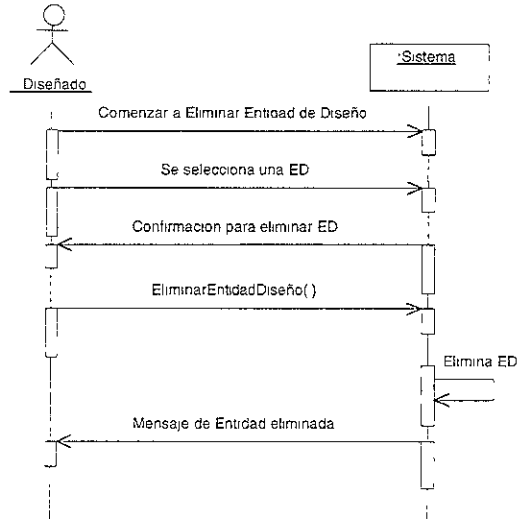


Diagrama de secuencia del escenario Eliminar ED

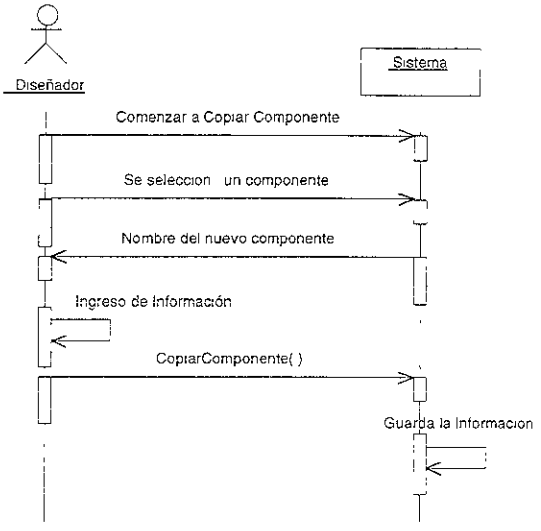


Diagrama de secuencia del escenario Copiar Componente

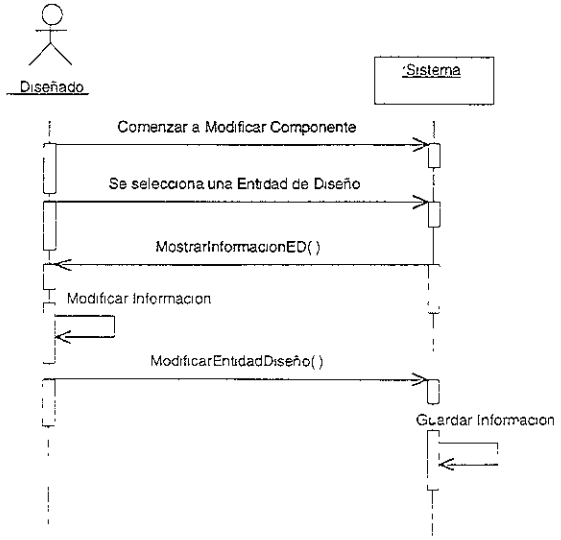


Diagrama de secuencia del escenario Modificar Componente

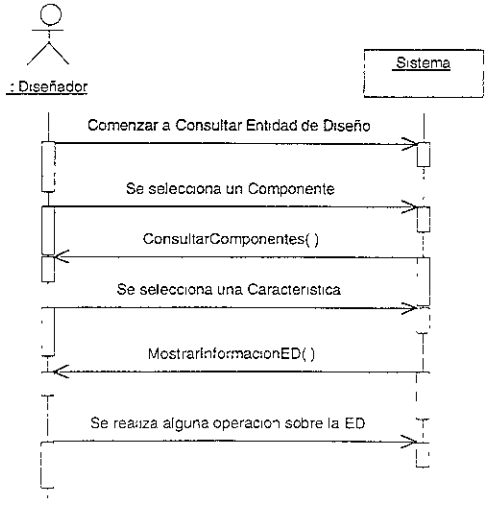


Diagrama de secuencia del escenario Consultar ED

Figura 6.4 *Diagramas de secuencia del MET*

6.3.3 Desarrollo de la aplicación MET

El MET se desarrolló en el ambiente de programación *Visual C++* 6.0 y los pasos que se siguieron fueron los siguientes:

- Creación del proyecto del tipo de documento único, utilizando como vista o pantalla principal un diálogo común.
- Inclusión del soporte requerido para trabajar con *ObjectStore*, instrucciones para compilar el esquema de la base de datos (*schema.osg*).
- Definición de la apariencia del programa, es decir la interfaz de usuario, comenzando con el menú y a continuación las principales ventanas.
- Creación de las clases pertenecientes al dominio del sistema.
- Dar funcionalidad a las ventanas y vincularlas con la capa del dominio y la base de datos.

Como se ha mencionado anteriormente, el asistente Modelador de Ejes de Transmisión (MET) proporciona una interfaz para poblar o llenar modelos del producto, específicamente da la opción de crear y editar ejes, definir su estructura y su descripción física. A continuación se muestran las principales ventanas de la aplicación MET, es decir la interfaz de usuario del programa, así como su entorno de programación.

La figura 6.5 nos muestra la pantalla principal del ambiente de desarrollo Visual C++, ésta consta de dos secciones principales. el espacio de trabajo (*workspace*) y el documento o documentos en los que se está trabajando. En el *workspace* se muestra un diagrama de árbol de todo el proyecto y se pueden escoger tres diferentes vistas de éste: los archivos, las clases y los recursos visuales.

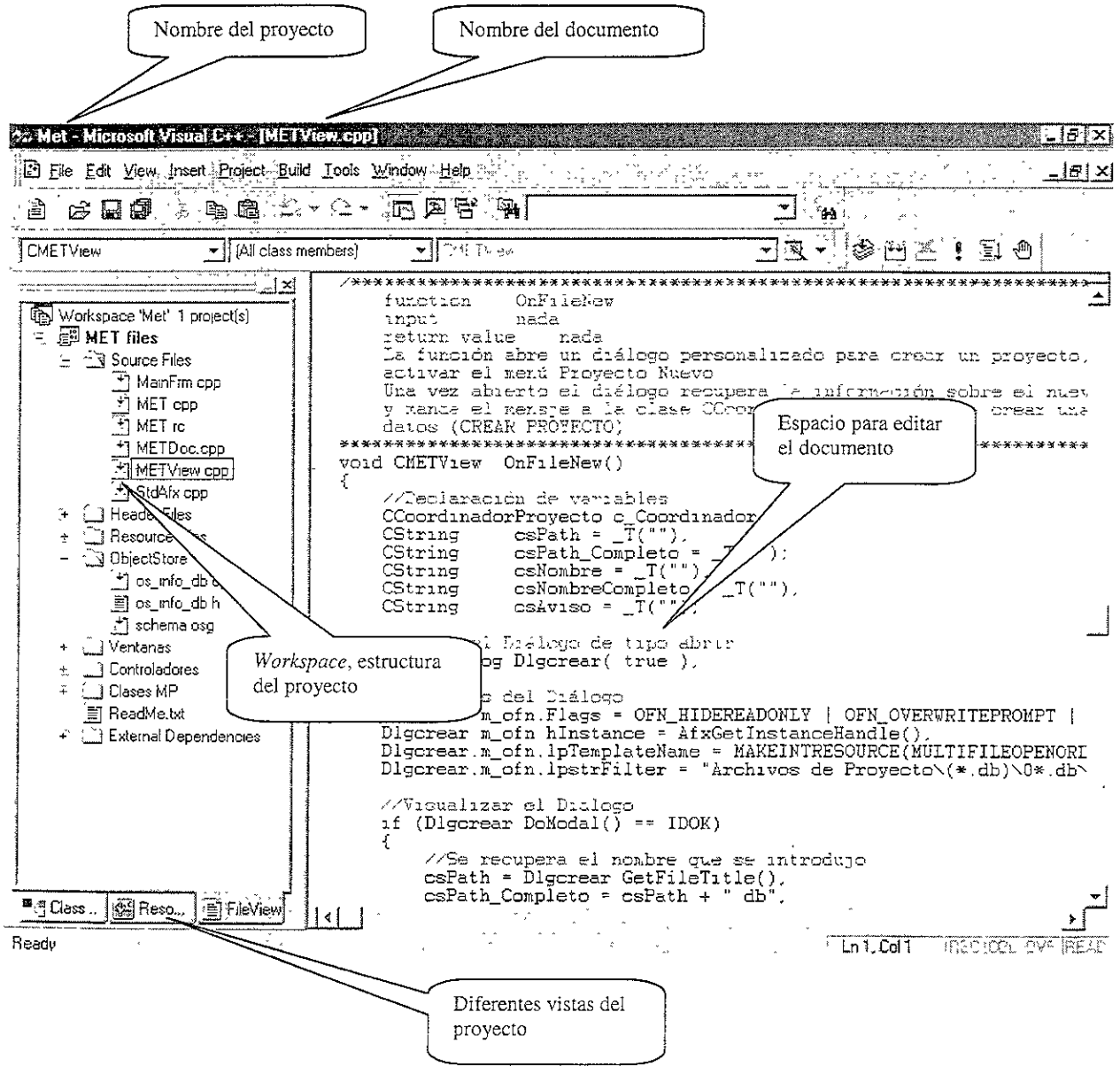


Figura 6.5 *Ambiente de desarrollo Visual C++*

En la figura 6.6 se muestra la ventana principal de la aplicación MET, en la barra superior se muestran el nombre de la base de datos en la que se está trabajando, el nombre del eje y su versión. Se muestra también un diagrama de árbol representando la estructura general del eje que se tiene abierto y del lado derecho de este diagrama se muestra información general sobre el eje y la persona que trabaja en él.

Como paso inicial para comenzar a trabajar el diseñador debe crear un proyecto, es decir una nueva base de datos, posteriormente podrá crear los ejes que desee dentro de este proyecto.

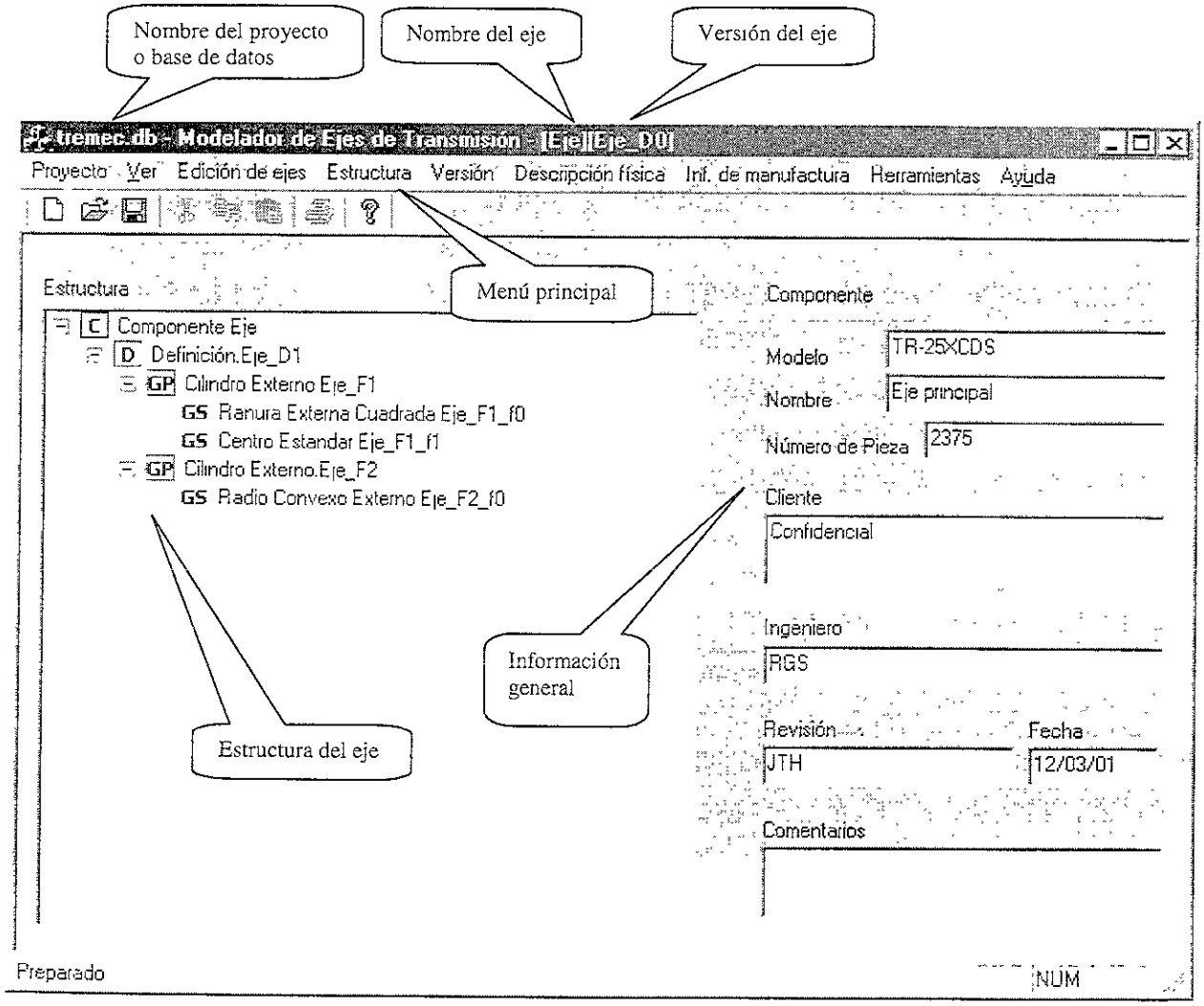


Figura 6.6 *Ventana principal de la aplicación MET*

Como se mencionó anteriormente una vez que se abrió un nuevo proyecto, es necesario crear un eje, para ello seguimos la siguiente ruta: en el menú seleccionamos "Edición de ejes" y luego "Crear", ahora se escribe el nombre del eje y se oprime el botón "Crear Eje". La figura 6.7 muestra la ventana para crear un eje. Al realizar este procedimiento la aplicación está creando un objeto de la clase *componente* (se muestra en un círculo en el diagrama de clases anexo a la figura 6.7) y le asigna el nombre que hemos elegido.

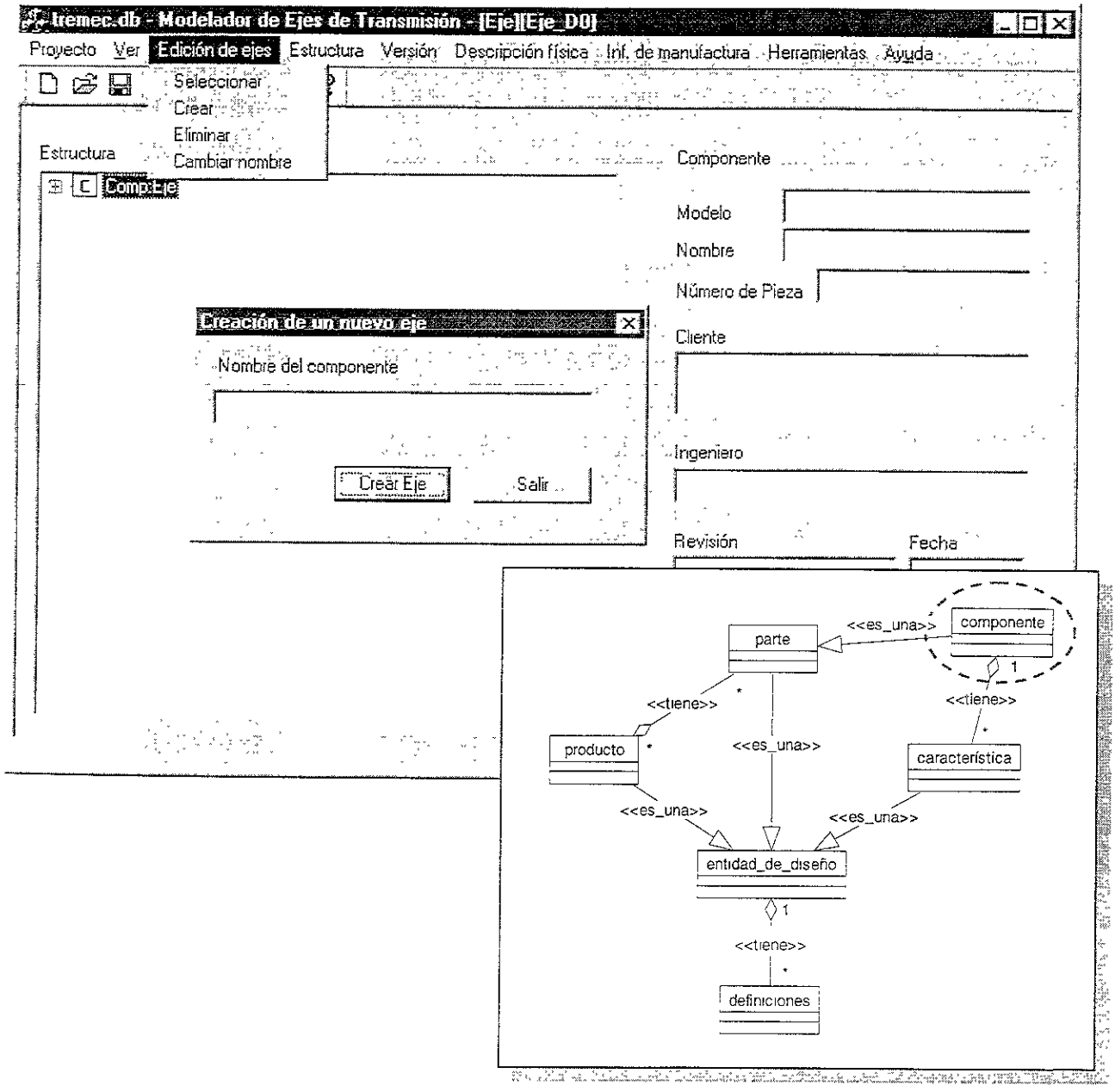


Figura 6.7 *Interfaz para crear un eje*

Al crear un eje se crea también una versión original de él, recordemos que una versión o definición de un componente nos sirve para modelar sus características de diseño. Aparte de la versión original existirán ocasiones en que sea necesario crear nuevas versiones del eje, para ello seguimos la siguiente ruta: en el menú seleccionamos "Versión" y luego "Crear", ahora se introduce una descripción de la versión y se oprime el botón "Crear Versión". La figura 6.8 muestra la ventana para crear una versión. Al realizar este procedimiento la aplicación está creando un objeto de la clase *definiciones* (se muestra en un círculo en el diagrama de clases anexo a la figura 6.8) y le asigna el nombre que como hemos visto crea automáticamente el programa.

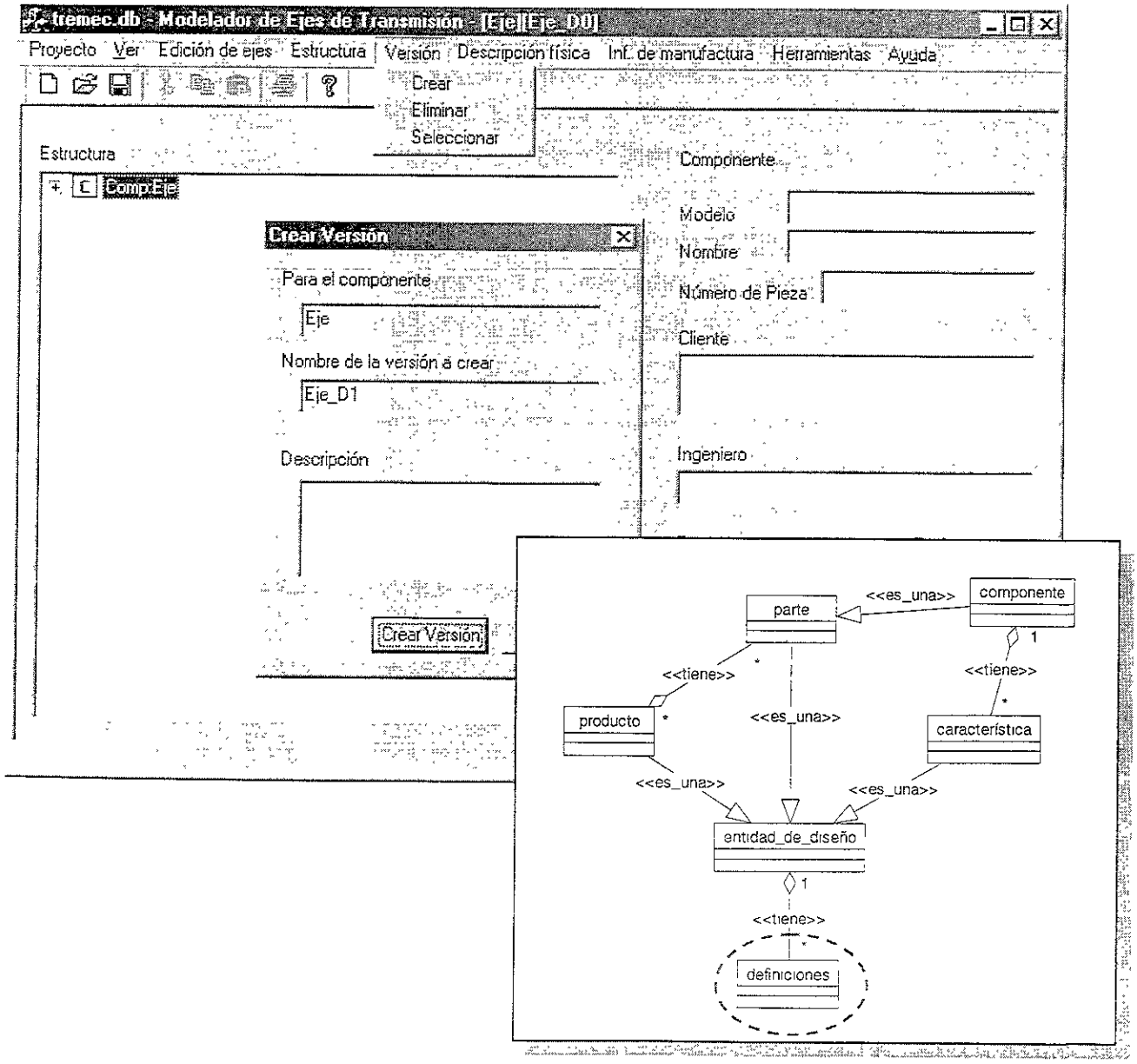


Figura 6.8 *Interfaz para crear una versión*

Una vez que se han creado uno o varios ejes y sus definiciones, es necesario poder escoger con cual trabajar, para ello se da la opción de seleccionarlos.

Ruta para seleccionar un eje (figura 6.9).

Edición de ejes/Seleccionar/Escoger un eje de la lista/Abrir/Ok

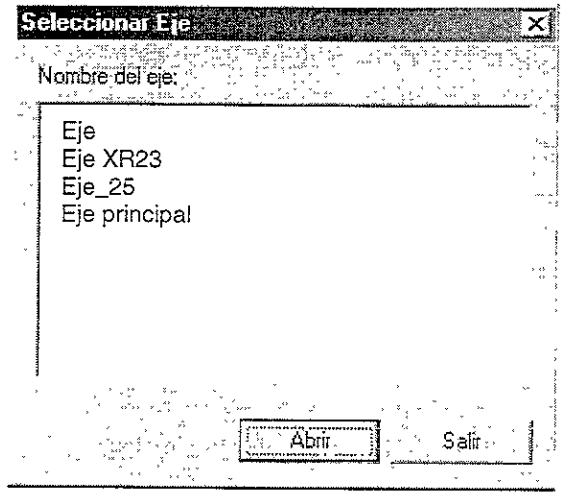


Figura 6.9 *Interfaz para seleccionar un eje*

Ruta para seleccionar una versión (figura 6.10).

Versión/Seleccionar/Escoger una versión de la lista/Abrir/Ok

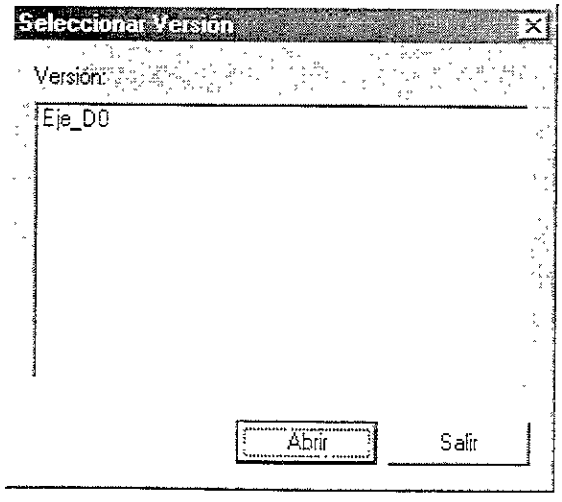


Figura 6.10 *Interfaz para seleccionar una versión*

Nuestro siguiente paso será definir la estructura del eje comenzando por una característica o geometría primaria como se nombró en la interfaz de usuario para que se entendiera mejor. La ruta para crear una geometría primaria (figura 6.11) es la siguiente:

Estructura/Crear geometría primaria/Escoger el tipo/Crear/Ok

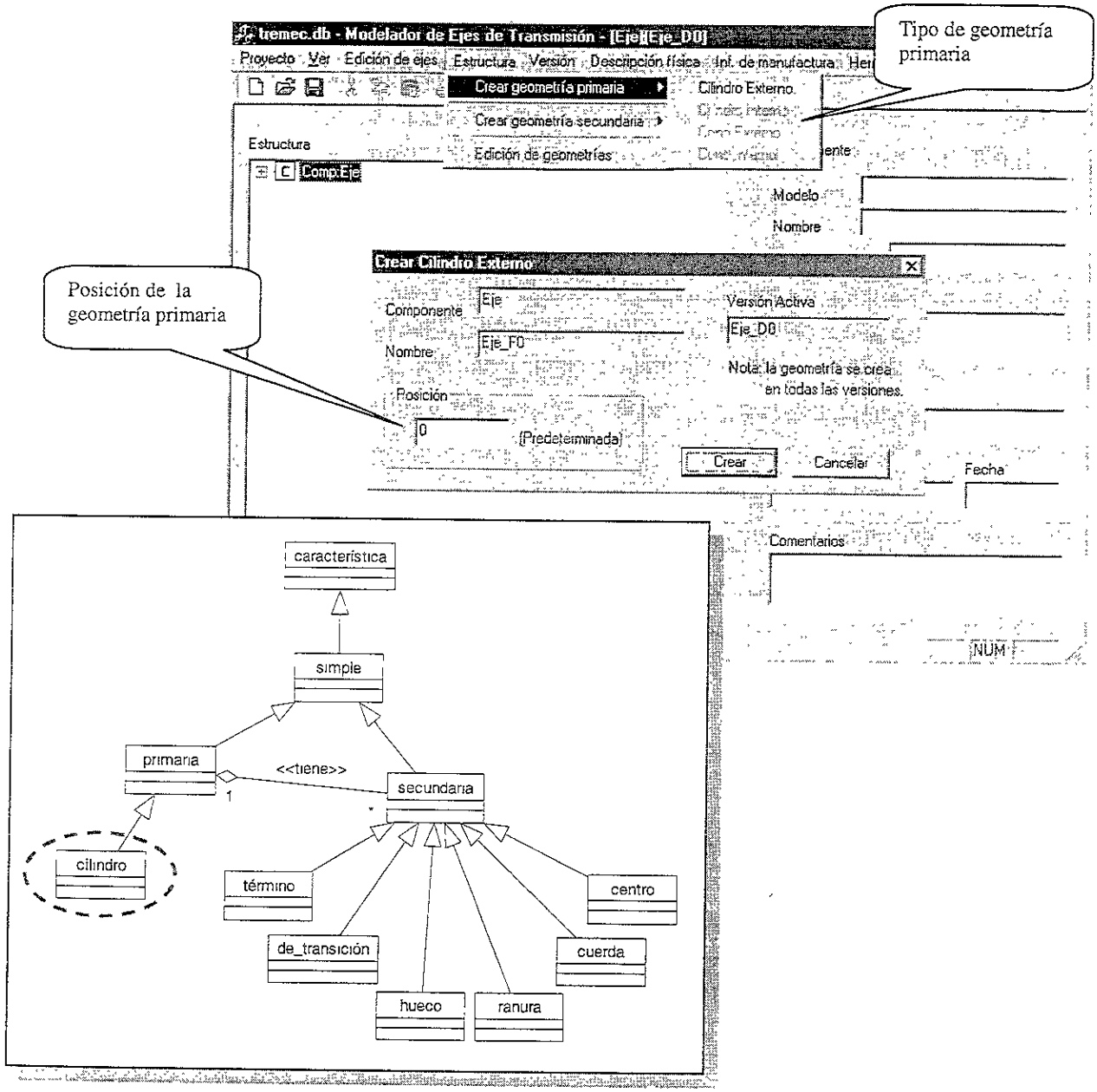


Figura 6.11 *Interfaz para crear una geometría primaria*

Al realizar este procedimiento la aplicación está creando un objeto de la clase *cilindro* (se muestra en un círculo en el diagrama de clases anexo a la figura 6.11) y le asigna el nombre predeterminado y lo ubica en la posición especificada por el usuario.

Ya que se creó una geometría primaria podemos continuar definiendo sus geometrías secundarias, para lo cual se siguen las siguientes instrucciones: en el menú seleccionamos "Estructura" y luego "Crear geometría secundaria", ahora escogemos el tipo de geometría secundaria a crear, posteriormente se elige a que geometría primaria se le asignará y por último se oprime el botón "Crear". La figura 6.12 nos muestra la ventana para crear una geometría secundaria del tipo radio convexo externo.

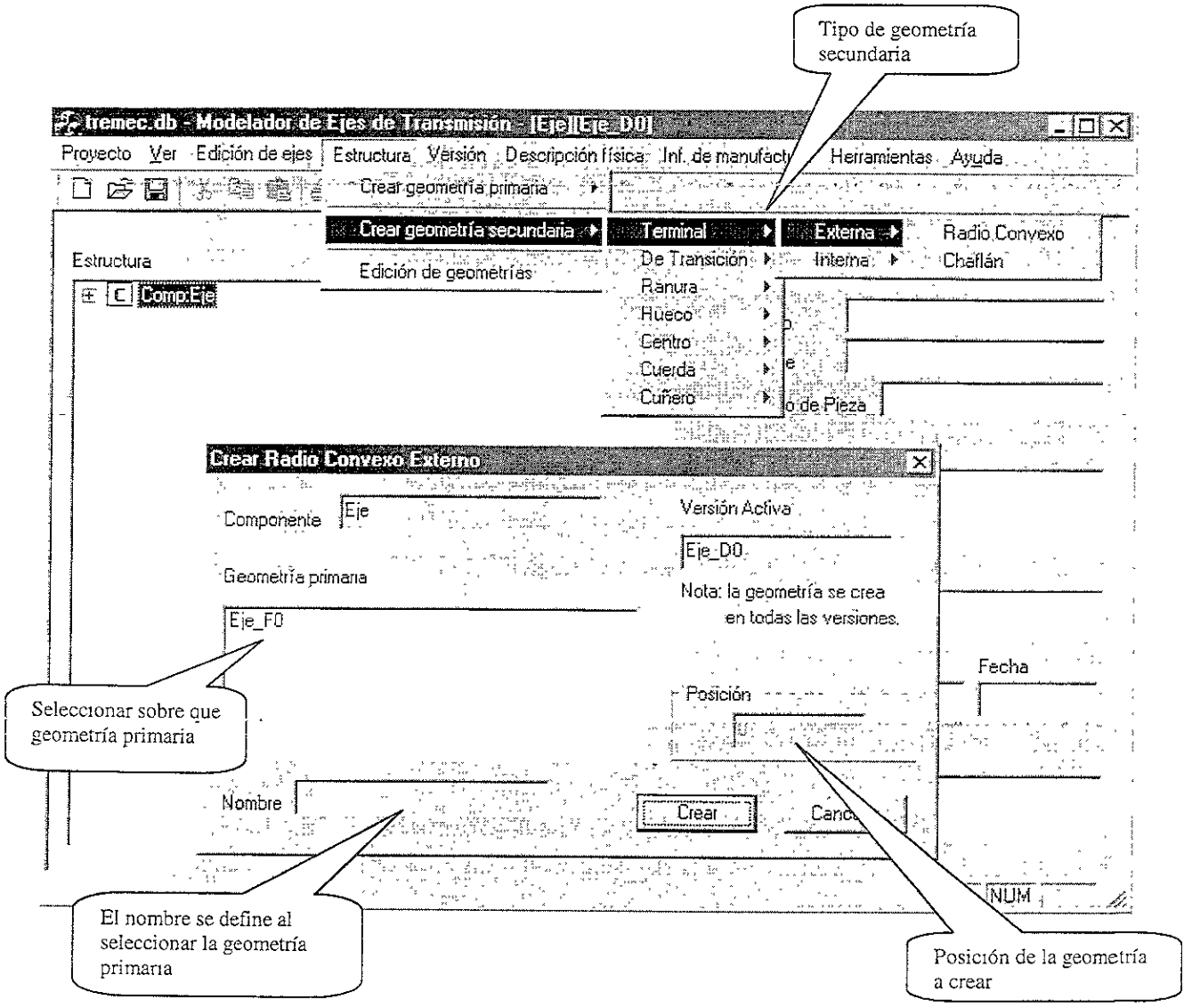


Figura 6.12 *Interfaz para crear una geometría secundaria*

Ya que han sido creadas las geometrías es necesario tener una manera de editarlas, es decir de eliminarlas o moverlas de lugar, para lo cuál, se utiliza la ventana de edición de geometrías que se muestra a continuación

Ruta para editar geometrías (figura 6.13).

Estructura/Edición de geometrías/Escoger una geometría/Escoger operación/Salir

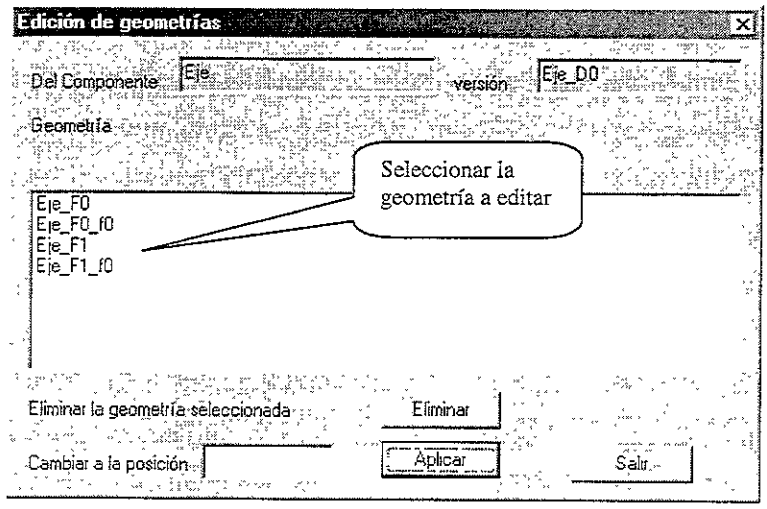


Figura 6.13 *Interfaz para editar geometrías*

Una vez definida la estructura del eje estamos en posibilidad de hacer su descripción física, para lo cual se deben definir las dimensiones de las geometrías utilizando la ventana de definición de dimensiones mostrada en la figura 6.14. Para realizar este procedimiento se usa la siguiente ruta:

Descripción física/Definir Dimensiones/Escoger una geometría/Definir atributos/Salir

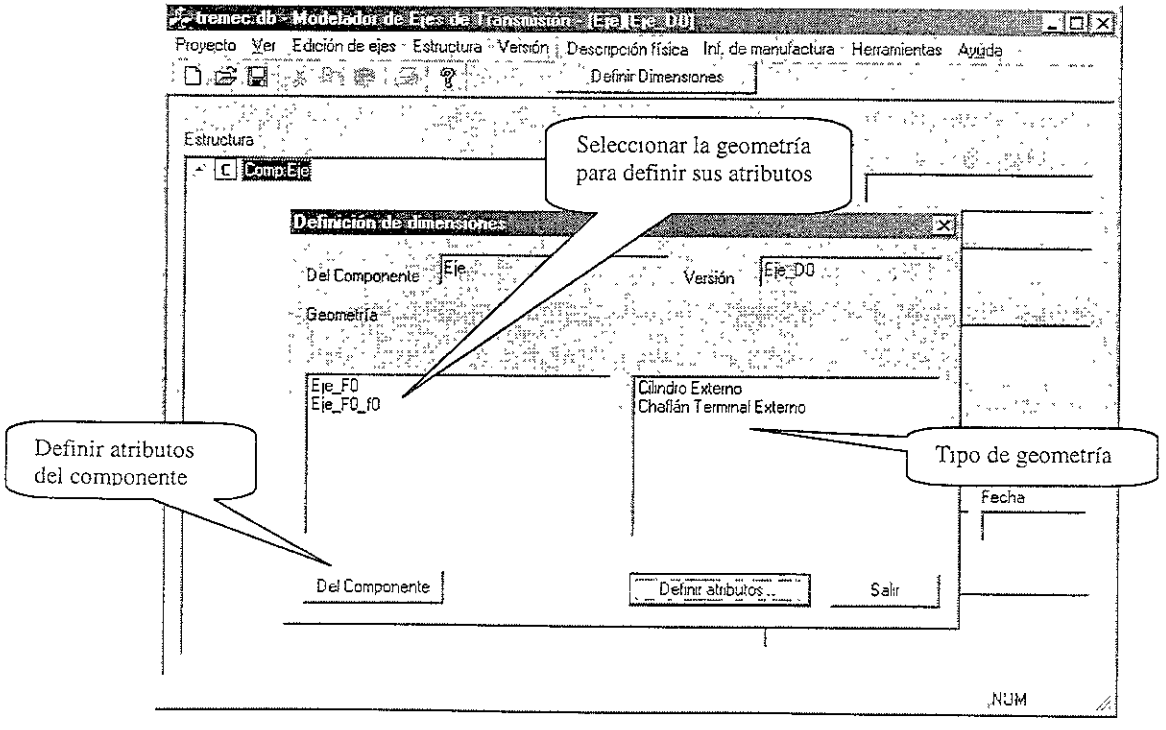


Figura 6.14 *Interfaz para definir dimensiones*

En la figura 6.14 vemos que es posible definir los atributos del componente completo (principales características) o escoger una geometría y definir sus dimensiones. La figura 6.15 muestra la interfaz para definir los atributos de un componente cilíndrico y en el diagrama de clases anexo, el tipo de objeto que se está creando en la base de datos.

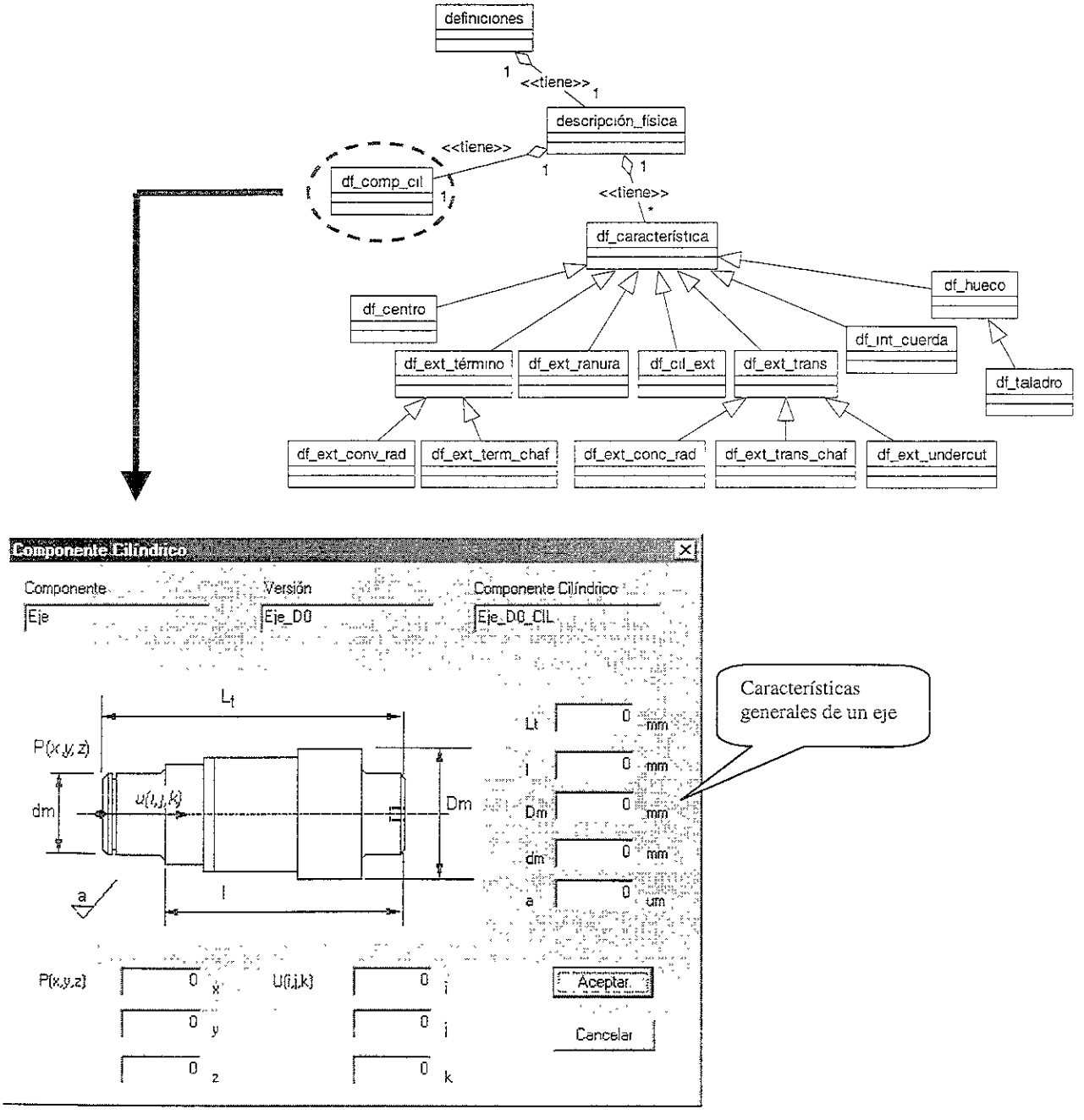


Figura 6.15 *Interfaz para definir atributos de un componente cilíndrico*

La figura 6.16 muestra la interfaz para definir las dimensiones de una geometría, en este caso se representa un cilindro externo pero se debe mencionar que todas las geometrías tienen una ventana para definir sus atributos, aunque con fines de ejemplificar solo se presenta una.

Ruta para definir dimensiones de una geometría (figura 6.16).

Definir atributos/Ingresar la información/Aceptar

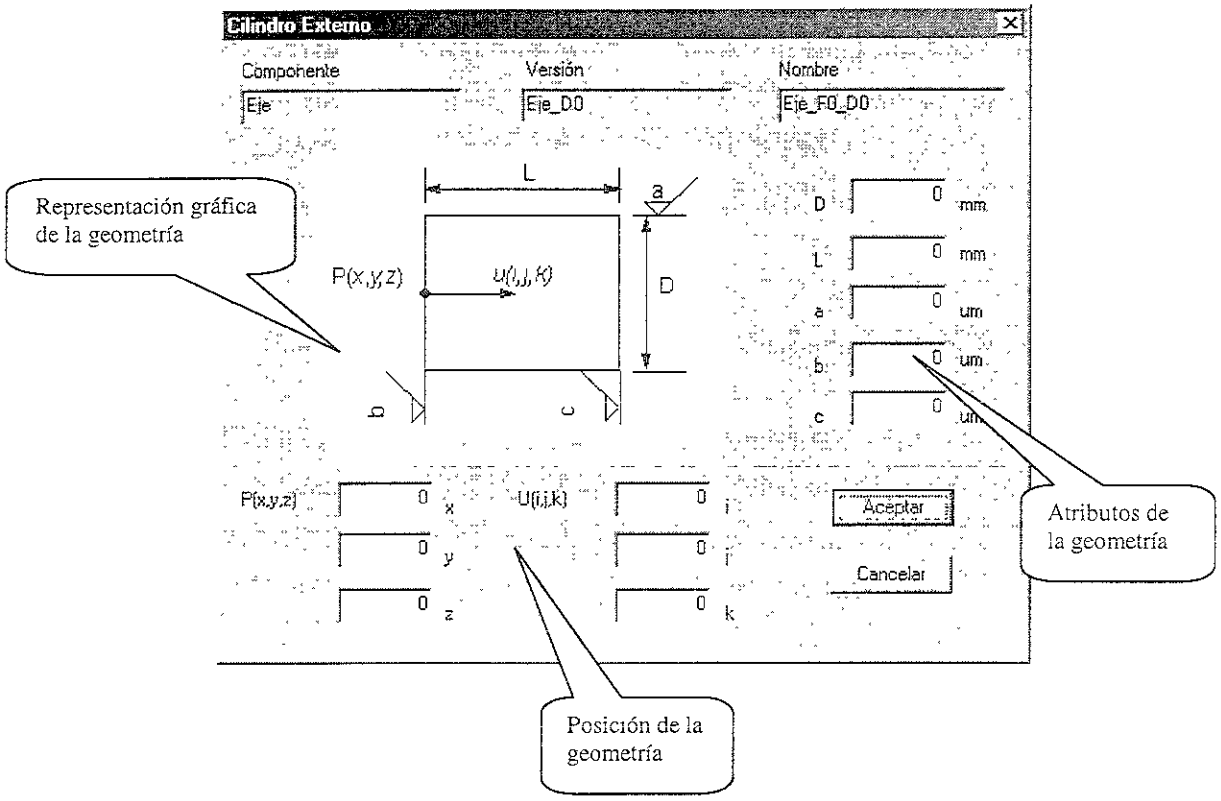


Figura 6.16 *Interfaz para definir dimensiones de una geometría*

Hasta aquí hemos representado la interfaz gráfica de la aplicación MET, en el siguiente capítulo se utilizará un caso de estudio para probar dicho programa.

CAPÍTULO 7

CASO DE ESTUDIO

7.1 Introducción

A continuación se presenta el caso de estudio utilizado en esta tesis para probar la aplicación MET desarrollada y para comprobar la utilidad del modelo de información del producto. En la primera sección del capítulo se especifican los objetivos del caso de estudio, en la siguiente se describe el componente para el caso de estudio. Y por último se hace uso del programa MET para representar el componente del caso de estudio.

7.2 Objetivos del caso de estudio

El propósito del caso de estudio es validar y probar el modelo del producto propuesto, por medio de la simulación de cómo se utilizaría el programa MET en una situación real en la industria. Con base en esto identificamos los siguientes objetivos específicos:

- Probar la aplicación del MP presentado en el Capítulo 4.
- Evaluar la utilidad y funcionalidad de la herramienta MET.
- Experimentar con información de un caso real.

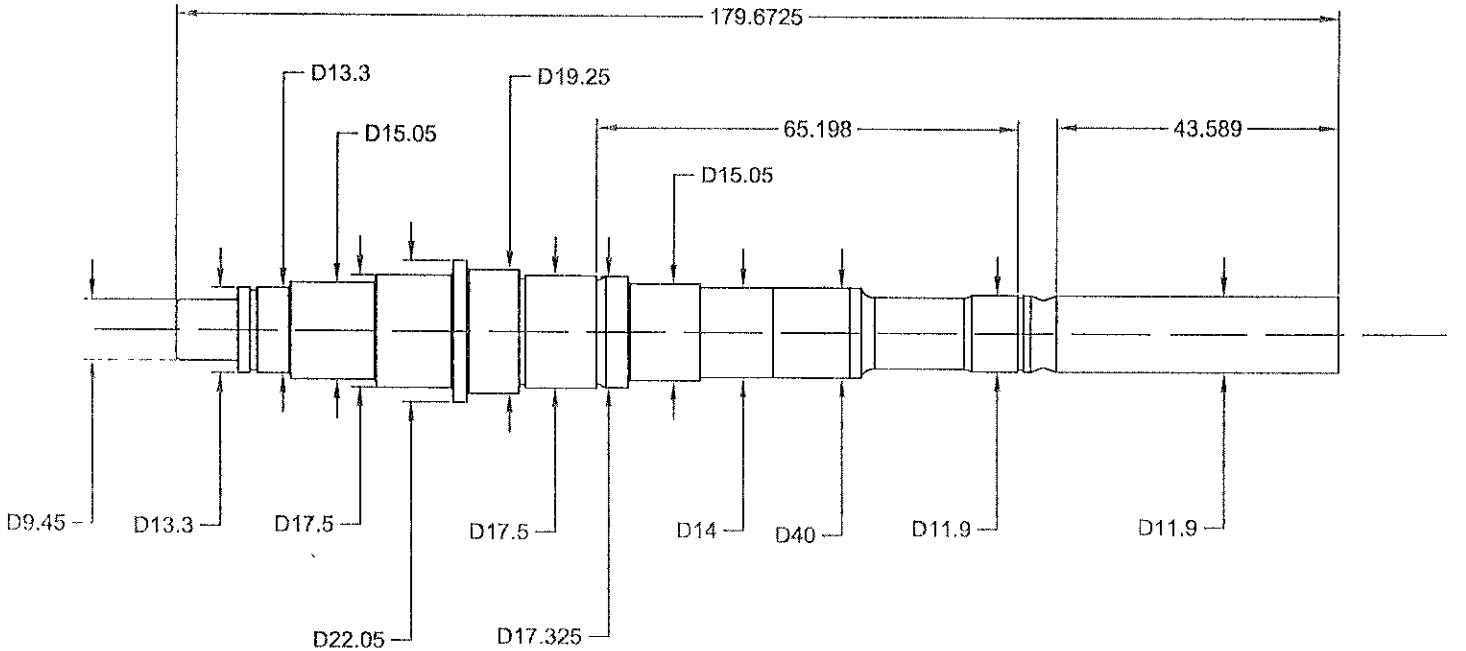
7.3 El componente para el caso de estudio

Para seleccionar el caso de estudio se buscó un componente que cumpliera con los siguientes requisitos:

a) Componente mecánico fabricado de un solo material.
b) Sección transversal cilíndrica, puede incluir características prismáticas como por ejemplo ranuras.
c) Manufacturado por un proceso de maquinado, en particular por torneado.
d) Información disponible sobre su diseño y manufactura.

El componente escogido para el caso de estudio fue el eje de potencia de una caja de transmisión de un colaborador industrial. En la figura 7.1 se muestra el componente para el caso de estudio.

NOTA	CAMB.	OBSERVACIONES	DIB.	REV.	FECHA.	SECTOR
	A					
	B					
	C					



MILIMETROS

TOL. NO ESP. LINEAL +/-
ANGULAR +/-

UNAM
FACULTAD DE INGENIERIA
DEPARTAMENTO DE INGENIERIA MECANICA

LIMAC
CAD-CAM-CAE

PLANO 1 DE 1

No. A4

DIB:

CENTRAL

TERM

DUREZA

TODOS LOS SIMBOLOS Y TOLERANCIAS ESTAN DE ACUERDO CON LAS NORMAS LIMAC

7.4 Uso del MET para representar el caso de estudio

Una vez escogido el caso de estudio se definieron las siguientes actividades como parte del ejercicio:

1. Descomponer el eje en formas básicas, como se plantea en la taxonomía característica. En la figura 7.2 se muestra un esquema de la representación de una parte del eje. Como se podrá apreciar el eje se va descomponiendo en características y con ellas se describe su estructura.
2. El siguiente paso es el definir las características de diseño o atributos físicos del componente como son las dimensiones, tolerancias, materiales, etc.
3. Introducir la información recopilada a la base de datos utilizando el programa MET.
4. Verificación de la integridad de la información en la base de datos utilizando *ObjectStore Inspector*.

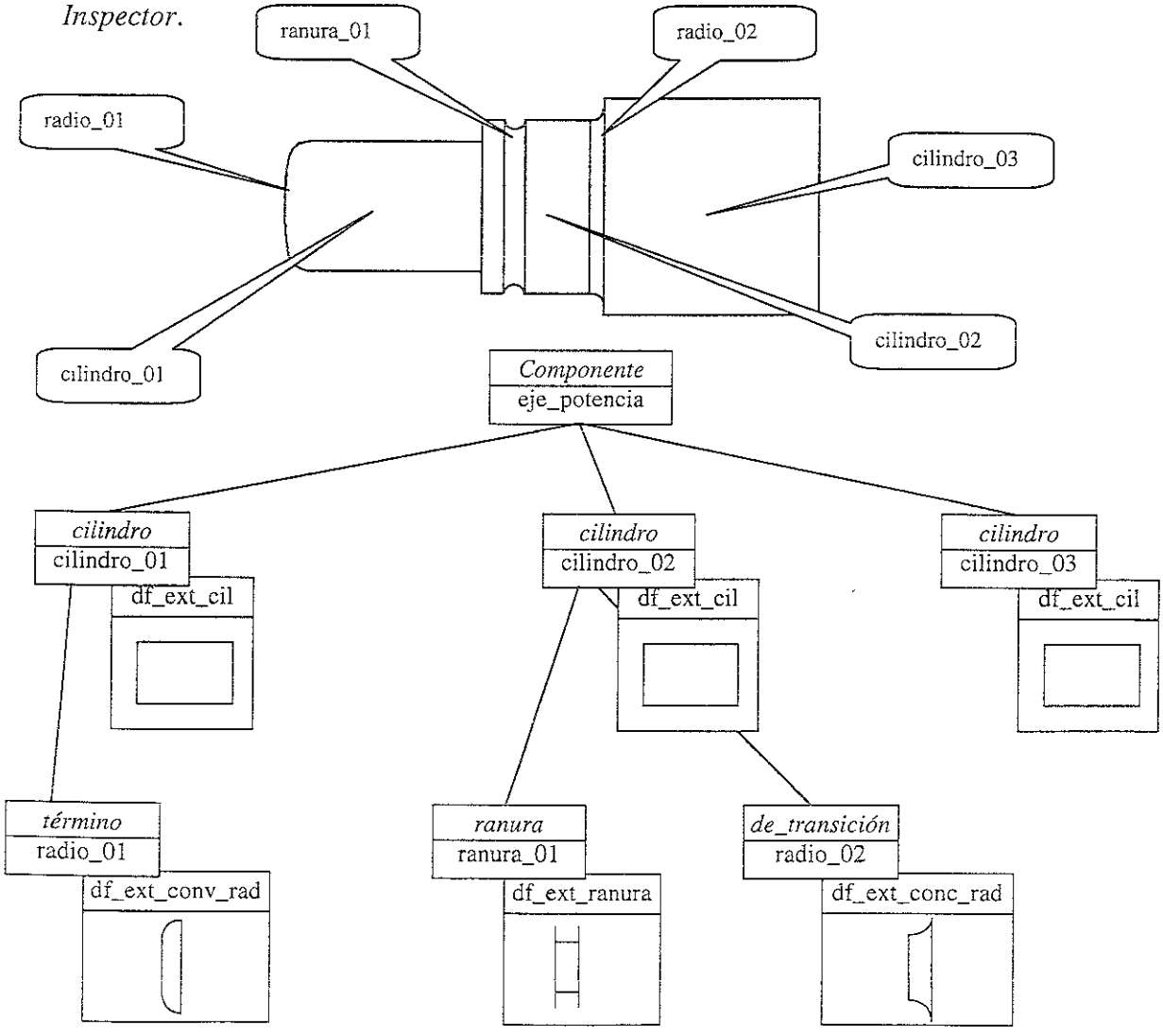


Figura 7.2 *Representación de una parte del eje de transmisión*

Como se mencionó en el Capítulo 6 para comenzar a utilizar el programa MET debemos crear un nuevo proyecto, es decir una nueva base de datos (archivo .db), la cuál puede contener varios componentes o ejes. A continuación se muestra la ruta para crear un nuevo proyecto (figura 7.3).

Proyecto/Nuevo/Escribir el nombre del proyecto/Crear/Ok

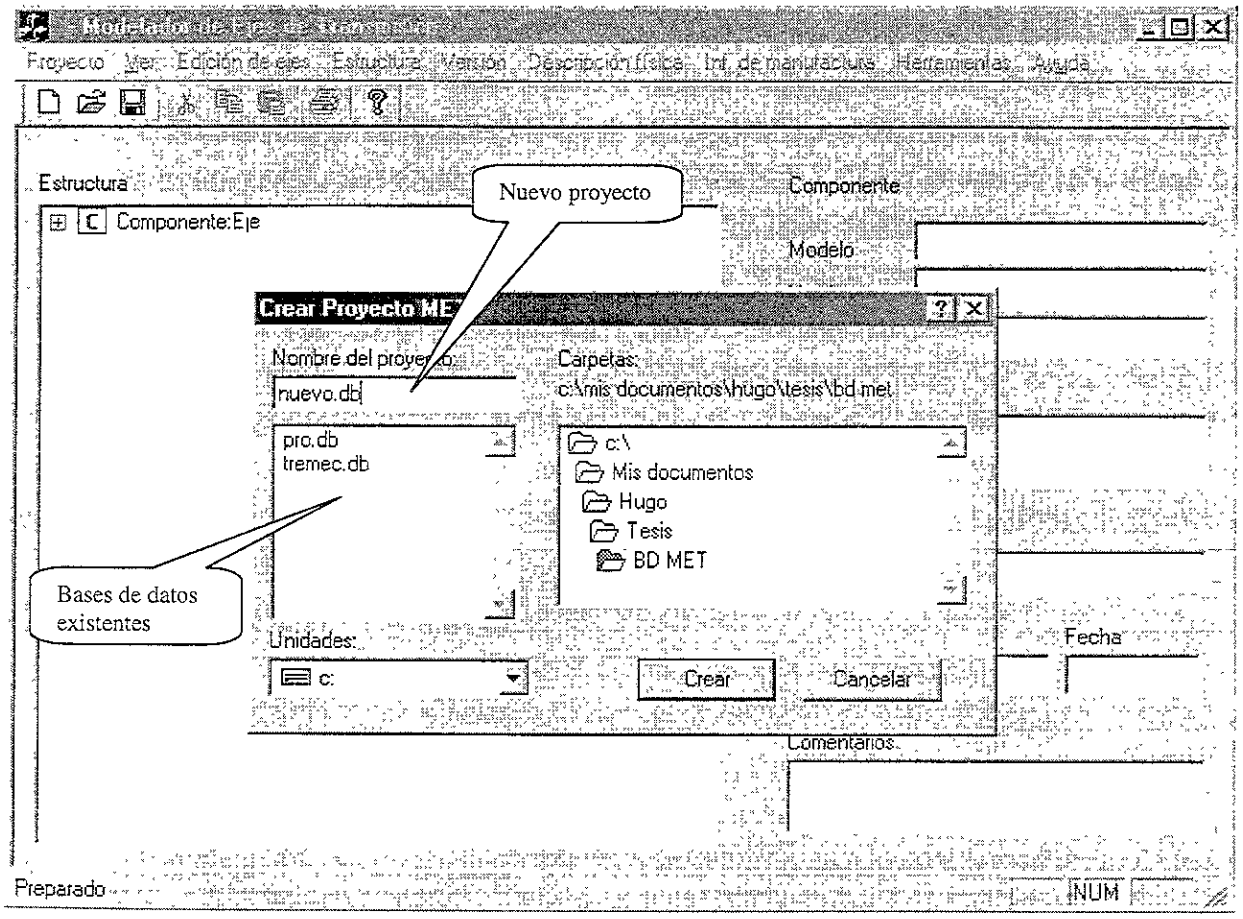


Figura 7.3 *Interfaz para crear un nuevo proyecto*

Una vez que se ha creado el proyecto lo siguiente es crear un componente, definir una versión o trabajar sobre la definición original, definir la estructura básica del eje (geometrías primarias y secundarias) y posteriormente definir sus dimensiones. En el capítulo 6 se muestra la interfaz gráfica y las instrucciones para llevar a cabo cada una de estas funciones.

Ya que se introdujo toda la información podemos utilizar una herramienta mas de la familia *ObjectStore* para revisar la estructura y la información contenida en la base de datos, ésta es *ObjectStore Inspector.*

A continuación se muestra en la figura 7.4 el contenido de la base de datos utilizando *Inspector*.

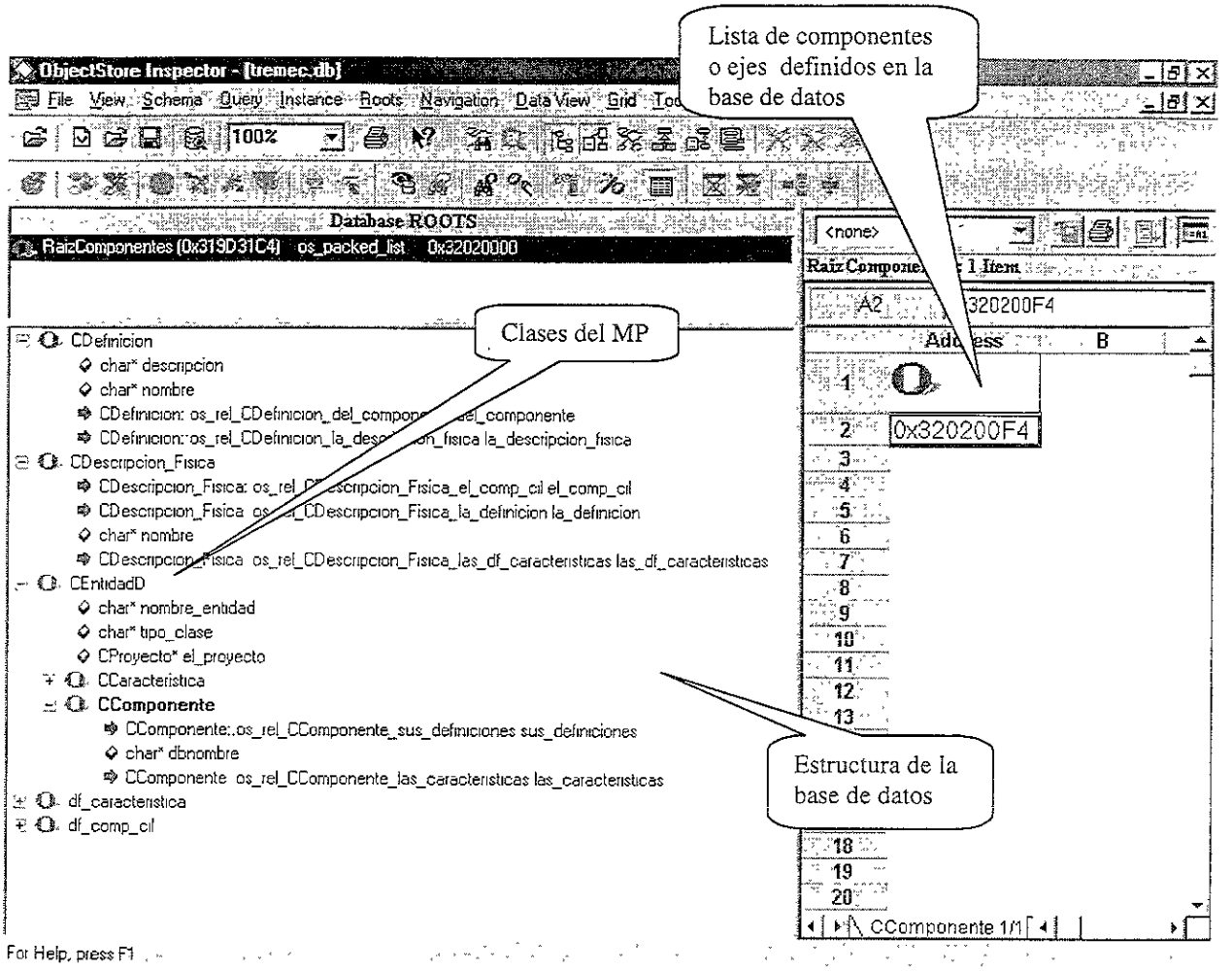


Figura 7.4 *Estructura de la base de datos en ObjectStore Inspector*

Como podemos ver en la figura anterior el programa nos muestra del lado derecho una lista de los componentes que tenemos en la base de datos (RaizComponentes) en este caso tenemos un solo eje definido por el momento. Del lado izquierdo se muestra la estructura o el árbol jerárquico de la base de datos, es decir como está organizada la información; se muestran las clases, sus atributos y sus funciones, así como las relaciones entre clases. Con esto verificamos que nuestra base de datos organiza la información como lo definimos en nuestro modelo (capítulo 4).

La figura 7.5 nos muestra el esquema físico de la base de datos. En la columna de la izquierda nos muestra los sectores utilizados, en la columna central nos da estadísticas sobre los elementos contenidos en la base de datos y en la columna de la derecha vemos una lista de las instancias contenidas en la base de datos.

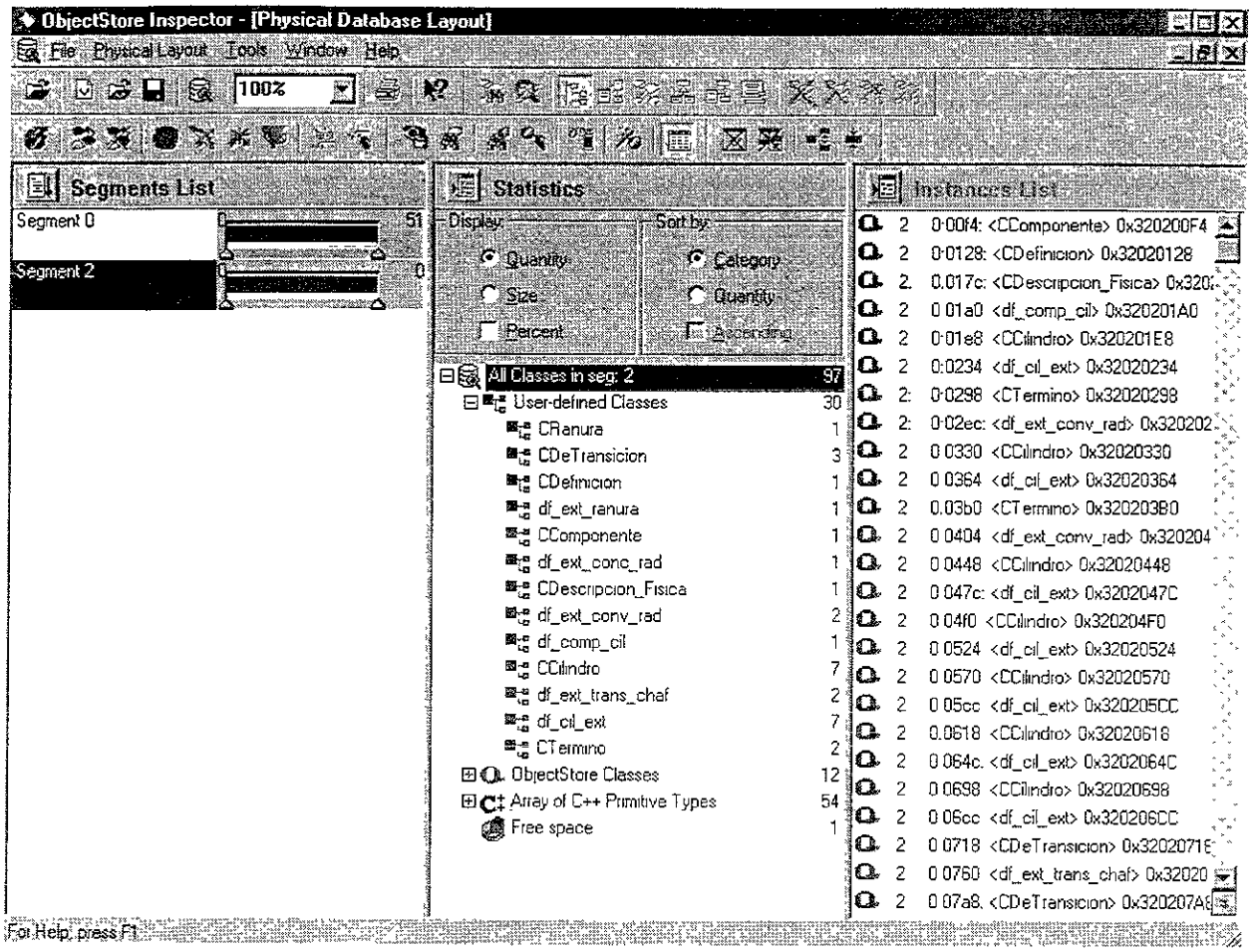


Figura 7.5 *Esquema físico de la base de datos en ObjectStore Inspector*

De esta forma podemos confirmar que la información está almacenada en nuestra base de datos y por lo tanto que el programa MET está funcionando correctamente. Al seleccionar un elemento en la base de datos podemos navegar manualmente por la información relacionada a dicho elemento y revisar que sean los valores introducidos desde el MET.

En la figura 7.6 se muestra un diagrama de navegación para recuperar información de la estructura de un eje y sus atributos físicos, junto con los diagramas de clases correspondientes. De esta forma podemos ver como la base de datos organiza la información como lo definimos en nuestro modelo (capítulo 4).

En la misma figura se muestran dos rutas de navegación, las cuales se representan también en el diagrama de clases del MP propuesto en el Capítulo 4.

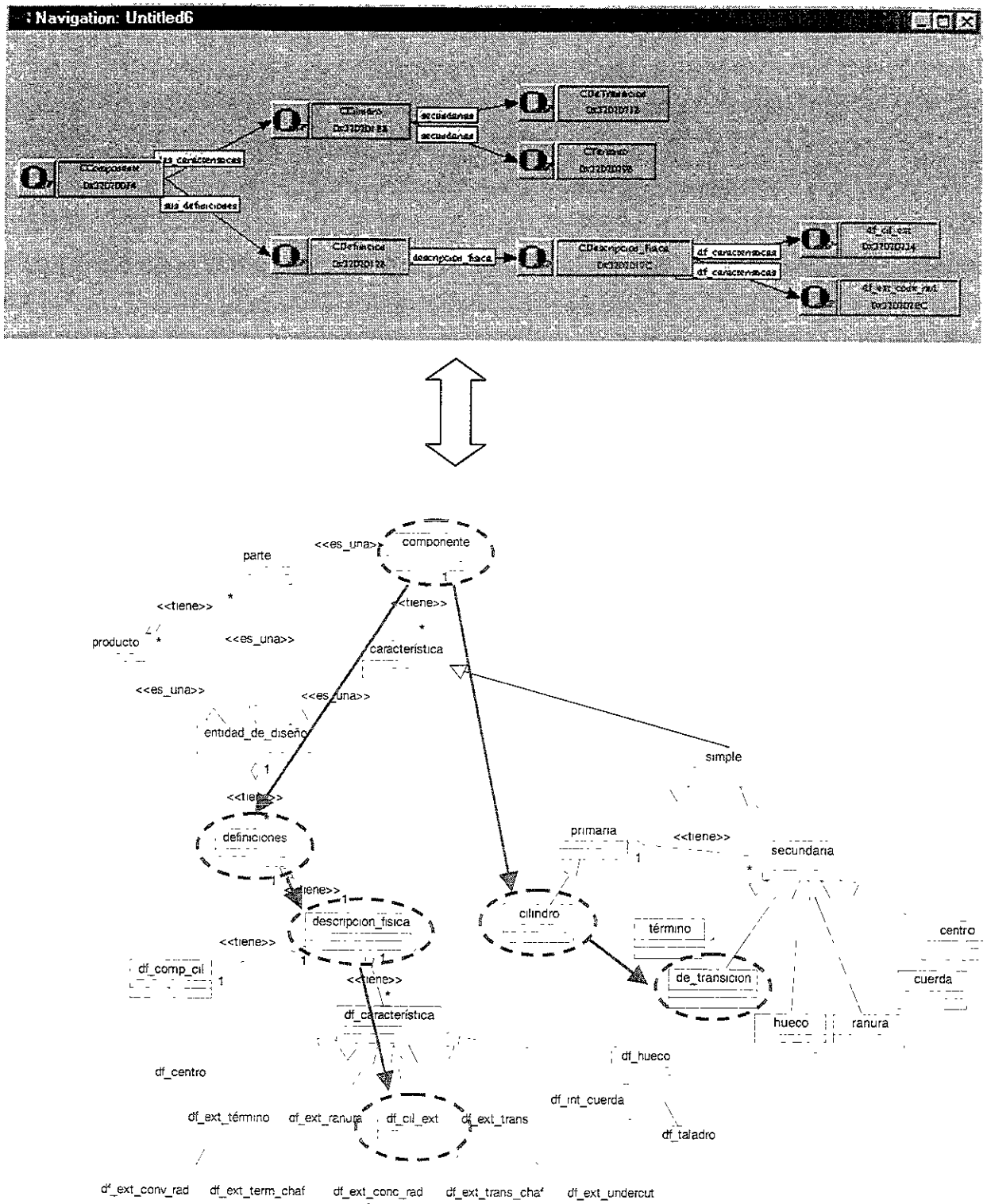


Figura 7.6 *Diagrama de navegación en la base de datos y el MP*

En la siguiente figura 7.7 se muestra un ejemplo de los valores de los atributos de un objeto de la clase *df_comp_cil*, correspondiente a los atributos generales de un componente cilíndrico. Y la misma información en la ventana del MET.

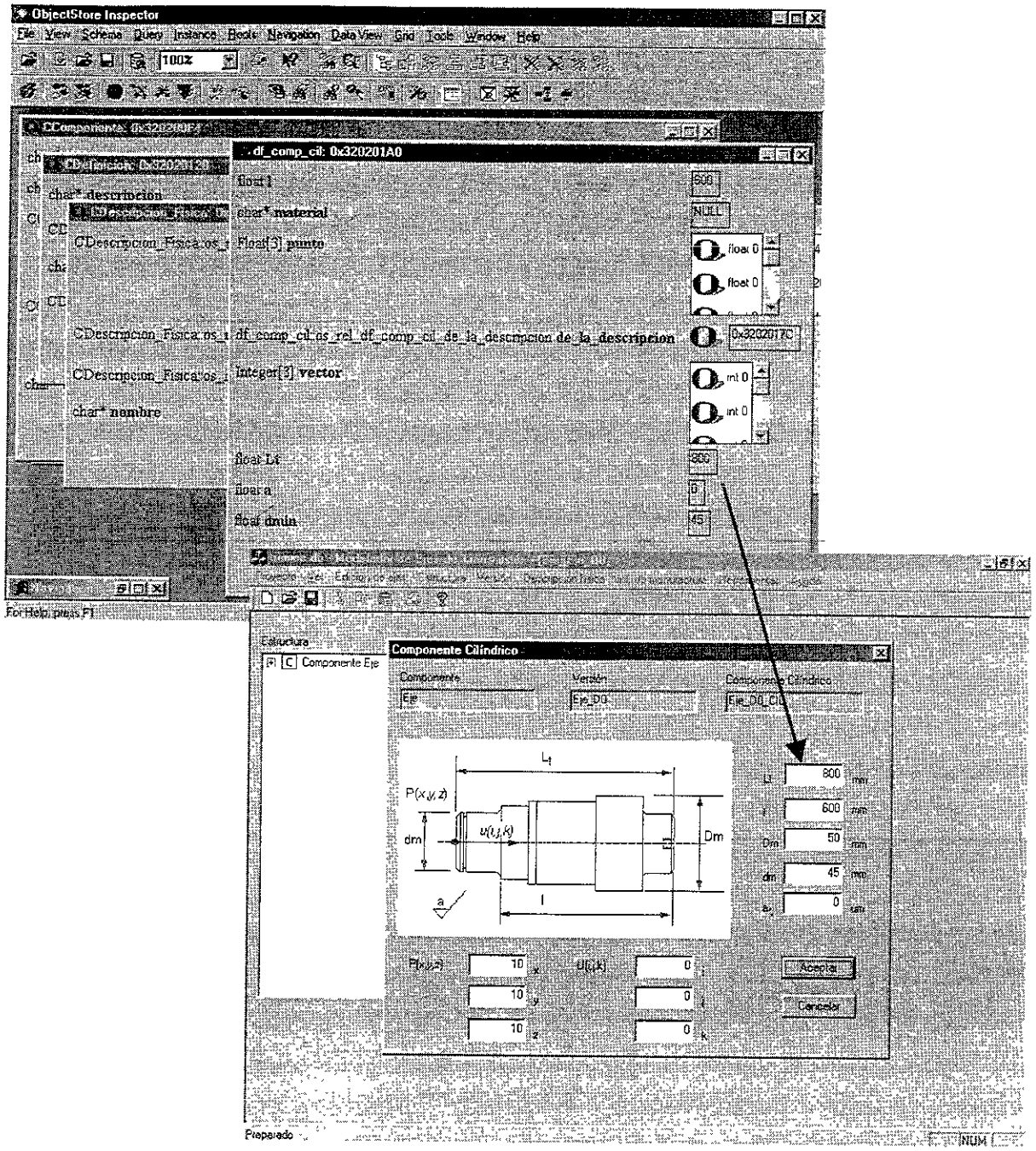


Figura 7.7 *Atributos del componente cilíndrico en la base de datos y en el MET*

De la misma forma podemos navegar por toda la información de la base de datos y revisar o modificar la información si así se desea. Así fue como se verificó la integridad de la información en la base de datos utilizando *ObjectStore Inspector*.

CAPÍTULO 8

CONCLUSIONES

En la actualidad se exploran sistemas CAE basados en modelos de información para asistir el diseño de productos siguiendo principios de ingeniería concurrente. SADET es un sistema de este tipo que auxilia el diseño de ejes de transmisión. Dos de sus elementos fundamentales han sido presentados en esta tesis, son: el MP y el MET, el primero es uno de los modelos de información con los cuales interactúan los demás programas integrados en el sistema, entre estos el MET, que es capaz de almacenar, recuperar y editar información de la base de datos, es decir la interfaz de usuario para poblar el MP. Cada programa realiza en forma independiente una tarea específica del diseño, pero a través de un ambiente de integración, pueden pedir y proporcionar información a los otros programas del sistema y modelos de información.

El MP propuesto, se basa en investigación madura en el área, es resultado de una revisión y simplificación del modelo propuesto por (Borja, 1997), el MP es capaz de representar componentes rotacionales fabricados por procesos de arranque de viruta, en específico solo se definieron las estructuras de clases necesarias para representar la información referente a la estructura y descripción física de componentes.

Para el modelado visual se utilizó el Lenguaje Unificado para Modelado o UML lo que nos permitió tener un modelo más comprehensivo y en un lenguaje estandar, también se utilizó el paradigma orientado a objeto, lo que hizo posible una representación flexible y distintiva de los componentes. La implementación del MP se hizo usando una base de datos orientada a objeto (*ObjectStore*), y *Visual* C++ en un ambiente común y amigable como es *Windows* 98. *ObjectStore* tiene una interfaz directa con C++ por lo cuál se decidió utilizar este lenguaje de programación para realizar la implementación.

También se creó el programa MET que fue desarrollado en el mismo ambiente de programación, esta aplicación tiene una interacción directa con la base de datos, toda la información es guardada en su forma nativa (clase en C++) lo cual nos facilita la manera de almacenar y recuperar la información, nos aumenta también la velocidad de interacción con la base de datos.

En la tesis se ha ilustrado la aplicación del MP y el MET para representar la estructura física de un eje de transmisión. El caso de estudio verificó la utilidad del modelo propuesto y la herramienta realizada. Se representó un componente real y se utilizó *ObjectStore Inspector* para verificar la integridad de la información en la base de datos.

Por otro lado, se encuentran en desarrollo los otros elementos de SADET. El siguiente paso es vincular el MP con un modelador de sólidos y juntar los demás elementos del sistema bajo un mismo ambiente de integración. Al tener concluido un sistema prototipo, el MP estará encargado de modelar ejes, asistir en el rediseño de componentes considerando reglas generales de maquinado y auxiliar en la planeación de la producción.

REFERENCIAS

Al-Ashaab, A. H. S., Ruiz, S. 1998. Using a PDM as a tool to support a Concurrent Engineering aplication in a Mexican Company. VITRO.

Al-Ashaab, A. H. S., Molina, A. , Gadh, R. 1998. The Application of the Manufacturing Model Over the INTERNET to Support the Integrated Product Development ITESM Campus Monterrey.

Al-Ashaab, Quiroz D. and Aparicio J. 1997. " Concurrent Engineering Activities in Mexico" *4th International Conference on Concurrent Enterprising*. The University of Nottingham, UK.

Bauert, F. 1989. *Composition of Product Modelling Systems Based on Commercial Software and their Influence on Industrial Companies, Design Methods and Education*, Proceedings of the Institution of Mechanical Engineers, International Conference, Engineering Design, Vol. 1, 22-25 August 1989, Harrogate International Centre, pp. 655-676.

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

CEIC 1998. Concurrent engineering Information Centre of the Mexican Society of Concurrent Engineering. *http://www.mor.itesm.mx/~smic/ceic*

Craig, L. 1997. *Applying UML and patterns: an introduction to object-oriented analysis and design*. Prentice Hall PTR. ISBN 0137488807.

ISO 10303, 1994. "*Industrial automation systems-product data representation and exchange*".

ITESM 1995. Procesamiento Distribuido Abierto en Apoyo al Desarrollo de Ingeniería Concurrente Asistida por Computadora. ITESM Campus Morelos.

Kimura, F., & Suzuki, H. 1989. *A CAD System for Efficient Product Design Based on Design Intent*, Annals of the CIRP Vol 38/1/1989.

Krause, F., Kimura, F., Kjelberg, T., Lu, S.C. 1993. *Product Modelling*, Annals of the CIRP Vol. 42/2/1993, pp.695-706.

Mantyla, M. 1989. *Directions for Research in Product Modeling*, Computer Applications in Production and Engineering, F. Kimura and A. Rolstadas Editors, Elsevier Science Publishers B.V. (North-Holland), IFIP, pp. 71-85.

McKay, A. 1993. *Product Model, The framework of the Product Data Model*, moses-core-pm-2 Issue 1, MOSES, Dept. of Mechanical Engineering, University of Leeds, Dept. of Manufacturing Engineering, Loughborough University.

Molina, A. 1995. *A Manufacturing Model to Support Data-Driven Applications for Design and Manufacture*, PhD Thesis, Department of Manufacturing Engineering, Loughborough University.

Molina, A., Al-Ashaab, A.H., Ellis, T.I.A., Young, R.I.M., Bell, R. 1995. *A Review of Computer Aided Simultaneous Engineering Systems, Reserch in Engineering Design*, Vol. 7, pp. 38-63.

MOSES (1992-1995). *Model Oriented Simultaneous Enfineering System*, The Home Page of the MOSES Project. *http://leve.leeds.ac.uk/www_moses/moses.html*.

Prasad, B. 1996. *Toward a Functional Design of a Concurrent Information Modeling System*, Computing Modeling and Simulation in engineering, Vol. 1, pp. 7-29.

Quatrani, T. 1998. Visual Modeling with Rational Rose and UML. Addison Wesley.

APÉNDICE A

EL MP PROPUESTO POR BORJA

Introducción

En este apéndice se presenta el modelo de información del producto (MP) propuesto por Borja (1997). Dicho modelo es parte de un sistema CAE que asiste el proceso de ingeniería inversa. Se comienza por definir el procedimiento de construcción del MP, a continuación se plantea su estructura principal y finalmente se discute la representación de las diferentes categorías de información requeridas para soportar el rediseño de productos. La implementación completa en UML (Diagramas de Clases) se reporta en los capítulos 4 y 6.

Metodología de Diseño

El proceso que se siguió para definir el MP incluye:

a) La definición de los requerimientos de información usando un modelo funcional. Para definir el esquema del modelo de información se modeló el proceso de rediseño de productos usando la técnica IDEF0 (*Integrated Definition for Function Modeling*) (ver Apéndice C). Considerando los elementos esenciales del proceso, se seleccionaron los nodos relevantes del modelo IDEF0. En el Apéndice C se muestran algunos de los principales nodos. Aspectos importantes del rediseño de componentes se describen en los nodos: Extracción y Análisis de Información, Rediseñar Entidad, Seleccionar Instalación y Especificar Información de Manufactura.

b) La generación de un esquema conceptual (categorías de información) y estructural, como un modelo semántico empleando el paradigma orientado a objetos. Este se implementó en UML (*Unified Modeling Language*). Los diagramas de clases se crearon de tal forma que fueran flexibles, simples de aprender y de usar. Además, estos debían ser eficientes en la comunicación de ideas.

c) La documentación detallada de los atributos, su dominio y restricción en un lenguaje de definición de información (ddl).

El modelo de información del producto organiza la información requerida para dicho modelo en seis categorías de información; esto como resultado del análisis de las limitaciones o restricciones de los principales nodos del modelo IDEF0. Las categorías son:

1. Estructura del producto. Modela la estructura física de una entidad de diseño —una entidad de diseño es un objeto físico o elemento, el cual corresponde a la solución desarrollada a través de actividades de diseño con el objetivo de satisfacer un conjunto de requerimientos —, representa la organización de sus elementos y como están asociados entre ellos. La estructura del producto es desarrollada por entidades descompuestas en sus partes constitutivas, hasta llegar al nivel deseado.

2. Especificaciones. Describe el conjunto de características que se desean de una entidad de diseño. Esta información restringe las posibles soluciones y provee un criterio para diferenciar entre buenas y malas alternativas durante el proceso de rediseño. Esta categoría incluye la declaración del problema, una descripción general de los clientes, sus requerimientos y las especificaciones de ingeniería.

3. Conceptos. Incluye la información que describe la funcionalidad de los productos y su estructura funcional.

4. Características de Diseño. Modela la forma, dimensiones, tolerancias, acabado superficial, materiales, entre otras propiedades de las entidades a través de todo su ciclo de vida.

5. Información de Manufactura. Modela la definición de productos desde el punto de vista de sus aspectos de manufactura. La información de manufactura requerida por el MP se puede resumir como: secuencia y descripción de operaciones, parámetros específicos para cada uno de ellos y referencias a instancias del modelo de información de manufactura.

6. Propiedades. Representa las características cuantificables después de la manufactura del producto.

Estas categorías de información fueron usadas como base para definir el marco de referencia y contenidos del MP.

Estructura del Modelo de Información del Producto

La estructura y los esquemas del modelo de información fueron definidos considerando conceptos genéricos los cuales pueden ser aplicados a una gran variedad de productos electro-mecánicos. Sin embrago, las especificaciones de las clases están definidas considerando componentes rotacionales manufacturados por procesos de maquinado, en particular torneado.

La figura A.1 presenta el diagrama de clases en UML (ver Apéndice B) del marco de referencia del MP. Éste se basa en el paradigma orientado a objetos y provee el contexto para las estructuras que representan las categorías de información enunciadas anteriormente. El marco de referencia se centra en la estructura física del producto y en las etapas de su ciclo de vida. Está influenciado por el MP desarrollado por McKay (1993).

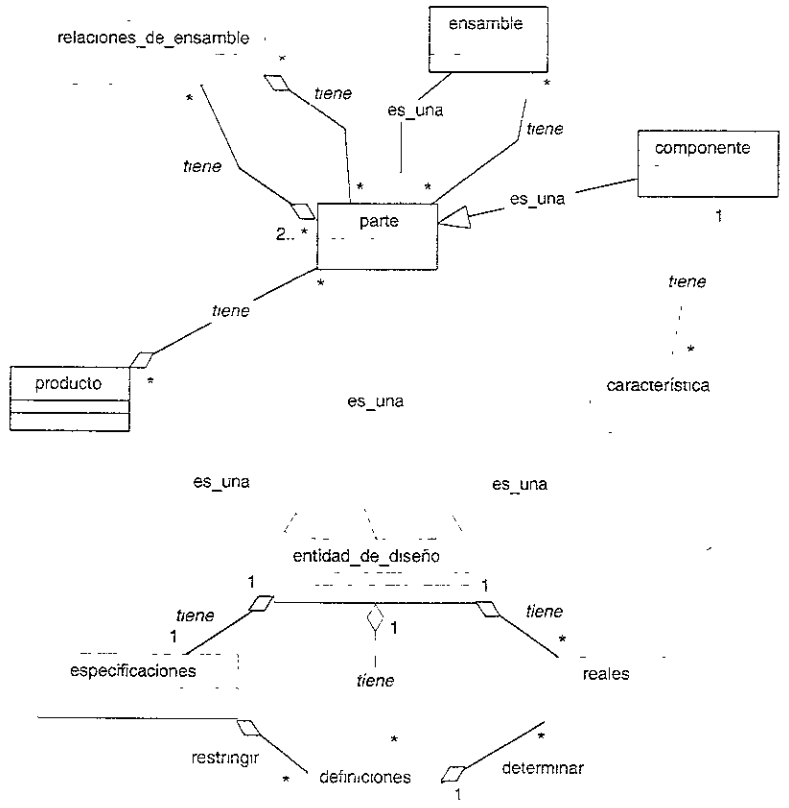


Figura A.1 *Marco de referencia del MP*

Cada elemento de la estructura del MP es una metaclase, y por lo tanto, todas ellas son clases superiores de taxonomías distintivas.

La siguiente tabla muestra todas las clases involucradas en el marco de referencia del MP. Se indica a que categoría de información pertenecen, y se da una breve especificación de cada una.

Categoría	Clase	Especificación
Estructura del producto	*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	*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.
Conceptos	*definición_funcional*	La información contenida en esta clase representa la funcionalidad de los productos.
Características de diseño	*descripción_física*	Contiene información sobre atributos físicos del producto, como: dimensiones, tolerancias, materiales, etc.
Información de manufactura	*información_de_manufactura*	Contiene información sobre la materia prima, el costo total de manufactura, el número de operaciones y el tiempo requerido para ejecutarlas.
Propiedades	*reales*	Representa información de la entidad real manufacturada.

Tabla *Clases del marco de referencia del MP*

Modelado de la categoría de información "Estructura del Producto"

El núcleo del MP es la clase *entidad_de_diseño*. A través de las subclases y asociaciones de *entidad_de_diseño* (parte superior del MP), se puede representar la categoría de información "Estructura del Producto".

Productos, *partes* y *características* son entidades de diseño particulares. Las estructuras de entidades están comúnmente representadas usando diagramas de árbol jerárquicos, los cuales pueden modelar partes o productos completos: un *producto* puede ser subdividido en un gran número de *partes*, una *parte* puede ser un *componente* o un *ensamble*; éste a su vez puede ser descompuesto en subensambles. Los *componentes* están formados por *características*, que no pueden estar solas físicamente; siempre deben asociarse con otras para construir componentes o características mas complejas.

La taxonomía *característica* para representar partes rotacionales maquinadas

Una gran variedad de taxonomías pueden ser asociadas a los elementos del marco de referencia del MP. En la figura A.2 se presenta la taxonomía *característica* para representar partes rotacionales producidas por operaciones de maquinado.

Una *característica* puede ser *simple* ó *compuesta*, la segunda está formada por dos o mas simples. Las características *simples* son a su vez subdivididas en *primarias* las cuales constituyen la estructura principal de componentes, y *secundarias*, las cuales se añaden a las *primarias* para satisfacer requerimientos funcionales o de manufactura. Por lo tanto, los componentes están formados por una serie de características primarias, asociadas una con otra, las cuales tienen características secundarias.

Considerando que el lector está familiarizado con la mayoría de los nombres seleccionados para las clases de la figura A.2, solo dos serán explicadas: 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 otra característica primaria.

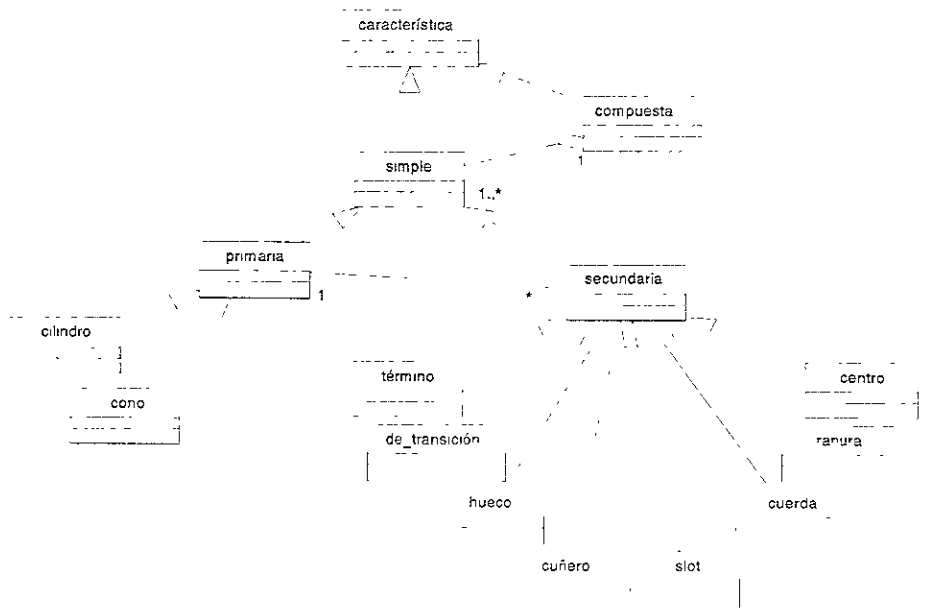


Figura A.2 *Taxonomía característica para representar partes rotacionales maquinadas*

El método para definir entidades de diseño

Como se mencionó anteriormente, el MP describe entidades considerando tres etapas de su ciclo de vida: como especificado, como diseñado y planeado para manufactura y como manufacturado. La segunda etapa está representada por la clase *definiciones* de la estructura del modelo de información y por sus clases asociadas. Aún cuando diferentes tipos de definiciones pueden ser usadas, incluyendo dibujos, modelos geométricos sólidos, representaciones de frontera, patrones de esfuerzo, diagramas de bloques, etc., la definición de productos se realiza desde tres perspectivas de diseño: su funcionalidad (conceptos), su aspecto físico (características de diseño) y sus aspectos de manufactura (información de manufactura). Estas perspectivas corresponden a tres categorías de información requeridas por el MP y son modeladas usando tres clases como se muestra en la figura A.3. Las clases son *definición_funcional*, *descripción_física* e *información_de_manufactura*.

La cardinalidad uno a uno, la cuál asocia la clase *definiciones* con las otras, implica que una combinación particular de instancias de las dos subclases determina una definición específica de la entidad de diseño. Consecuentemente, si cualquiera de éstos cambia, se creará una definición

diferente de la misma entidad de diseño. De esta forma se pueden representar soluciones alternativas o versiones de diseño diferentes de la misma entidad.

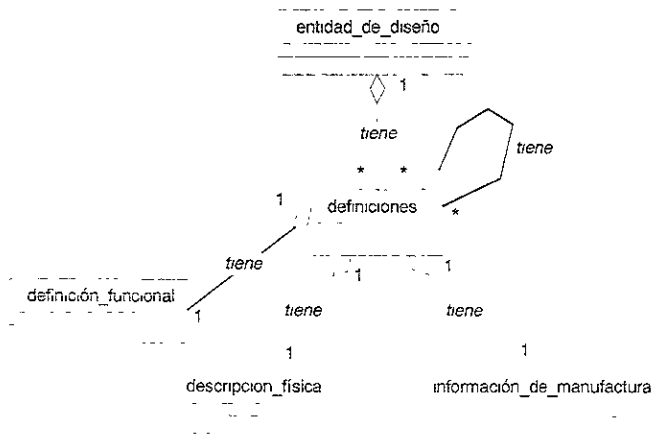


Figura A.3 *Clases de definición del MP*

Para terminar de definir la entidad de diseño, la cuál está compuesta por otras, se deben asociar definiciones específicas de cada entidad. Esto se representa por la asociación *tiene* de la clase *definiciones* con ella misma.

Modelado de la categoría de información "Características de Diseño"

Las actividades principales de una persona mientras se encuentra diseñando una entidad de diseño son las de definir sus características. Estas características son: estructura, forma, dimensiones, tolerancias, acabado superficial, materiales y proceso de manufactura. Las características de diseño determinan la funcionalidad y todas las propiedades de entidades a través de las etapas de su ciclo de vida.

En el MP, las características de diseño están modeladas dentro de dos diferentes categorías de información. La estructura del producto ha sido discutida anteriormente y el proceso de manufactura será analizado como parte de la información de manufactura. La forma, dimensiones, tolerancias, acabado superficial y materiales son referidos como atributos físicos, estos son modelados en la clase *descripción_física*, ésta es una metaclase que está asociada a diferentes taxonomías de clases de descripción física las cuales representan una gran variedad de

productos, partes y características. Los atributos físicos son modelados con la estructura de información que se muestra en la figura A.4.

El propósito de las clases *vector* y *punto* es el de localizar la posición y orientación de la entidad en el espacio. La clase *material* representa las propiedades de los materiales de la entidad.

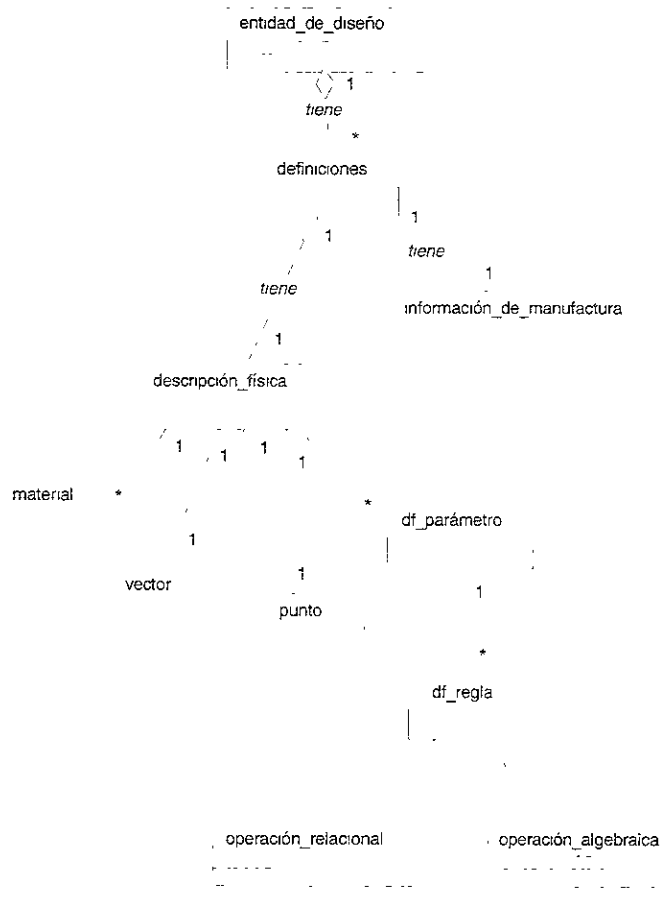


Figura A.4 *La clase descripción_física del MIP.*

El método para modelar atributos físicos, excepto materiales, emplea una combinación de representaciones visuales (archivos) de 2D y/o 3D, los cuales definen parámetros con una notación comúnmente utilizada en dibujos mecánicos, e información detallada acerca de los parámetros contenidos dentro del MP. La representación visual no está considerada como parte del MP, pero se mantiene una referencia hacia ella. Los parámetros tienen un nombre, un valor, una tolerancia, unidades, una regla y una referencia, y son modelados dentro de la clase *df_parametros*, los parámetros se pueden referir a cualquier tipo de atributo cuantitativo como dimensión o acabado superficial.

Las reglas especifican el dominio o limitaciones de los parámetros, de esta manera se asegura que la entidad de diseño desempeñe sus funciones o mantenga su forma básica. Se han definido dos tipos de reglas: operaciones relaciónales y operaciones algebraicas. Las operaciones relaciónales se refieren a las relaciones de igualdad, mayor que y menor que. Las operaciones algebraicas se refieren a las cuatro operaciones matemáticas elementales: adición, substracción, división y multiplicación. Los operandos de las reglas pueden ser cualquier parámetro definido de cualquier entidad de diseño, o un valor específico.

La referencia de un parámetro es la fuente de la regla y/o el valor del parámetro. Ejemplo de referencias son un estándar, una publicación, el nombre del diseñador. La combinación de representaciones visuales y parámetros hacen posible una descripción flexible, comprensiva y estructurada de los atributos físicos de productos y partes cuando son usados en diferentes niveles de la estructura del producto. Información esencial es incluida en los niveles superiores de la estructura (productos y partes) y la información detallada de cualquier entidad está disponible en los niveles inferiores (características).

Modelado de la categoría de información "Información de Manufactura"

El objetivo del MP es el de proveer ambientes de aplicación con la información requerida para garantizar las actividades de diseño para manufactura, como por ejemplo soportar la definición de la descripción física de entidades de tal forma que éstas puedan ser manufacturadas económicamente usando las instalaciones disponibles y para proveer soporte en la propuesta de posibles rutas de proceso. Una ruta de proceso incluye los procesos de manufactura, recursos y cambios geométricos de una entidad de diseño a través de operaciones las cuales transforman esta de su estado inicial al producto deseado o componente. La representación de una ruta de proceso requiere la definición del estado inicial (materia prima), el estado deseado (producto deseado) y los estados intermedios de una parte o producto en todas sus operaciones de manufactura, junto con el proceso de producción, recursos de manufactura y parámetros de operación.

La información de manufactura requerida para el MP se puede resumir como: secuencia y descripción de operaciones, parámetros específicos para cada uno de ellos, y referencias a instancias del modelo de información de manufactura. Con esto se asume que la clase

descripción_física del modelo de información es capaz de almacenar la información geométrica asociada a los estados inicial, final e intermedio del producto.

El diagrama de clases del modelo de información de manufactura se presenta en la figura A.5. La información de manufactura es modelada usando cinco clases: *información_de_manufactura*, *operación*, *descripción_operación*, *parámetro_operación* y *recurso_de_manufactura*. Además, hay una asociación entre la clase *operación* y la clase *entidad_de_diseño*.

La clase *información_de_manufactura*, contiene la materia prima, el costo total de manufactura, el número de operaciones y el tiempo requerido para ejecutarlas. Como podemos ver en la figura A.5, información de manufactura tiene operaciones asociadas a ella. Una *operación* es una parte básica del proceso de manufactura, y se define por la combinación de fuentes y parámetros particulares de manufactura. Por ejemplo, una operación de maquinado se caracteriza por una máquina herramienta, una herramienta de corte y parámetros de maquinado.

Las operaciones son modeladas usando cuatro clases. La clase *operación* incluye:

a) Un número de operación, el cuál relaciona su secuencia en la ruta de proceso.
b) Un proceso de producción, asociado a una instancia del modelo de información de manufactura (MM).
c) El tiempo y costo de la operación.
d) Una referencia a un archivo de instrucción el cuál contiene información útil para la manufactura, por ejemplo, un archivo de instrucción puede ser un programa CN.

La clase *operación* también incluye una asociación *tiene* con sí misma para posibilitar la representación de una sub-operación, por ejemplo la operación de torneado tiene las sub-operaciones de desbastado y acabado. La clase *operación* es el elemento superior de la taxonomía que representa diferentes tipos de operaciones (ej. maquinado, ensamble, inspección, etc.). Cada tipo de operación puede requerir una estructura de información particular. La clase *operación* descrita se refiere al proceso de maquinado pero puede ser aplicada a otras operaciones con requerimientos de información similares.

La información específica para aplicar el proceso de producción para manufacturar las partes está contenida dentro de la clase *descripción_operación*. Esta clase tiene *parámetros_operación* los cuales están compuestos por un nombre, una magnitud y unidades. Estos parámetros pueden representar parámetros físicos (ej. diámetro, longitud) y parámetros de manufactura (ej. velocidad, alimentación).

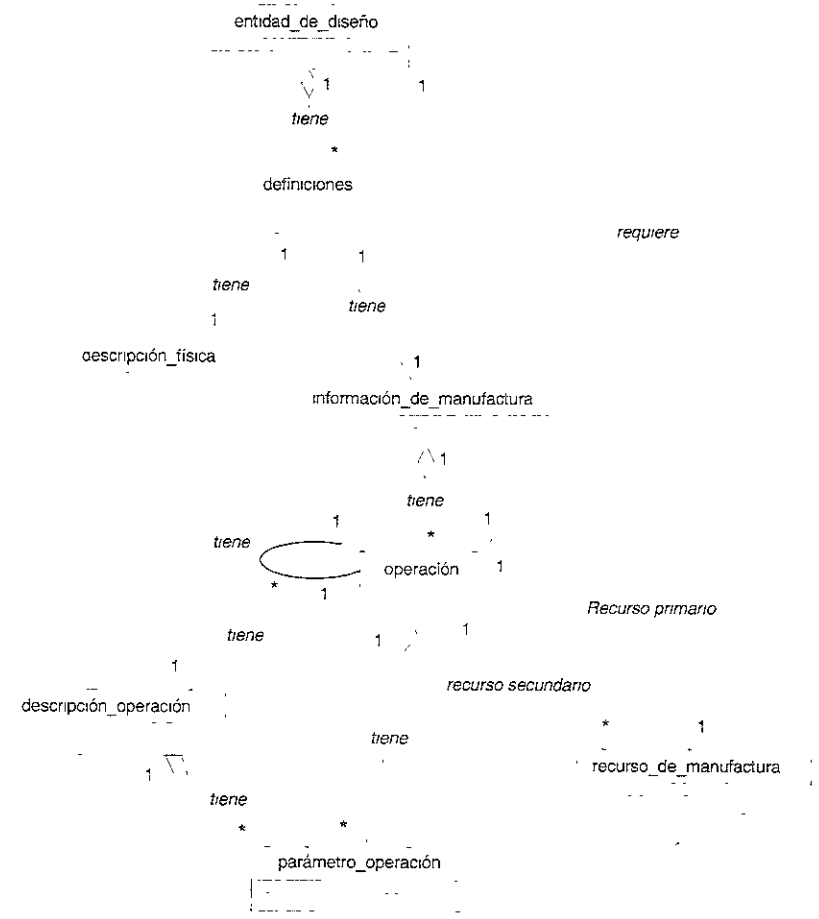


Figura A.5 *La clase información_de_manufactura del MP*

La última clase utilizada para representar operaciones es *recurso_de_manufactura*. Ésta es capaz de almacenar identificadores de recursos (ej. máquinas, herramientas) capturados dentro de modelos de manufactura. Esta clase tiene dos asociaciones con la clase *operación*: *recurso primario* se refiere a la capacidad de los recursos para desempeñar el proceso, por ejemplo maquinaria de producción; *recurso secundario* se refiere a los recursos que deben estar presentes en un proceso, por ejemplo herramientas de producción.

APÉNDICE B

LENGUAJE UNIFICADO PARA MODELADO

Introducción

El Lenguaje Unificado para Modelado o UML, es un lenguaje orientado a objetos de tercera generación, que adapta y extiende lo publicado en los trabajos de Grady Booch, Jim Rumbaugh, e Ivar Jacobson [Booch94, OMT91, OOSE92] aparte de contener mejoras y sugerencias hechas por decenas de personas. El UML fue presentado por el *Object Management Group*, con la intención de que se convirtiera en un lenguaje de modelado estandar para el desarrollo orientado a objetos. Dado que el UML es aplicable para modelar cualquier tipo de sistemas, lo mismo se usa en sistemas de tiempo real, que en cliente/servidor, y otras clases de aplicaciones estandar. Este lenguaje provee un rico conjunto de notaciones y premisas que pueden ser soportadas por la mayoría de los proveedores de herramientas para desarrollo de software.

Método Orientado a Objetos para el Modelado de Sistemas

Los métodos estructurados separan la información de las funciones, disminuyendo así su cohesión. Los métodos Orientados a Objetos tienen un acercamiento diferente, estos unifican la información y las funciones que operan sobre ésta en componentes del software llamados objetos. En el mundo de tiempo-real, los objetos son modelos de cosas como sensores, motores, interfaces de comunicación, etc.

Ya que alguno de los objetos en un sistema puede ser el duplicado de algún otro, llega a ser redundante y tedioso especificar la información y las funciones para cada uno. Los métodos Orientados a Objetos usan la notación de *clase* para capturar en un solo lugar la estructura (por ejemplo, los campos de información comunes) y el comportamiento de tales objetos.

Una *clase* es como la estructura declarada en lenguaje C, que contiene campos de información y de funciones. Las estructuras son definiciones de grupos de campos que están estrechamente relacionados. Todas las variables del mismo tipo de estructura se parecen en cuanto que todas ellas tienen los mismos campos. Correspondientemente, cada objeto tiene una definición que es

prescrita por su clase. Pero a diferencia de estructuras en C las cuales generalmente definen solo campos de información, una clase de objetos define ambos, su información y sus funciones en una sola estructura. Usando la terminología del UML, la parte de información de un objeto es definida por un conjunto de atributos, y la parte funcional del objeto es definida por un conjunto de operaciones.

Mientras cada objeto de una clase dada tiene exactamente la misma estructura y comportamiento, cada objeto es único en cuanto que los cambios que se hacen en uno no afectan automáticamente a los otros.

El estructurar información y funciones dentro de clases es fundamental para el modelado con un método Orientado a Objetos. Otra de las principales características de la orientación a objetos es el encapsulamiento. Éste establece principalmente que la parte de información de un objeto es accesible sólo a través de funciones definidas por la clase de objetos. El encapsulamiento hace que los objetos sean mas confiables y seguros.

Modelando con Diagramas UML

El UML puede modelar varios tipos de información. La facilidad de describir los varios aspectos de modelado del UML es a través de la notación definida por sus diferentes tipos de diagramas.

El UML distingue entre las notaciones de modelo y diagrama. Un *modelo* contiene todos los elementos de información fundamentales acerca de un sistema considerado y es independiente de cómo se presenten visualmente estos elementos. Un *diagrama* es una visualización particular de ciertos tipos de elementos de un modelo y generalmente exponen sólo un subconjunto de esos elementos, información detallada. Un elemento de un modelo dado puede existir en múltiples diagramas, pero hay una sola definición de ese elemento en el modelo fundamental.

Los diversos elementos notacionales que presenta el UML pretenden ser un lenguaje común para el modelado de cualquier sistema.

Los conceptos se agrupan por tipo de diagrama:

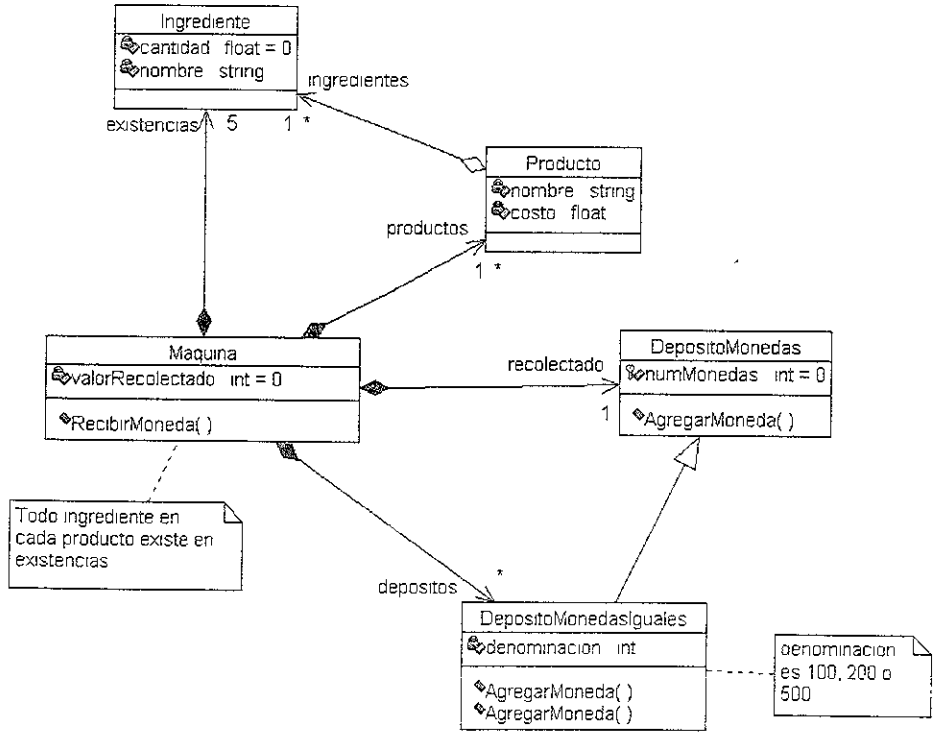
1. Diagrama de Estructura Estática
2. Diagrama de Casos de Uso
3. Diagrama de Secuencia
4. Diagrama de Colaboración
5. Diagrama de Estados
6. Diagrama de Actividades
7. Diagrama de Implementación

Cada tipo de diagrama captura una perspectiva o punto de vista diferente del modelo. Para algunos de los tipos de diagramas, existen múltiples diagramas de ese mismo tipo.

Diagrama de estructura estática

Un diagrama de estructura estática muestra el conjunto de clases y objetos importantes que forman parte de un sistema, junto con las relaciones existentes entre estos. Muestra de una manera estática la estructura de información del sistema y la visibilidad que tiene cada una de las clases, dada por sus relaciones con las demás en el modelo.

Supongamos el modelado de una máquina que vende X producto, su diagrama de estructura estática inicial podría ser:



Clase

Representada por un rectángulo con tres divisiones internas, son los elementos fundamentales del diagrama. Una clase describe un conjunto de objetos con características y comportamiento idéntico. Los tres compartimientos alojan el nombre de la clase, sus atributos y sus mensajes, respectivamente. En el ejemplo se encuentran las clases *Ingrediente*, *Producto*, *Máquina*, *DepósitoMonedas* y *DepósitoMonedasIguales*.

Atributos

Identifican las características propias de cada clase. Generalmente son de tipo simple, ya que los atributos de tipo compuesto se representan mediante asociaciones de composición con otras clases. La sintaxis de un atributo es:

visibility name : type-expression = initial-value { property-string }

Donde *visibility* es uno de los siguientes:

- *visibilidad pública*
- *visibilidad protegida*
- *visibilidad privada*

type-expression es el tipo del atributo con nombre *name*. Puede especificarse un valor inicial y un conjunto de propiedades del atributo.

En el caso del ejemplo, la clase *Ingrediente* tiene dos atributos: uno denominado *cantidad*, de tipo float y con valor inicial 0; y el atributo *nombre* de tipo string sin valor inicial.

Operación

El conjunto de operaciones describen el comportamiento de los objetos de una clase. La sintaxis de una operación en UML es:

visibility name (parameter-list) : return-type-expression { property-string }

Cada uno de los parámetros en *parameter-list* se denota igual que un atributo. Los demás elementos son los mismos que en la notación de un atributo.

Asociación

Una asociación en general es una línea que une dos o más símbolos. Pueden tener varios tipos de adornos, que definen su semántica y características. Los tipos de asociaciones entre clases presentes en un diagrama estático son:

1. Asociación binaria
2. Asociación n-aria
3. Composición
4. Generalización
5. Refinamiento

Cada asociación puede presentar algunos elementos adicionales que dan detalle a la relación, como son:

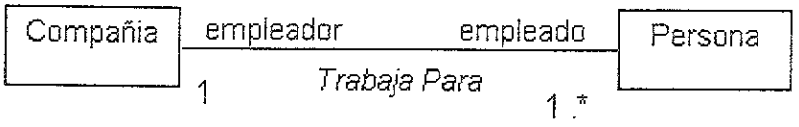
Rol: Identificado como un nombre en los finales de la línea, describe la semántica de la relación en el sentido indicado. Por ejemplo, la asociación de composición entre *Máquina* e *Ingrediente* recibe el nombre de *existencias*, como rol en ese sentido.

Multiplicidad: Describe la cardinalidad de la relación. En el ejemplo se utilizan **1**, **1** ..*, **5**, *, como indicadores de multiplicidad.

Asociación binaria

Se identifica como una línea sólida que une dos clases. Representa una relación de algún tipo entre las dos clases, no muy fuerte (es decir, no se exige dependencia existencial ni encapsulamiento).

El posible ejemplo es la relación entre una compañía y sus empleados



en este caso la relación recibe el nombre genérico *Trabaja Para*, la compañía tiene uno o más instancias de la clase *Persona* denominadas *empleado* y cada empleado conoce su *empleador* (en este caso).

Composición

Es una asociación fuerte, que implica tres cosas:

- Dependencia existencial. El elemento dependiente desaparece al destruirse el que lo contiene y, si es de cardinalidad 1, es creado al mismo tiempo.
- Hay una pertenencia fuerte. Se puede decir que el objeto contenido es parte constitutiva y vital del que lo contiene.
- Los objetos contenidos no son compartidos, esto es, no hacen parte del estado de otro objeto.

Se denota dibujando un rombo relleno del lado de la clase que contiene a la otra en la relación. El ejemplo inicial se presentan varios ejemplos de relaciones de composición entre *Máquina* y *Producto*, *Máquina* y *DepositoMonedas* y *Máquina* y *DepositoMonedasIguales*.

Existe también una relación de composición menos fuerte (no se exige dependencia existencial) que es denotada por una un rombo sin rellenar en uno de los extremos. Un ejemplo puede encontrarse entre *Producto* e *Ingrediente*.

Generalización

La relación de generalización denota una relación de herencia entre clases. Se representa dibujando un triángulo sin rellenar en el lado de la superclase. La subclase hereda todos los atributos y mensajes descritos en la superclase. Como ejemplo se encuentra una generalización entre *DepositoMonedas* (superclase) y *DepositoMonedasIguales* (subclase).

APÉNDICE C

MODELO IDEF0 DE INGENIERÍA INVERSA

Introducción

A continuación se presenta el índice de nodos correspondiente a los diagramas IDEF0 del proceso de substitución de un producto por medio de Ingeniería Inversa. Se muestran sólo los diagramas considerados más importantes para el desarrollo del MP, si se desea revisar el modelo completo consultar Borja 1997. Tanto en el índice de nodos como en los diagramas, las abreviaturas usadas son:

Info, info Información *PR* Producto de Referencia *PS*Producto Substituto

Índice de Nodos IDEF0

A-0 Substitutción de un producto por medio de Ingeniería Inversa

A0 Productos de Ingeniería Inversa

A1 Extracción y Análisis de Información
- A11 Definición de Requerimientos
  - A111 Definir el Problema
  - A112 Investigación de Mercado
  - A113 Definir Requerimientos del Cliente
- A12 Determinar Especificaciones
  - A121 Definir Información del Producto Substituto (PS)
  - A122 Tipo de Especificaciones
  - A123 Definir Información requerida para el Producto de Referencia (PR)
- A13 Análisis de la Información del PR
  - A131 Extracción de propiedades y definición del PR
    - A1311 Seleccionar una entidad
    - A1312 Cuantificar Propiedades
    - A1313 Analizar función
    - A1314 Identificar partes o interfaces
    - A1315 Extraer características de diseño
    - A1316 Revisar requerimientos e información
  - A132 Documentar e inferir métodos
    - A1321 Buscar información de diseño
    - A1322 Identificar información relevante
    - A1323 Síntesis de información
    - A1324 Herramienta para el desarrollo de Software
  - A133 Documentar e inferir conocimiento adicional

A2 Rediseño

- A21 Seleccionar una entidad
- A22 Rediseñar entidad
  - A221 Rediseño para Funcionalidad
  - A222 Rediseño para Manufactura
    - A2221 Propósito del Proceso de Manufactura
    - A2222 Revisión del Producto Substituto considerando Procesos
    - A2223 Cambiar información no adecuada
    - A2224 Añadir información requerida
  - A223 Rediseño considerando otras perspectivas
- A23 Síntesis y evaluación de cambios

A3 Especificar Información de Manufactura

- A31 Seleccionar una entidad
- A32 Seleccionar Instalación
  - A321 Revisar procesos de instalaciones
  - A322 Revisar máquinas de instalaciones
  - A323 Revisar herramientas de instalaciones
- A33 Especificar Información de Manufactura
  - A331 Seleccionar Procesos
  - A332 Seleccionar Máquinas
  - A333 Seleccionar Herramientas
  - A334 Especificar Herramientas
  - A335 Detallar Operaciones

CONTEXTO

Superior

Información
de manufactura

Necesidad

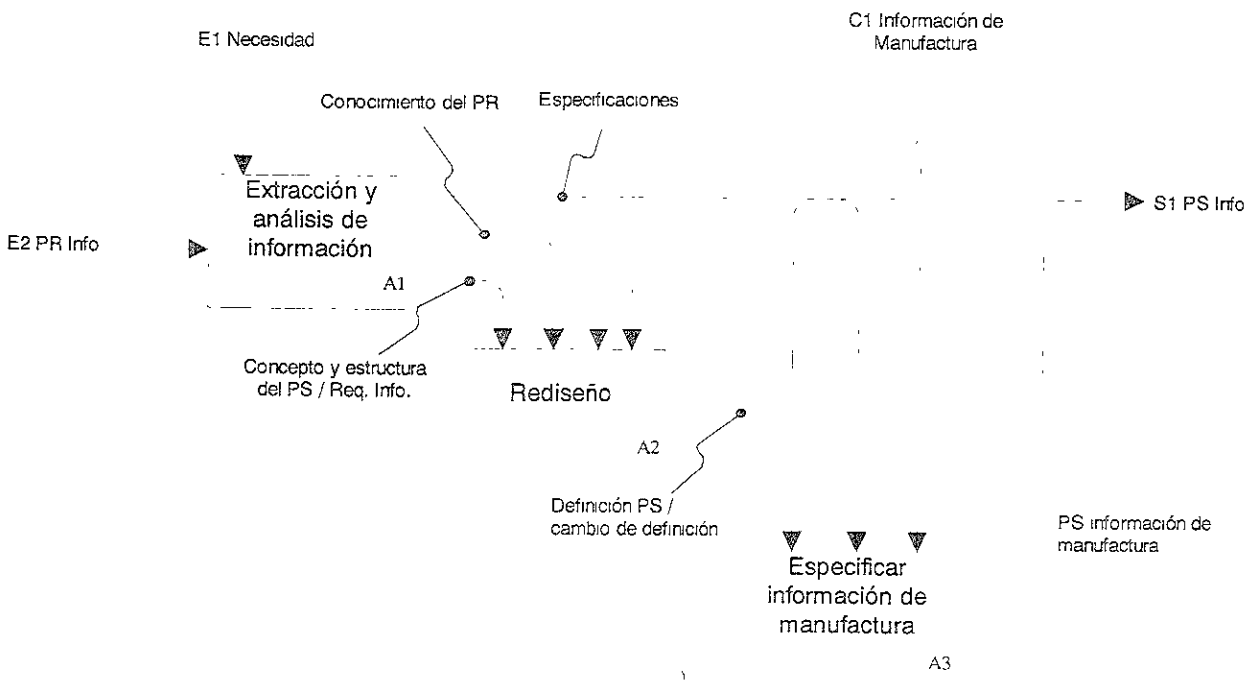
Información Producto
de Referencia

Substitución de un
producto mediante
ingeniería inversa

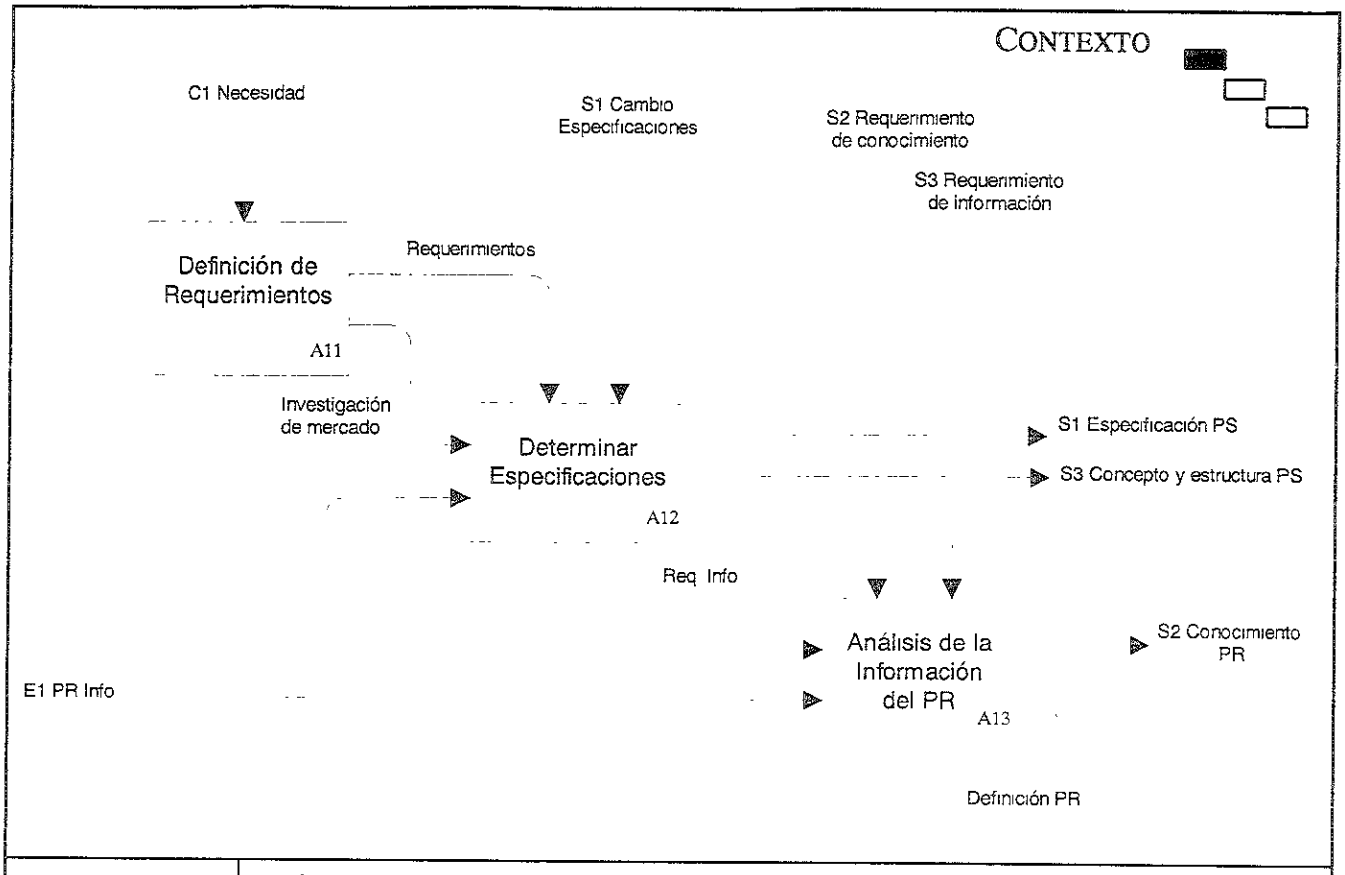
Información Producto
Substituto

NODO: A-0	TÍTULO: **Substitución de un producto por medio de Ingeniería Inversa**

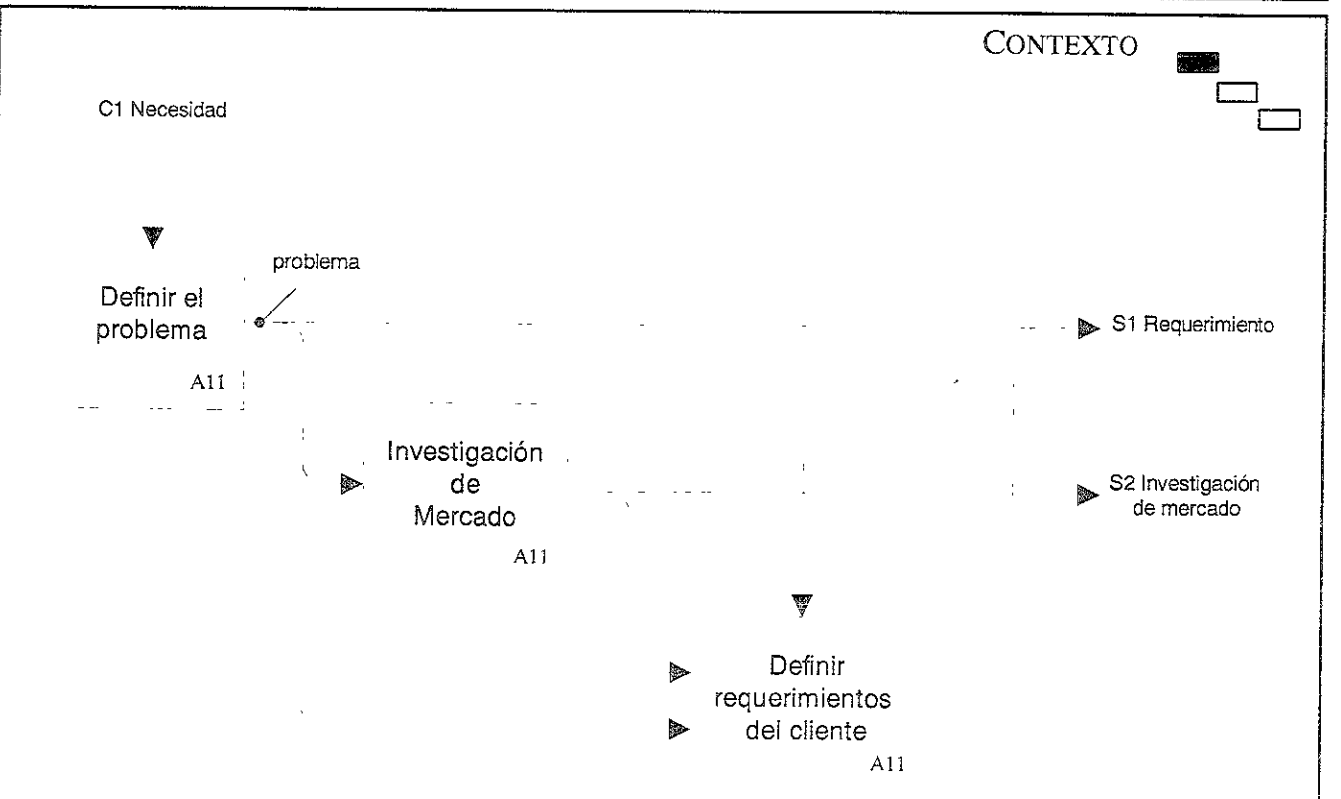
CONTEXTO



NODO: A0	TÍTULO: **Productos de Ingeniería Inversa**

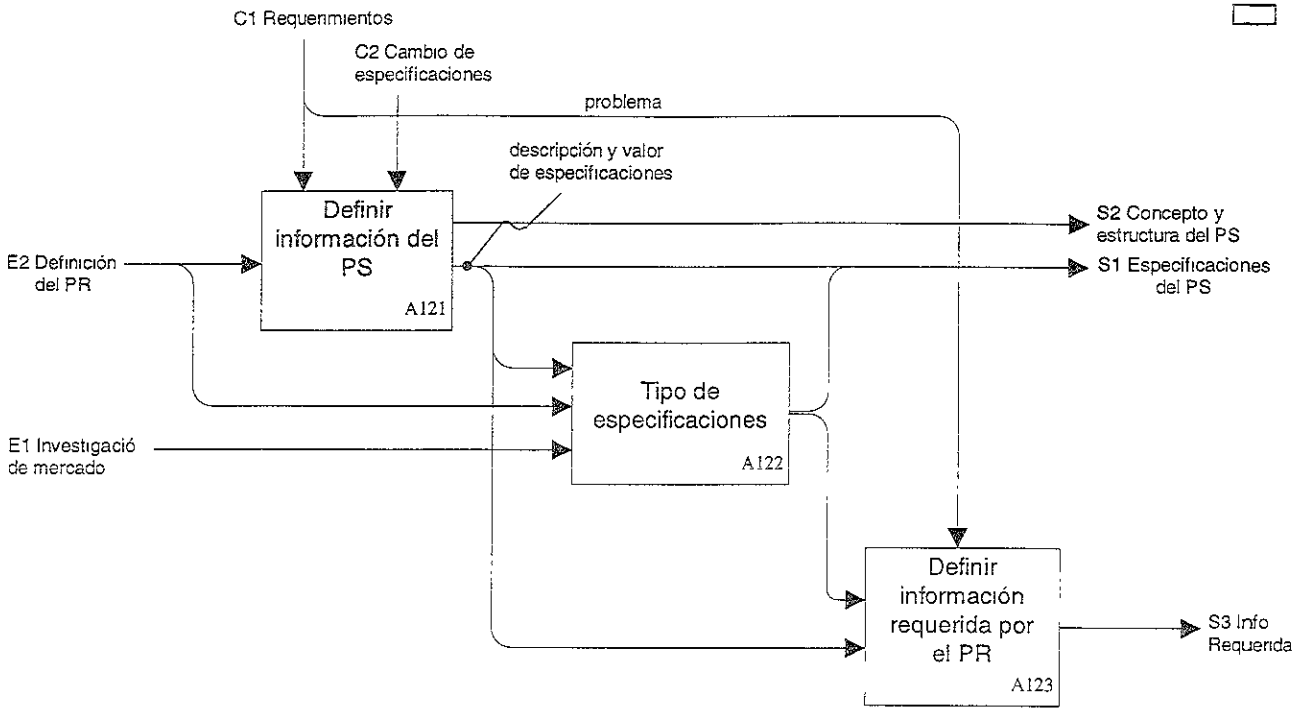


NODO: A1	TÍTULO: **Extracción y Análisis de Información**



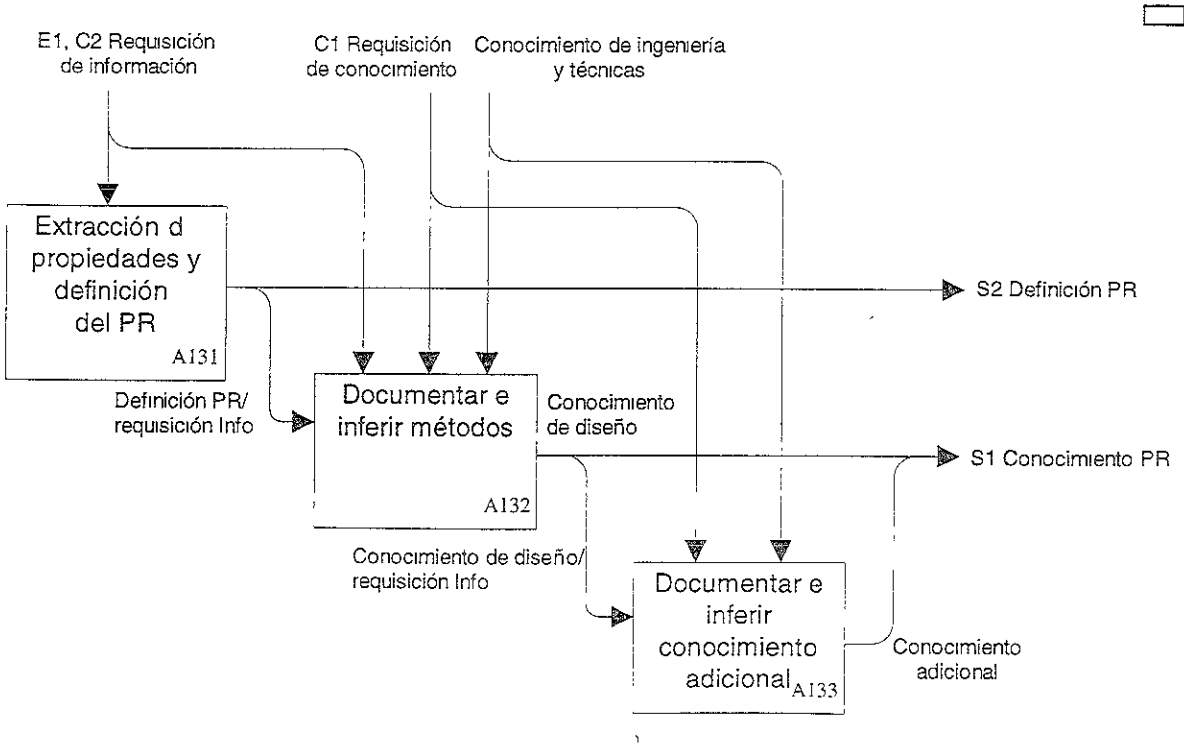
NODO: A11	TÍTULO: **Definición de Requerimientos**

CONTEXTO



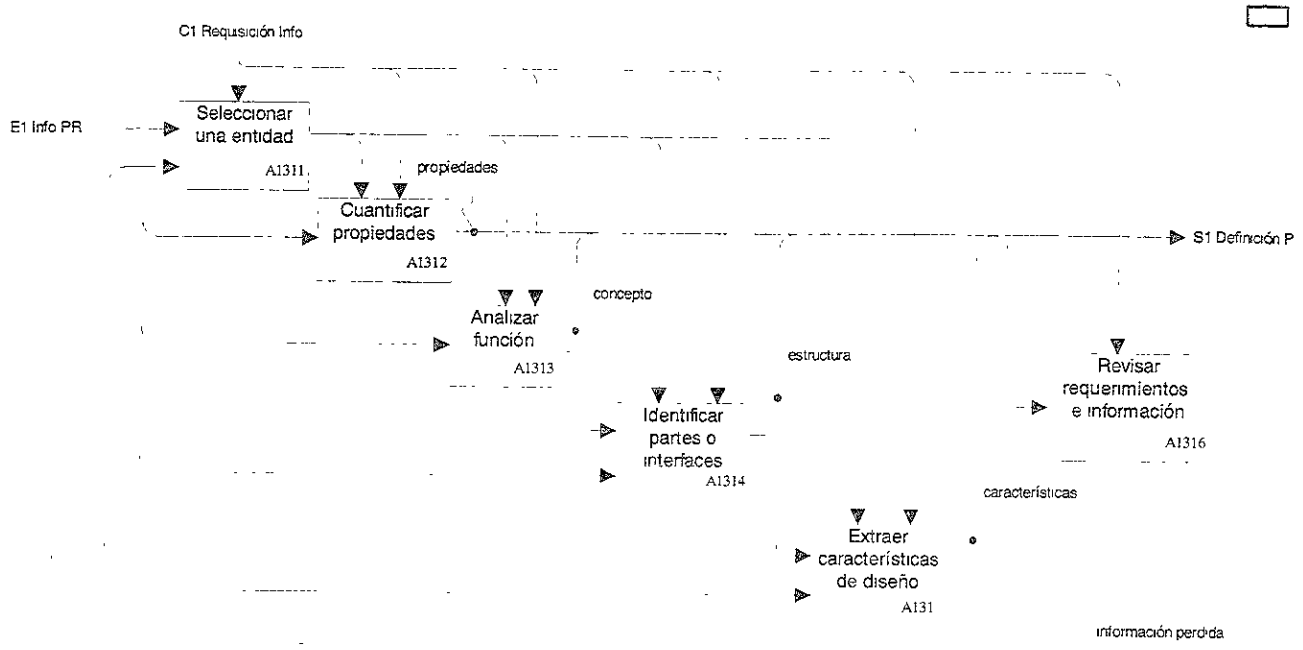
NODO: **A12**	TÍTULO: **Determinar Especificaciones**

CONTEXTO



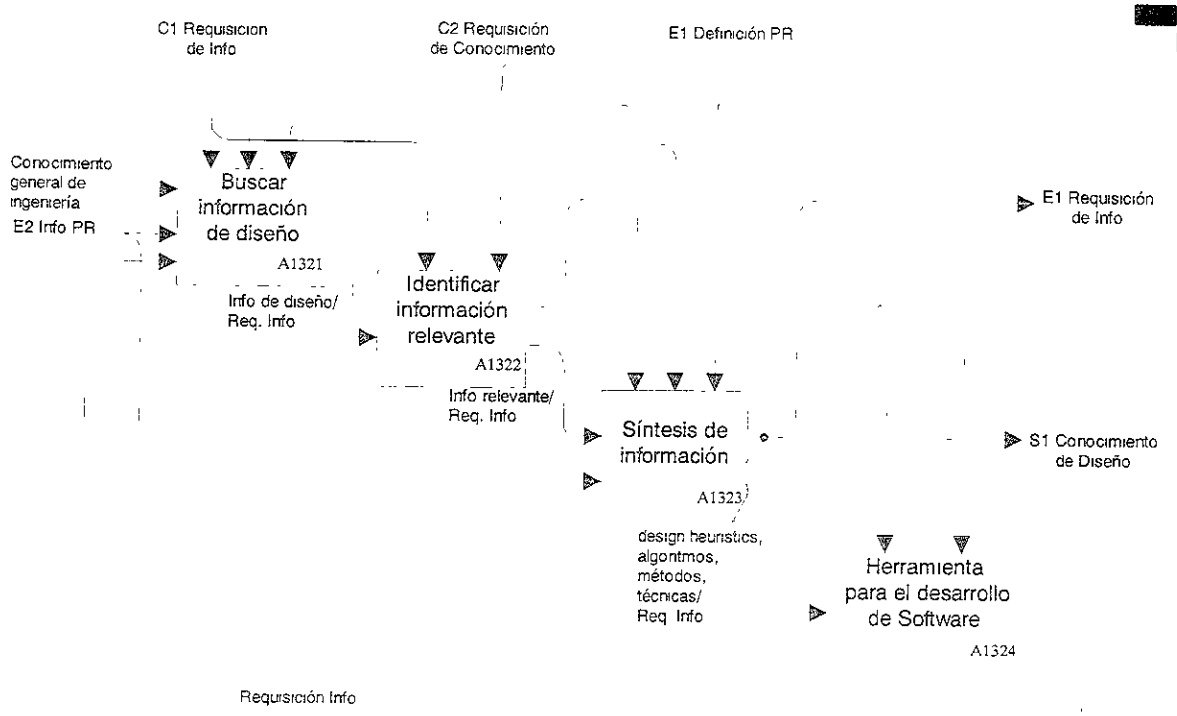
NODO: **A13**	TÍTULO: **Análisis de la Información del PR**

CONTEXTO



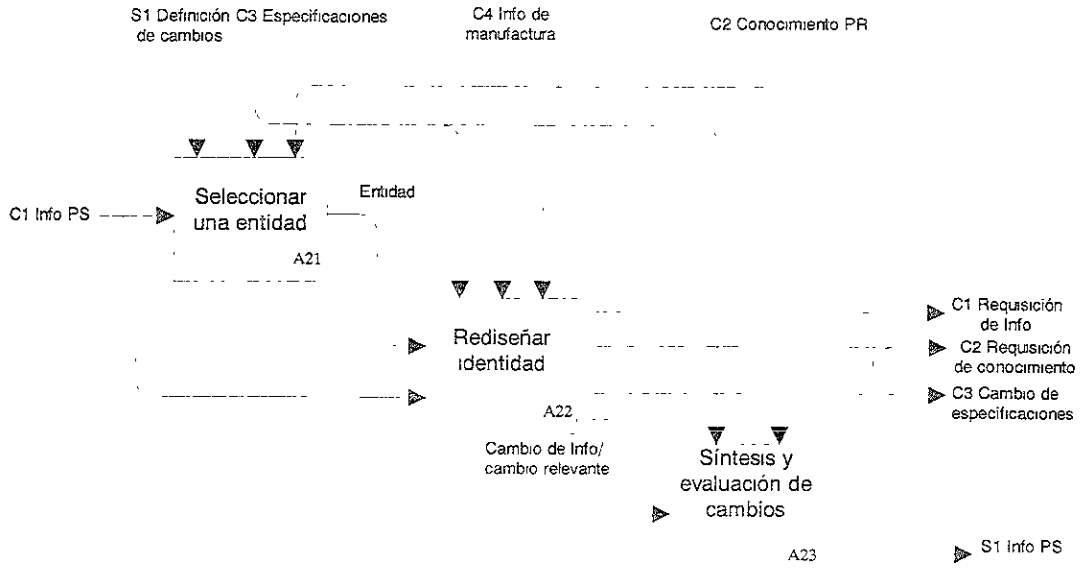
NODO: A131	TÍTULO: **Extracción de propiedades y definición del PR**

CONTEXTO



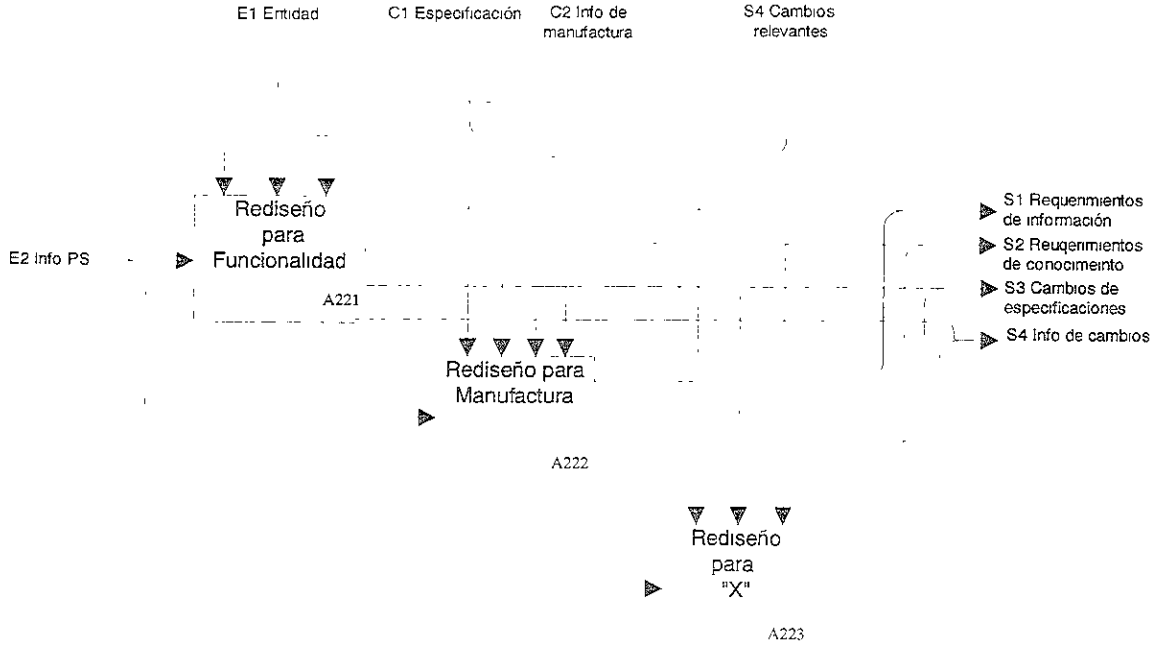
NODO: A132	TÍTULO: **Documentar e inferir métodos**

CONTEXTO



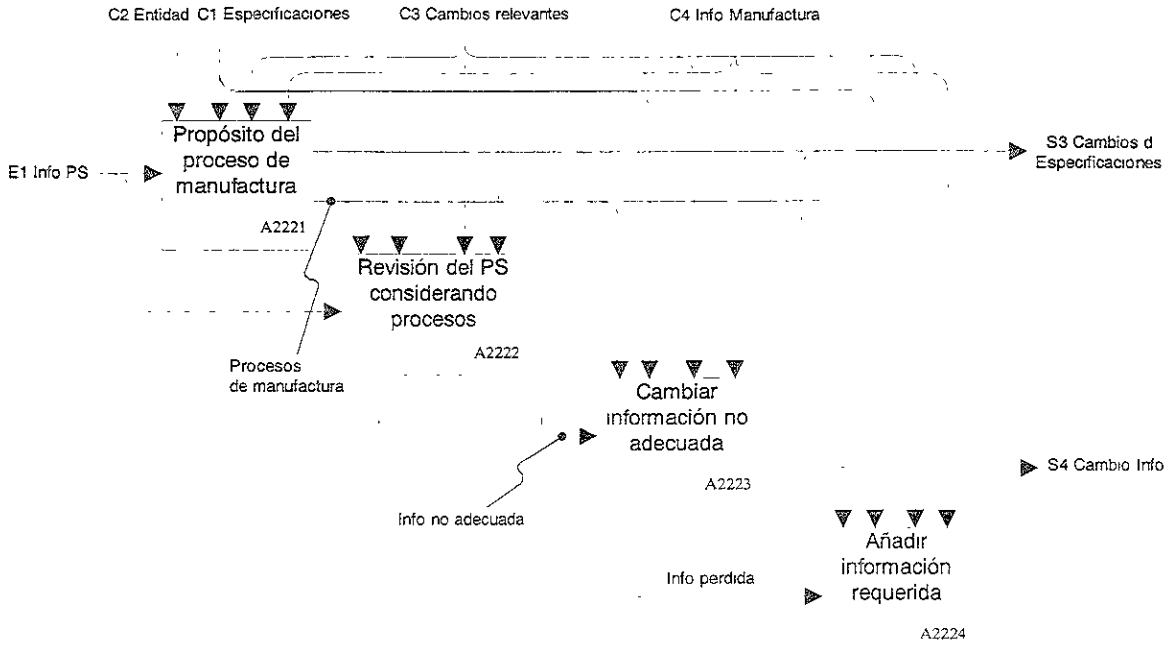
NODO: A2	TÍTULO: **Rediseño**

CONTEXTO

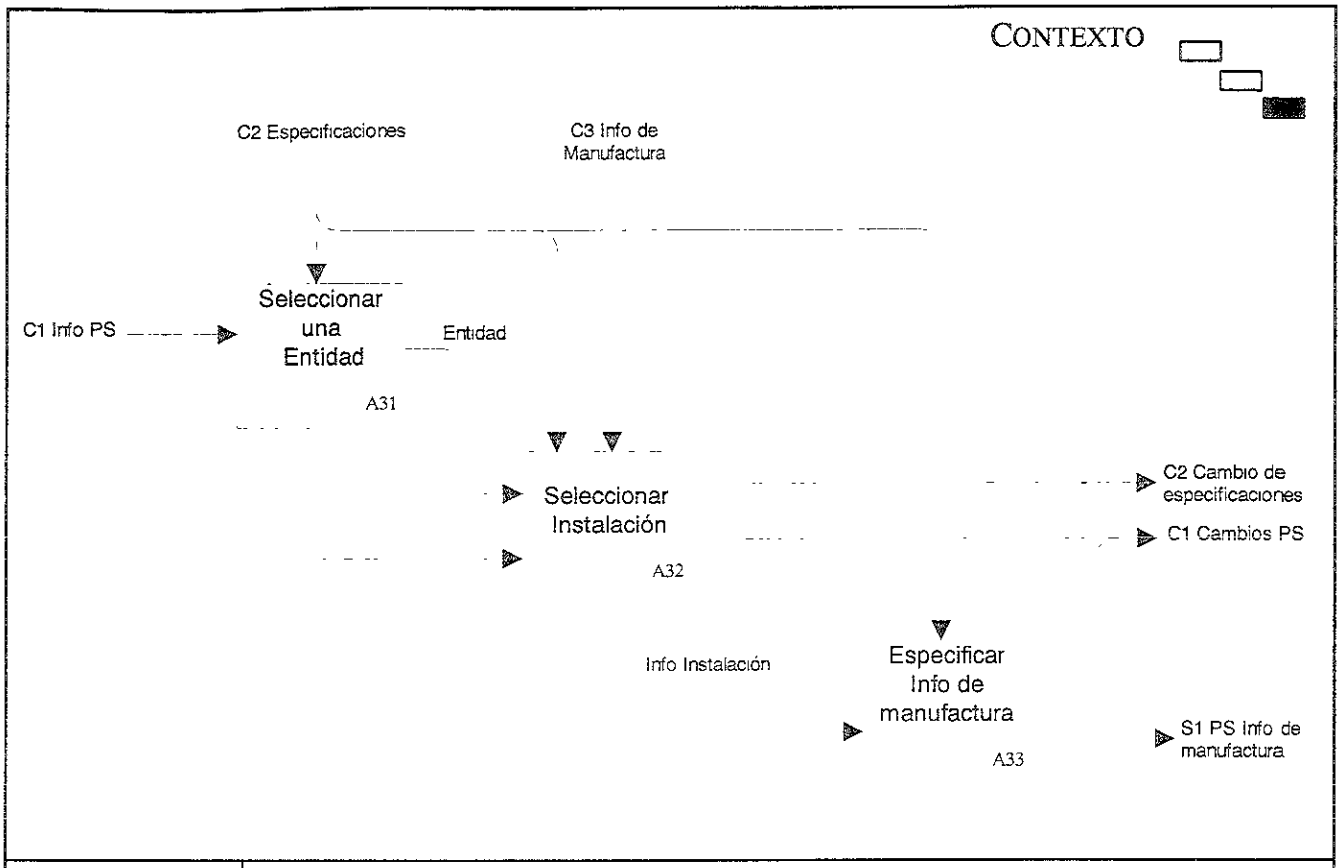


NODO: A22	TÍTULO: **Rediseñar Entidad**

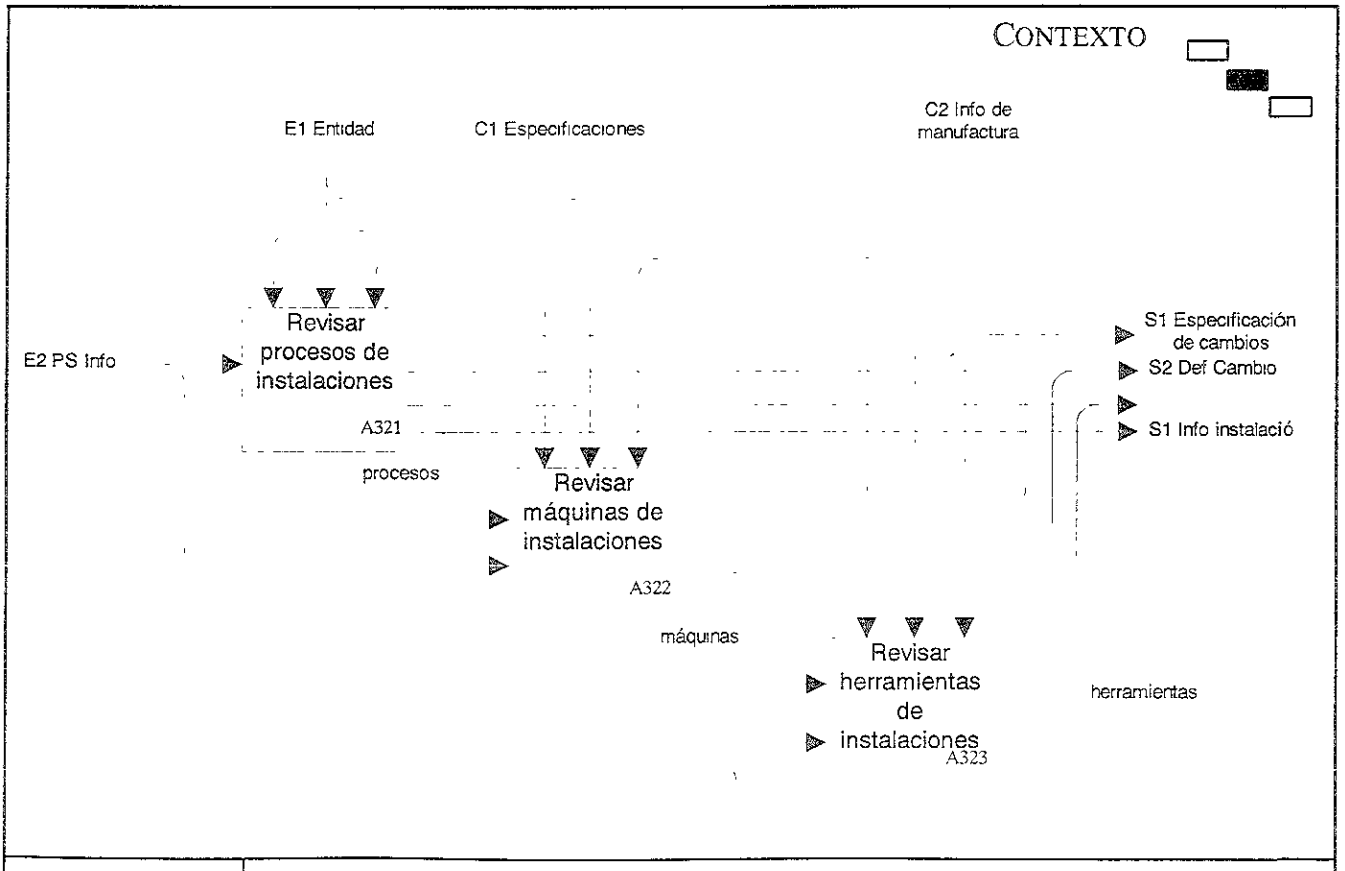
CONTEXTO



NODO: **A222**	TÍTULO: **Rediseño para Manufactura**

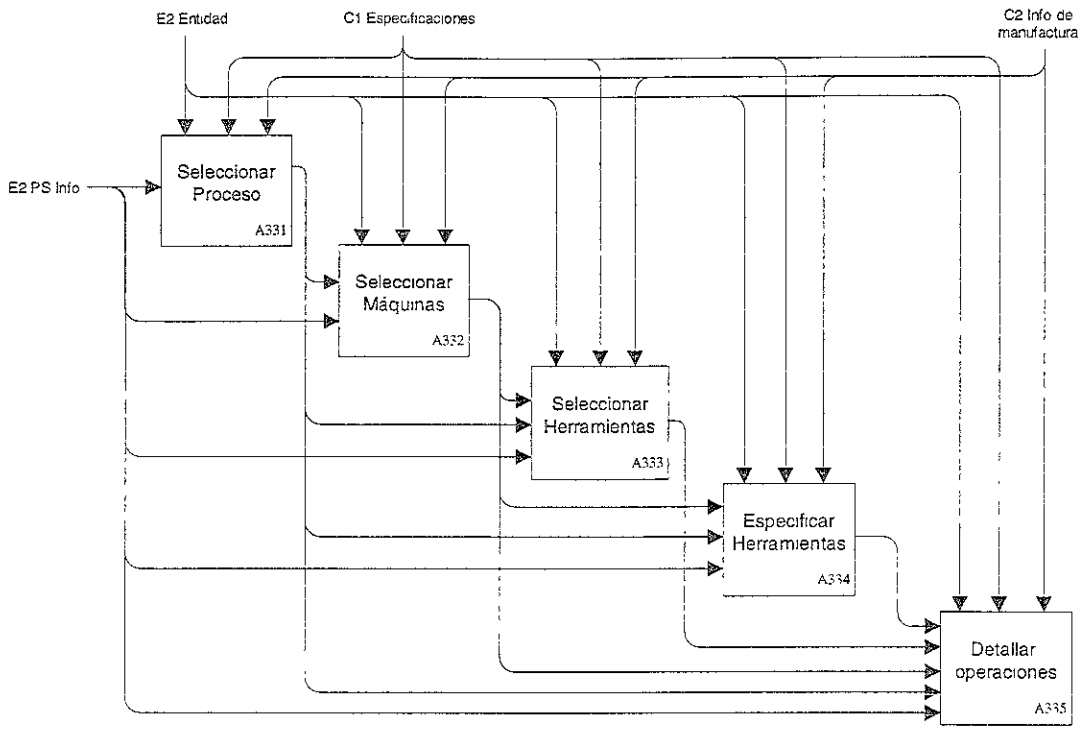


NODO: A3 | TÍTULO: **Especificar Información de Manufactura**



NODO: A32 | TÍTULO: **Seleccionar Instalación**

CONTEXTO



NODO: A33	TÍTULO: **Especificar Información de Manufactura**

APÉNDICE D

DETALLES DE CLASES DEL MP

En el presente apéndice se muestran cuatro ejemplos de las definiciónes de clases del MP, en concreto las clases CComponente, CEntidadD, CDescripcion_fisica y df_cil_ext así como el archivo *schema.osg* y el archivo proyecto.cpp que es una de las partes principales de la interfaz de usuario, el cuál es un esquema de la base de datos en el que se enumeran las clases que pueden ser persistentes y de cuales se pueden formar colecciones.

Clase CEntidadD

```
// EntidadD.h: interface for the CEntidadD class.

/*****************************************************************************
        clase:    CEntidadD

        Descripción: es la clase que representa a una Entidad de Diseño, máximo nivel
        en el Modelo del Producto (MP)

*****************************************************************************/

#include "Proyecto.h"

class CEntidadD
{
public:
        CEntidadD();
        CEntidadD(const char * _tipo_clase, const char * _nombre_entidad);
        virtual ~CEntidadD();

        static os_typespec* get_os_typespec();
//Atributos
public:
        CProyecto * el_proyecto;
        char * tipo_clase;
        char * nombre_entidad;
//Operaciones
public:
        void set_tipo_clase(const char *value);
        void set_nombre_entidad(const char *value);

};
```

Clase CComponente

```
// Componente.h: interface for the CComponente class.
/*****************************************************************************
        clase:    CComponente
        Descripción: es la clase que representa un Componente en el MP

*****************************************************************************/
```

```
#include "Proyecto.h"
#include "EntidadD.h"
#include "Primaria.h"
#include "Definicion.h"

class CComponente : public CEntidadD
{
public:
        CComponente();
        CComponente(const char * _dbnombre,const char * _tipo_clase, const char * _nombre_entidad);
        virtual ~CComponente();

        static os_typespec* get_os_typespec();

// Attributes
public:
        char * dbnombre;

// Operations
public:
        void set_dbnombre(const char *value);
        CComponente* get_Componente(const char * cpszNombre);
        bool Crear_Componente(const char * cpszNombre);
        bool Eliminar_Componente(const char * cpszNombre);
        int     Get_Num_CaractPrimarias(const char * cpszDelComponente);
        const char * Get_Nombre_CaractPrimarias(const char * cpszDelComponente, int iPosicion),
        const char * Get_TipoCPrimaria(const char * cpszDelComponente, int iPosicion);
        void Ordenar_las_Caracteristicas(int iPosicion_Inicial, int iPosicion_Final);
        int Get_Num_Versiones(const char * cpszDelComponente);
        const char * Get_Nombre_Versiones(const char * cpszDelComponente, int iPosicion);

//ObjectStore Relation
public:
        os_relationship_m_1(CComponente,las_caracteristicas,CPrimaria,del_componente,os_List<CPrimaria *>)
        las_caracteristicas;
        os_List<CPrimaria*> * get_las_caracteristicas(){return &las_caracteristicas.getvalue();};

        os_relationship_m_1(CComponente,sus_definiciones,CDefinicion,del_componente,os_List<CDefinicion *>)
        sus_definiciones;
        os_List<CDefinicion*> * get_sus_definiciones(){return &sus_definiciones.getvalue();};

};
```

Clase CDescripcion_Fisica

```
// Descripcion_Fisica.h: interface for the CDescripcion_Fisica class.

/*************************************************************************
        clase:  CDescripcion_Fisica

        Descripción: es la clase que representa la descripción física de una entidad
        de diseño

*************************************************************************/

#include "Proyecto.h"
#include "Definicion.h"
#include "df_comp_cil.h"

class CDescripcion_Fisica
```

```
{
public:
        CDescripcion_Fisica();
        CDescripcion_Fisica(const char * _nombre);
        virtual ~CDescripcion_Fisica();
        static os_typespec* get_os_typespec();

// Attributes
public:
        char * nombre;

// Operations
public:
        void set_nombre(const char *value);
        void Crear_DescripcionFisica(const char * cpszComponente, const char * cpszDefinicion, const char *
        cpszNombreDescripcion);

//ObjectStore Relation
public:
        os_relationship_1_1(CDescripcion_Fisica,la_definicion,CDefinicion,la_descripcion_fisica,CDefinicion*)
        la_definicion;
        void set_la_definicion(CDefinicion* value){la_definicion = value;};

        os_relationship_1_1(CDescripcion_Fisica,el_comp_cil,df_comp_cil,de_la_descripcion,df_comp_cil*)
        el_comp_cil;
        void set_el_comp_cil(df_comp_cil* value){el_comp_cil = value;};

        os_relationship_m_1(CDescripcion_Fisica,las_df_caracteristicas,df_caracteristica,de_la_descripcion,os_List
        <df_caracteristica*>) las_df_caracteristicas;
        os_List<df_caracteristica*> * get_las_df_caracteristicas(){return &las_df_caracteristicas.getvalue();};
};
```

Clase df_cil_ext

```
// df_cil_ext.h: interface for the df_cil_ext class.

/*************************************************************************
        clase:    df_cil_ext

        Descripción: es la clase que representa la descripción física de un cilindro
        externo

*************************************************************************/

#include "Proyecto.h"
#include "Descripcion_Fisica.h"
#include "df_caracteristica.h"

class df_cil_ext : public df_caracteristica
{
public:
        df_cil_ext();
        df_cil_ext(float _D, float _L, float _a, float _b, float _c, char * _nombre, char * _material, float* _punto, int*
        _vector, char * _tipo);
        virtual ~df_cil_ext();

        static os_typespec* get_os_typespec();

// Attributes
public:
        void set_D(float value){ D = value;};
```

```
        float D;
        void set_L(float value){ L = value;};
        float L;
        void set_a(float value){ a = value;};
        float a;
        void set_b(float value){ b = value;};
        float b;
        void set_c(float value){ c = value;};
        float c;

// Operations
        void Crear_cil_ext(const char * cpszComponente, const char * cpszDefinicion, const char *
        cpszNombreDFCaract);
        df_cil_ext* get_Cil_Ext(const char * cpszComponente, const char * cpszDefinicion, const char *
        cpszNombreDFCaract);

        float Get_D(const char * cpszComponente, const char * cpszDefinicion, const char * cpszNombreCaract);
        void Guardar_D(const char * cpszComponente, const char * cpszDefinicion, const char *
        cpszNombreCaract, float fValor);

        float Get_L(const char * cpszComponente, const char * cpszDefinicion, const char * cpszNombreCaract),
        void Guardar_L(const char * cpszComponente, const char * cpszDefinicion, const char *
        cpszNombreCaract, float fValor);

        float Get_a(const char * cpszComponente. const char * cpszDefinicion, const char * cpszNombreCaract);
        void Guardar_a(const char * cpszComponente, const char * cpszDefinicion, const char *
        cpszNombreCaract, float fValor);

        float Get_b(const char * cpszComponente, const char * cpszDefinicion, const char * cpszNombreCaract);
        void Guardar_b(const char * cpszComponente, const char * cpszDefinicion, const char *
        cpszNombreCaract, float fValor);

        float Get_c(const char * cpszComponente, const char * cpszDefinicion, const char * cpszNombreCaract);
        void Guardar_c(const char * cpszComponente, const char * cpszDefinicion, const char *
        cpszNombreCaract, float fValor);

        float Get_Punto_X(const char * cpszComponente, const char * cpszDefinicion, const char *
        cpszNombreCaract);
        float Get_Punto_Y(const char * cpszComponente, const char * cpszDefinicion, const char *
        cpszNombreCaract);
        float Get_Punto_Z(const char * cpszComponente, const char * cpszDefinicion, const char *
        cpszNombreCaract);
        void Guardar_Punto(const char * cpszComponente, const char * cpszDefinicion, const char *
        cpszNombreCaract, float fPunto_X, float fPunto_Y, float fPunto_Z);

        int Get_Vector_I(const char * cpszComponente, const char * cpszDefinicion, const char *
        cpszNombreCaract);
        int Get_Vector_J(const char * cpszComponente, const char * cpszDefinicion, const char *
        cpszNombreCaract);
        int Get_Vector_K(const char * cpszComponente, const char * cpszDefinicion, const char *
        cpszNombreCaract);
        void Guardar_Vector(const char * cpszComponente, const char * cpszDefinicion, const char *
        cpszNombreCaract, int fVector_I, int fVector_J, int fVector_K);
};
```

Esquema de la base de datos

Schema.osg

```
#include <ostore/ostore.hh>
#include <ostore/coll.hh>
#include <ostore/manschem.hh>
#include <ostore/relat.hh>

/* Marco de Referencia */
#include "EntidadD.h"
#include "Componente.h"
#include "Caracteristica.h"
#include "Definicion.h"

/* Taxonomía Característica */
#include "Simple.h"
#include "Primaria.h"
#include "Secundaria.h"
#include "Cilindro.h"
#include "Ranura.h"
#include "Centro.h"
#include "Cuerda.h"
#include "DeTransicion.h"
#include "Hueco.h"
#include "Termino.h"

/* Taxonomía Descripción Física */
#include "Descripcion_Fisica.h"
#include "df_caracteristica.h"
#include "df_cil_ext.h"
#include "df_comp_cil.h"
#include "df_centro.h"
#include "df_ext_conc_rad.h"
#include "df_ext_conv_rad.h"
#include "df_ext_ranura.h"
#include "df_ext_term_chaf.h"
#include "df_ext_termino.h"
#include "df_ext_trans.h"
#include "df_ext_trans_chaf.h"
#include "df_ext_undercut.h"
#include "df_hueco.h"
#include "df_int_cuerda.h"
#include "df_taladrado.h"

/*
 * persistence macros
 */

OS_MARK_SCHEMA_TYPE(CEntidadD)
OS_MARK_SCHEMA_TYPE(CComponente)
OS_MARK_SCHEMA_TYPE(CCaracteristica)
OS_MARK_SCHEMA_TYPE(CSimple);
OS_MARK_SCHEMA_TYPE(CPrimaria)
OS_MARK_SCHEMA_TYPE(CSecundaria)
OS_MARK_SCHEMA_TYPE(CCilindro)
OS_MARK_SCHEMA_TYPE(CRanura)
OS_MARK_SCHEMA_TYPE(CDefinicion);
OS_MARK_SCHEMA_TYPE(CDescripcion_Fisica);
OS_MARK_SCHEMA_TYPE(CCentro);
OS_MARK_SCHEMA_TYPE(CCuerda);
OS_MARK_SCHEMA_TYPE(CDeTransicion);
OS_MARK_SCHEMA_TYPE(CHueco);
```

```
OS_MARK_SCHEMA_TYPE(CTermino);
OS_MARK_SCHEMA_TYPE(df_caracteristica);
OS_MARK_SCHEMA_TYPE(df_cil_ext);
OS_MARK_SCHEMA_TYPE(df_comp_cil);
OS_MARK_SCHEMA_TYPE(df_centro);
OS_MARK_SCHEMA_TYPE(df_ext_conc_rad);
OS_MARK_SCHEMA_TYPE(df_ext_conv_rad);
OS_MARK_SCHEMA_TYPE(df_ext_ranura);
OS_MARK_SCHEMA_TYPE(df_ext_term_chaf);
OS_MARK_SCHEMA_TYPE(df_ext_termino);
OS_MARK_SCHEMA_TYPE(df_ext_trans);
OS_MARK_SCHEMA_TYPE(df_ext_trans_chaf);
OS_MARK_SCHEMA_TYPE(df_ext_undercut);
OS_MARK_SCHEMA_TYPE(df_hueco);
OS_MARK_SCHEMA_TYPE(df_int_cuerda);
OS_MARK_SCHEMA_TYPE(df_taladrado);

OS_MARK_SCHEMA_TYPE(os_List<CEntidadD*>);
OS_MARK_SCHEMA_TYPE(os_List<CComponente*>);
OS_MARK_SCHEMA_TYPE(os_List<CCaracteristica*>);
OS_MARK_SCHEMA_TYPE(os_List<CSimple*>);
OS_MARK_SCHEMA_TYPE(os_List<CPrimaria*>);
OS_MARK_SCHEMA_TYPE(os_List<CSecundaria*>);
OS_MARK_SCHEMA_TYPE(os_List<CCilindro*>);
OS_MARK_SCHEMA_TYPE(os_List<CRanura*>);
OS_MARK_SCHEMA_TYPE(os_List<CDefinicion*>);
OS_MARK_SCHEMA_TYPE(os_List<CDescripcion_Fisica*>);
OS_MARK_SCHEMA_TYPE(os_List<CCentro*>);
OS_MARK_SCHEMA_TYPE(os_List<CCuerda*>);
OS_MARK_SCHEMA_TYPE(os_List<CDeTransicion*>);
OS_MARK_SCHEMA_TYPE(os_List<CHueco*>);
OS_MARK_SCHEMA_TYPE(os_List<CTermino*>);
OS_MARK_SCHEMA_TYPE(os_List<df_caracteristica*>);
OS_MARK_SCHEMA_TYPE(os_List<df_cil_ext*>);
OS_MARK_SCHEMA_TYPE(os_List<df_comp_cil*>);
OS_MARK_SCHEMA_TYPE(os_List<df_centro*>);
OS_MARK_SCHEMA_TYPE(os_List<df_ext_conc_rad*>);
OS_MARK_SCHEMA_TYPE(os_List<df_ext_conv_rad*>);
OS_MARK_SCHEMA_TYPE(os_List<df_ext_ranura*>);
OS_MARK_SCHEMA_TYPE(os_List<df_ext_term_chaf*>);
OS_MARK_SCHEMA_TYPE(os_List<df_ext_termino*>);
OS_MARK_SCHEMA_TYPE(os_List<df_ext_trans*>);
OS_MARK_SCHEMA_TYPE(os_List<df_ext_trans_chaf*>);
OS_MARK_SCHEMA_TYPE(os_List<df_ext_undercut*>);
OS_MARK_SCHEMA_TYPE(os_List<df_hueco*>);
OS_MARK_SCHEMA_TYPE(os_List<df_int_cuerda*>);
OS_MARK_SCHEMA_TYPE(os_List<df_taladrado*>);
```

Archivo proyecto.cpp

```
// Proyecto.cpp: implementation of the CProyecto class.

#include "stdafx.h"
#include "MET.h"
#include "Proyecto.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

#include "Componente.h"

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CProyecto::CProyecto()
{
        Nombre_BaseDatos = _T("");
        Path_Proyecto = _T("");
        Componente_Activo = _T("");
        Version_Activa = _T("");
}

CProyecto::~CProyecto()
{

}

/*****************************************************************************
        functions:      Fijar_Nombre_BaseDatos
                        Fijar_Path_Proyecto
                        Fijar_Componente_Activo
                        Fijar_Version_Activa
        input:          el valor de la variable en cuestión
        return value:   nada
        Las funciones fijan el valor de la variable y lo guardan en un archivo de texto.
*****************************************************************************/
void CProyecto::Fijar_Nombre_BaseDatos(const char * value)
{
        //Nombre_BaseDatos =  value;

        //Nombre del fichero
        CString csNomFichero = "c:\\met\\met_1.tmp";

        //Abrir el fichero para escribir
        CFile MiFichero;

        //Estructura para almacenar el estado del fichero
        CFileStatus EstadoFi;
        UINT nModoDeAcceso = CFile::modeWrite;//acceso para escribir

        //GetStatus devuelve true si el fichero existe, o false si no existe
        //Si existe, se borra
        if(MiFichero.GetStatus(csNomFichero, EstadoFi))
                CFile::Remove(csNomFichero);

        //Como no existe, se extiende el modo de acceso para crearlo
```

```
        if(!MiFichero.GetStatus(csNomFichero, EstadoFi))
                nModoDeAcceso |= CFile::modeCreate;

        if(!MiFichero.Open(csNomFichero, nModoDeAcceso))
        {
                AfxMessageBox("No se puede abrir el fichero para escribir");
                return;
        }

        //Escribir el texto en el fichero
        CString csTexto;
        csTexto = value;
        MiFichero.Write((const char *)csTexto, csTexto.GetLength());
        MiFichero.Close();
}

void CProyecto::Fijar_Path_Proyecto(const char * value)
{
        //Nombre del fichero
        CString csNomFichero = "c:\\met\\met_2.tmp";

        //Abrir el fichero para escribir
        CFile MiFichero;

        //Estructura para almacenar el estado del fichero
        CFileStatus EstadoFi;
        UINT nModoDeAcceso = CFile::modeWrite;//acceso para escribir

        //GetStatus devuelve true si el fichero existe, o false si no existe
        //Si existe, se borra
        if(MiFichero.GetStatus(csNomFichero, EstadoFi))
                CFile::Remove(csNomFichero);

        //Como no existe, se extiende el modo de acceso para crearlo
        if(!MiFichero.GetStatus(csNomFichero, EstadoFi))
                nModoDeAcceso |= CFile::modeCreate;

        if(!MiFichero.Open(csNomFichero, nModoDeAcceso))
        {
                AfxMessageBox("No se puede abrir el fichero para escribir");
                return;
        }

        //Escribir el texto en el fichero
        CString csTexto;
        csTexto = value;
        MiFichero.Write((const char *)csTexto, csTexto.GetLength());
        MiFichero.Close();
}

void CProyecto::Fijar_Componente_Activo(const char * value)
{
        //Componente_Activo = value;

        //Nombre del fichero
        CString csNomFichero = "c:\\met\\met_3.tmp";

        //Abrir el fichero para escribir
        CFile MiFichero;

        //Estructura para almacenar el estado del fichero
        CFileStatus EstadoFi;
```

```
        UINT nModoDeAcceso = CFile::modeWrite;//acceso para escribir

        //GetStatus devuelve true si el fichero existe, o false si no existe
        //Si existe, se borra
        if(MiFichero.GetStatus(csNomFichero, EstadoFi))
                CFile::Remove(csNomFichero);

        //Como no existe, se extiende el modo de acceso para crearlo
        if(!MiFichero.GetStatus(csNomFichero, EstadoFi))
                nModoDeAcceso |= CFile::modeCreate;

        if(!MiFichero.Open(csNomFichero, nModoDeAcceso))
        {
                AfxMessageBox("No se puede abrir el fichero para escribir");
                return;
        }

        //Escribir el texto en el fichero
        CString csTexto;
        csTexto = value;
        MiFichero.Write((const char *)csTexto, csTexto.GetLength());
        MiFichero.Close();
}

void CProyecto::Fijar_Version_Activa(const char * value)
{
        //Version_Activa = value;

        //Nombre del fichero
        CString csNomFichero = "c:\\met\\met_4.tmp";

        //Abrir el fichero para escribir
        CFile MiFichero;

        //Estructura para almacenar el estado del fichero
        CFileStatus EstadoFi;
        UINT nModoDeAcceso = CFile::modeWrite;//acceso para escribir

        //GetStatus devuelve true si el fichero existe, o false si no existe
        //Si existe, se borra
        if(MiFichero.GetStatus(csNomFichero, EstadoFi))
                CFile::Remove(csNomFichero);

        //Como no existe, se extiende el modo de acceso para crearlo
        if(!MiFichero.GetStatus(csNomFichero, EstadoFi))
                nModoDeAcceso |= CFile::modeCreate;

        if(!MiFichero.Open(csNomFichero, nModoDeAcceso))
        {
                AfxMessageBox("No se puede abrir el fichero para escribir");
                return;
        }

        //Escribir el texto en el fichero
        CString csTexto;
        csTexto = value;
        MiFichero.Write((const char *)csTexto, csTexto.GetLength());
        MiFichero.Close();
}
```

```
/*****************************************************************************
        functions:      Traer_Nombre_BaseDatos
                        Traer_Path_Proyecto
                        Traer_Componente_Activo
                        Traer_Version_Activa
        input:          nada
        return value:   el valor de la variable en cuestión
        Las funciones recuperan del archivo de texto el valor de las variables.
*****************************************************************************/
const char * CProyecto::Traer_Nombre_BaseDatos()
{
        const char * csRespuesta = _T("");

        //Obtener nombre del fichero
        CString csNomFichero = "c:\\met\\met_1.tmp";

        //Abrir Fichero para leer
        CFile MiFichero;
        if(!MiFichero.Open(csNomFichero, CFile::modeRead))
        {
                AfxMessageBox("No se puede abrir el fichero para leer");
                return(csRespuesta);
        }

        //Leer el texto del fichero
        UINT tam = (UINT)(MiFichero.GetLength());
        char * cpszTexto = new char[tam+1];
        MiFichero.Read(cpszTexto, tam);
        cpszTexto[tam] = 0;

        csRespuesta = (CString) cpszTexto;

        return(csRespuesta);
}

const char * CProyecto::Traer_Path_Proyecto()
{
        const char * csRespuesta = _T("");

        //Obtener nombre del fichero
        CString csNomFichero = "c:\\met\\met_2.tmp";

        //Abrir Fichero para leer
        CFile MiFichero;
        if(!MiFichero.Open(csNomFichero, CFile::modeRead))
        {
                AfxMessageBox("No se puede abrir el fichero para leer");
                return(csRespuesta);
        }

        //Leer el texto del fichero
        UINT tam = (UINT)(MiFichero.GetLength());
        char * cpszTexto = new char[tam+1];
        MiFichero.Read(cpszTexto, tam);
        cpszTexto[tam] = 0;

        csRespuesta = (CString) cpszTexto;

        return(csRespuesta);
}

const char * CProyecto::Traer_Componente_Activo()
```

```
{
	const char * csRespuesta = _T("");

	//Obtener nombre del fichero
	CString csNomFichero = "c:\\met\\met_3.tmp";

	//Abrir Fichero para leer
	CFile MiFichero;
	if(!MiFichero.Open(csNomFichero, CFile::modeRead))
	{
		AfxMessageBox("No se puede abrir el fichero para leer");
		return(csRespuesta);
	}

	//Leer el texto del fichero
	UINT tam = (UINT)(MiFichero.GetLength());
	char * cpszTexto = new char[tam+1];
	MiFichero.Read(cpszTexto, tam);
	cpszTexto[tam] = 0;

	csRespuesta = (CString) cpszTexto;

	return(csRespuesta);
}

const char * CProyecto::Traer_Version_Activa()
{
	const char * csRespuesta = _T("");

	//Obtener nombre del fichero
	CString csNomFichero = "c:\\met\\met_4.tmp";

	//Abrir Fichero para leer
	CFile MiFichero;
	if(!MiFichero.Open(csNomFichero, CFile::modeRead))
	{
		AfxMessageBox("No se puede abrir el fichero para leer");
		return(csRespuesta);
	}

	//Leer el texto del fichero
	UINT tam = (UINT)(MiFichero.GetLength());
	char * cpszTexto = new char[tam+1];
	MiFichero.Read(cpszTexto, tam);
	cpszTexto[tam] = 0;

	csRespuesta = (CString) cpszTexto;

	return(csRespuesta);
}

/*****************************************************************************
	function:		Crear_Nuevo_Proyecto
	input:			la ruta del proyecto
	return value:	nada
	La función crea un nuevo proyecto (Base de Datos).
*****************************************************************************/
void CProyecto::Crear_Nuevo_Proyecto
(
	const char * cpszPath_Proyecto
)
{
```

```
	//Se crea la base de datos
	os_database *dBD = os_database::create(cpszPath_Proyecto,0,0664);

	//Una vez creada la BD es necesario inicializarla, es decir
	//incluir las caracteristicas principales de la base, como
	//raices, colecciones, objetos creados por default, etc.

	OS_BEGIN_TXN(trans,0,os_transaction::update)
	{
		os_database_root *pRaiz1;

		//Crear raiz de la BD
		pRaiz1 = dBD-> create_root("RaizComponentes");

		//Crear la lista de componentes
		os_List<CComponente *> *componentes = &os_List<CComponente*>::create(dBD,0,50);

		//Fijar el valor de la lista a la raiz
		pRaiz1->set_value(componentes);

	}
	OS_END_TXN(trans)

	dBD->close();
}


/*****************************************************************************
	function:	Abrir_Proyecto
	input:		la ruta y nombre del proyecto
	return value:	verdadero o falso, dependiendo si se pudo abrir el proyecto
	La función abre un proyecto existente (Base de Datos).
*****************************************************************************/
bool CProyecto::Abrir_Proyecto
(
	const char * cpszPath_Proyecto,
	const char * cpszNombre_Proyecto
)
{
	bool bAbrir = true;
	bool bRespuesta = true;

	//Revisar si existe un proyecto abierto
	if(Nombre_BaseDatos != _T(""))
	{
		if(AfxMessageBox("Existe un proyecto actualmente abierto, ¿desea cerrarlo y abrir
otro?",MB_YESNO | MB_ICONQUESTION) != IDYES)
		{
			bAbrir = false;
			bRespuesta = false;
		}
	}

	if(bAbrir == true)
	{
		Fijar_Nombre_BaseDatos(cpszNombre_Proyecto);
		Fijar_Path_Proyecto(cpszPath_Proyecto);
		Fijar_Componente_Activo(_T(""));
		Fijar_Version_Activa(_T(""));
	}

	return(bRespuesta);
```

```
}

/*****************************************************************************
        function:       Cerrar_Proyecto
        input:          nada
        return value:   nada
        La función cierra el proyecto activo (Base de Datos), reinicia los valores
        predeterminados de los atributos de CProyecto.
*****************************************************************************/
void CProyecto::Cerrar_Proyecto()
{
        Fijar_Nombre_BaseDatos(_T(""));
        Fijar_Path_Proyecto(_T(""));
        Fijar_Componente_Activo(_T(""));
        Fijar_Version_Activa(_T(""));
}



/*****************************************************************************
        function:       Get_Numero_de_Componentes
        input:          nada
        return value:   el número de componentes (ejes) en el proyecto
        La función regresa el número de componentes en el proyecto activo.
*****************************************************************************/
int CProyecto::Get_Numero_Componentes()
{
        //Declaración de variables
        int iNum_Componentes = 0;
        os_database *dBD = os_database::open(Traer_Path_Proyecto(),0,0664);

        //Recuperando elementos de la BD
        OS_BEGIN_TXN(trans1,0,os_transaction::update)
        {
                //Una variable raiz
                os_database_root *pRaiz;
                //Encontrar raiz
                pRaiz = dBD->find_root("RaizComponentes");
                //Una variable os_List
                os_List<CComponente *> *pListComp;
                pListComp = (os_List<CComponente *>*)pRaiz->get_value();
                iNum_Componentes = pListComp->cardinality();
        }
        OS_END_TXN(trans1)
        dBD->close();

        return(iNum_Componentes);
}

/*****************************************************************************
        function:       Get_Nombre_Componente
        input:          la posición en la lista de la BD
        return value:   el nombre del componente
        La función regresa el nombre del componentes en la posición solicitada.
*****************************************************************************/
const char * CProyecto::Get_Nombre_Componente
(
        int iPosicion
)
{
        const char * csNombre = _T("");
        CComponente *pComp;
```

```
        os_database *dBD = os_database::open(Traer_Path_Proyecto(),0,0664);

        //Recuperando elementos de la BD
        OS_BEGIN_TXN(trans1,0,os_transaction::update)
        {
                //Una variable raiz
                os_database_root *pRaiz;
                //Encontrar raiz
                pRaiz = dBD->find_root("RaizComponentes");
                //Una variable os_List
                os_List<CComponente *> *pListComp;
                pListComp = (os_List<CComponente *>*)pRaiz->get_value();

                pComp = pListComp->retrieve(iPosicion);
                csNombre = (CString) pComp->nombre_entidad;
        }
        OS_END_TXN(trans1)
        dBD->close();

        return(csNombre);
}
```