UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DESARROLLO DEL MODELO DEL PRODUCTO UTILIZANDO AML

T E S I S

QUE PARA OBTENER EL TÍTULO DE
INGENIERO EN COMPUTACIÓN

PRESENTAN:
JAVIER FERNÁNDEZ HERNÁNDEZ
JUAN CARLOS MÁRQUEZ TORRES

DIRECTOR DE TESIS:
DR. JESÚS MANUEL DORADOR GONZÁLEZ

MÉXICO D.F. **ABRIL 2004**

Agradecimientos

A mi Papá y a mi Mamá, quienes siempre me han alentado y apoyado para continuar con mis estudios y de quienes he aprendido lo que no se enseña en una escuela.

Gracias.

A Mayra, por su amor, apoyo y por ser la principal razón por la que quiero superarme.

A Erick y Abel que siempre han estado cerca de mi cuando he necesitado de un amigo.

A mis amigos del CDM y el LIMAC por todo el apoyo y sus muestras de cariño, en especial a Soco.

A mis compañeros y amigos de la carrera, gracias.

Juan Carlos Márquez Torres

Agradecimientos

Dedicada esta tesis a mi papá, a mi mamá y a mi director de la Secundaria por todo su apoyo y por creer en mi.

Gracias

Javier Fernández Hernández

CONTENIDO

1 Introducción

1.1 Contexto actual

Bajo las presiones que se presentan con el fenómeno de la globalización, las empresas se enfrentan a la problemática de una competencia internacional, por lo que se ven en la necesidad de hacer ciclos de vida de producto cada vez más eficientes y presentar una diversidad mayor de productos a bajo precio y de mayor calidad. Para poder solventar esta problemática han surgido diversas técnicas, herramientas y modelos que optimizan el desempeño en los procesos de producción logrando mejoras significativas en el costo del producto, funcionalidad y tiempo de comercialización.

Una de estas herramientas es la ingeniería concurrente, la cual permite tomar decisiones desde la fase temprana del ciclo de vida del producto, es decir, en la etapa de diseño. En esta etapa se toma la información del producto para generar prototipos virtuales (esto involucra todas las etapas del ciclo de vida del producto) y de esta manera analiza los datos obtenidos y ayuda a saber si son susceptibles de algún cambio y que consecuencias traerá en etapas posteriores, como manufactura y ensamble.

Gracias a los avances tecnológicos y reducción de precios en el área de cómputo, el desarrollo de la ingeniería concurrente se ha visto enormemente favorecido ya que mediante la utilización de equipo de cómputo se han conseguido evitar errores y reducir los tiempos y costos en la elaboración de diseños del producto, optimizando el ciclo de vida de éste.

Se han desarrollado diversos sistemas comerciales intentando resolver esta problemática, pero la mayoría de estos sistemas, no fueron desarrollados para compartir la información que obtienen con otros sistemas y no proporcionan ayuda al diseñador en las fases tempranas del diseño, por lo que no suministran un ambiente integrado para la ingeniería concurrente.

Una forma de proporcionar un ambiente integrado es mediante los Modelos de Información. Particularmente se ha trabajado en los Modelos de Producto y Modelos de Manufactura; en el primero se obtienen características del producto y en el segundo la información que comprende a los procesos de manufactura, esta información es descrita y almacenada, para ayudar a la toma de decisiones.

Además de los sistemas comerciales, se han desarrollado en varias universidades del mundo diversas propuestas para atacar este tipo de problemática, algunas de estas propuestas se listan a continuación:

Por Loughborough University, Inglaterra:

- Arturo Molina presentó una propuesta sobre el Modelo de Manufactura.
- Vicente Borja realizó una aplicación del Modelo del Producto para Ingeniería Inversa.
- Osiris Canciglieri desarrolló una propuesta de utilizar varias vistas en el Modelo del Producto.
- Carlos Costa trabajó en una aplicación del Modelo del Producto para procesos de inyección de plásticos.
- Jie Zhao implementó una aplicación del Modelo de Manufactura para generar código de control numérico.
- Jesús Manuel Dorador utilizó los Modelos del Producto y de Manufactura para realizar diseños de ensamble y planeación de procesos de ensamble.

Por la University of Leeds, Inglaterra, Allison McKay presentó una propuesta sobre el Modelo de Producto.

Por la Universidad Nacional Autónoma de México

- Álvaro Ayala realizó un Modelador de Celdas de Manufactura para asistir al diseñador en la representación de celdas de manufactura de acuerdo al Modelo de Manufactura.
- Fabiola Socorro Vega hizo uso del Modelo del Producto para generar representaciones graficas de ejes de transmisión en un sistema de CAD.
- Socorro Armenta usó el Modelo del Producto para evaluar la facilidad de ensamble en una familia de productos.

Sin embargo todos estos diseños se han visto ante la problemática de comunicación entre los diversos programas utilizados para el proyecto, debido a que en algunos de ellos las últimas versiones no soportan algunas características de versiones anteriores lo que implica tener forzosamente la versión en la que se desarrolló la aplicación, dificultando lá portabilidad, manejo y vida útil de dichas aplicaciones.

El estudio que se presenta en esta tesis, se enfoca en el desarrollo del Modelo del Producto desarrollado para el Lenguaje de Modelado Adaptáble (AML por sus siglas en inglés). Aprovechando las ventajas que presenta este lenguaje se quiere satisfacer la necesidad de modelar los datos almacenados en el Modelo del Producto para su estudio y análisis; al mismo tiempo se pretende conseguir de esta forma que, partiendo de un modelo generado por medio de este lenguaje se pueda extraer la información correspondiente a restricciones geométricas y paramétricas, además de la información de características no geométricas como lo son: el tipo material, funcionalidad, tolerancias, propiedades del material, especificaciones técnicas, entre otras.

1.2 Objetivos

El objetivo principal del presente trabajo es analizar la factibilidad para poder implementar el Modelo del Producto, en una fase temprana de este, utilizando el lenguaje de programación AML.

Para llevar a cabo este objetivo se requiere realizar las siguientes actividades:

- Revisar trabajos previos en materia de modelos de información, en particular lo referente a Modelo del Producto.
- Proponer un Modelo del Producto o utilizar uno existente para poder analizar la factibilidad de implementación en AML.
- En caso de ser posible la implementación del modelo, elegir un caso de estudio real, para ser representado mediante el Modelo del Producto.
- Desarrollar un sistema en el que se implemente el Modelo del Producto utilizando el caso de estudio como ejemplo, que sea capaz de almacenar la información tanto geométrica como no geométrica del caso de estudio.

1.3 Estructura de la tesis

A continuación se presenta la forma en que está distribuido el presente trabajo.

En el capítulo dos se presentan todos los antecedentes que son necesarios para lograr una mejor comprensión. Se comienza explicando de la Programación Orientada a Objetos, esto con el fin de homogenizar los conceptos sobre este tema; enseguida se da una breve introducción a lo que es ingeniería concurrente, para comprender con mayor claridad que es lo que se intenta hacer con los modelos de información; a continuación se ofrece una introducción a lo que es el lenguaje de programación AML; y finalmente se termina dando

de una introducción al Modelo del Producto de una forma sencilla para dar una primer idea de éste.

En el capítulo tres se hace una descripción del Modelo del Producto que servirá como base para la implementación dentro de AML. Así como la descripción del caso de estudio a utilizar como ejemplo.

En el capítulo cuatro se explica como se llevó a cabo la implementación en AML del Modelo del Producto descrito, considerando sólo algunas de las clases más representativas de este modelo.

Dentro del capítulo cinco se realiza una explicación de la forma de utilizar el sistema desarrollado haciendo uso del caso de estudio en que se podrá apreciar la forma en que éste cambia al cambiar los parámetros que sirven para describirlo así como la forma de almacenar estos datos.

Finalmente se presentan las conclusiones obtenidas en la realización de este estudio.

2 Antecedentes

2.1 Programación orientada a objetos.

La orientación a objetos promete mejoras de amplio alcance en la forma de diseño, desarrollo y mantenimiento del software ofreciendo una solución a largo plazo a los problemas y preocupaciones que han existido desde el comienzo en el desarrollo de software: la falta de código reutilizable y portable, código que es difícil de modificar, ciclos de desarrollo largos y técnicas de codificación no intuitivas.

Un lenguaje orientado a objetos ataca estos problemas. Tiene tres características básicas: debe estar basado en objetos, basado en clases y capaz de tener herencia de clases.

Dado que la programación orientada a objetos se basa en la idea natural de la existencia de un mundo lleno de objetos y que la resolución del problema se realiza en términos de objetos, un lenguaje se dice que está basado en objetos si soporta objetos como una característica fundamental del mismo.

El elemento fundamental de la programación orientada a objetos es, como su nombre lo indica, el ***objeto***. Podemos definir un objeto como *un conjunto complejo de datos y programas que poseen estructura y forman parte de una organización.*

Esta definición detalla varias propiedades importantes de los objetos. En primer lugar, un objeto no es un dato simple, sino que contiene en su interior cierto número de componentes bien estructurados. En segundo lugar, cada objeto no es un ente aislado, sino que forma parte de una organización jerárquica o de otro tipo.

2.1.1 Encapsulamiento y Ocultación.

Cada objeto es una estructura compleja en cuyo interior hay datos y programas, todos ellos relacionados entre sí, como si estuvieran encerrados conjuntamente en una cápsula.

Los objetos son inaccesibles, e impiden que otros objetos, los usuarios, o incluso los programadores conozcan cómo está distribuida la información o qué información hay disponible. Esta propiedad de los objetos se denomina ocultación de la información.

Las peticiones de información a un objeto, deben realizarse a través de mensajes dirigidos a él, con la orden de realizar la operación pertinente. La respuesta a estas órdenes será la información requerida, siempre que el objeto considere que quien envía el mensaje está autorizado para obtenerla.

El hecho de que cada objeto sea una cápsula facilita enormemente que un objeto determinado pueda ser transportado a otro punto de la organización, o incluso a otra organización totalmente diferente que precise de él. Si el objeto ha sido bien construido, sus métodos seguirán funcionando en el nuevo entorno sin problemas. Esta cualidad hace que la programación orientada a objetos sea muy apta para la reutilización de programas.

2.1.2 Organización de los objetos.

Los objetos forman siempre una organización jerárquica, en el sentido de que ciertos objetos son superiores a otros de cierto modo.

Existen varios tipos de jerarquías: serán *simples* cuando su estructura pueda ser representada por medio de un "árbol". En otros casos puede ser más compleja.

En cualquier caso, sea la estructura simple o compleja, podrán distinguirse en ella tres niveles de objetos.

- ***La raíz de la jerarquía***. Se trata de un objeto único y especial. Este se caracteriza por estar en el nivel más alto de la estructura y suele recibir un nombre muy genérico, que indica su categoría especial, como por ejemplo objeto padre, raíz o entidad.
- ***Los objetos intermedios***. Son aquellos que descienden directamente de la raíz y que a su vez tienen descendientes. Representan conjuntos o clases de objetos que pueden ser muy generales o muy especializados, según la aplicación. Normalmente reciben nombres genéricos que denotan al conjunto de objetos que representan, por ejemplo, ventana, cuenta, archivo. En un conjunto reciben el nombre de clases o tipos si descienden de otra clase o subclase.
- ***Los objetos terminales***. Son todos aquellos que descienden de una clase o subclase y no tienen descendientes. Suelen llamarse casos particulares, instancias o ítems porque representan los elementos del conjunto representado por la clase o subclase a la que pertenecen.

2.1.3 Estructura de un Objeto.

Un objeto puede considerarse como una especie de cápsula dividida en tres partes:

1. *Relaciones*. Permiten que el objeto se inserte en la organización y están formadas esencialmente por punteros a otros objetos.
2. *Propiedades*. Distinguen un objeto determinado de los restantes que forman parte de la misma organización y tienen valores que dependen de la propiedad de que se trate. Las propiedades de un objeto pueden ser heredadas a sus descendientes en la organización.
3. *Métodos*. Son las operaciones que pueden realizarse sobre el objeto, que normalmente estarán incorporados en forma de programas (código) que el objeto es capaz de ejecutar y que también pone a disposición de sus descendientes a través de la herencia.

2.1.3.1 Relaciones

Las relaciones entre objetos son los enlaces que permiten a un objeto relacionarse con aquellos que forman parte de la misma organización.

Las hay de dos tipos fundamentales:

- ***Relaciones jerárquicas***. Son esenciales para la existencia misma de la aplicación porque la construyen. Son bidireccionales, es decir, un objeto es padre de otro cuando el primer objeto se encuentra situado inmediatamente encima del segundo en la organización en la que ambos forman parte; asimismo, si un objeto es padre de otro, el segundo es hijo del primero.

En la Fig. 2.1. B es padre de D, E y F, es decir, D, E y F son hijos de B; en la Fig. 2.2, los objetos B y C son padres de F, que a su vez es hijo de ambos.

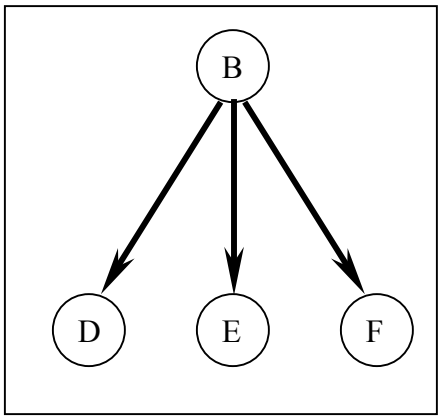

Figura 2.1 Jerarquía simple

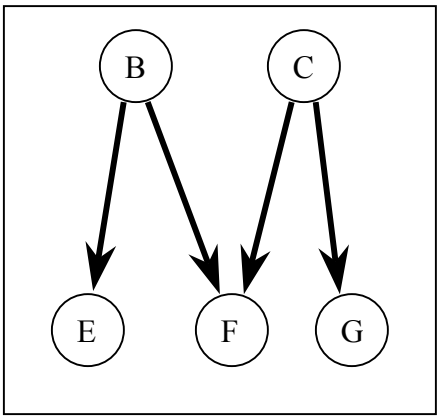

Figura 2.2 Jerarquía compleja

Una organización jerárquica simple puede definirse como aquella en la que un objeto puede tener un solo padre, mientras que en una organización jerárquica compleja un hijo puede tener varios padres.

- ***Relaciones semánticas***. Se refieren a las relaciones que no tienen nada que ver con la organización de la que forman parte los objetos que las establecen. Sus propiedades y consecuencias solo dependen de los objetos en sí mismos (de su significado) y no de su posición en la organización.

2.1.3.2 Propiedades.

Todo objeto puede tener cierto número de propiedades, cada una de las cuales tendrá, a su vez, uno o varios valores. En programación orientada a objetos, las propiedades corresponden a las clásicas "variables" de la programación estructurada.

Las propiedades se pueden heredar de unos objetos a otros. En consecuencia, un objeto puede tener una propiedad de maneras diferentes:

- ***Propiedades propias***. Están formadas dentro de la cápsula del objeto, es decir están definidas dentro del objeto.
- ***Propiedades heredadas***. Están definidas en un objeto diferente, antepasado de éste (padre,"abuelo", etc.). A veces estas propiedades se llaman "*propiedades miembro*" porque el objeto las posee por el solo hecho de ser miembro de una clase.

2.1.3.3 Métodos

Un método es una operación que realiza acceso a los datos. Un programa procedimental escrito en cualquier lenguaje, que está asociado a un objeto determinado y cuya ejecución sólo puede desencadenarse a través de un mensaje recibido por éste o por sus descendientes.

Si los métodos son programas, se deduce que podrían tener argumentos, o parámetros. Puesto que los métodos pueden heredarse de unos objetos a otros, un objeto puede disponer de un método de dos maneras diferentes:

- ***Métodos propios***. Están incluidos dentro de la cápsula del objeto.
- ***Métodos heredados***. Están definidos en un objeto diferente, antepasado de éste (padre,"abuelo", etc.). A veces estos métodos se llaman "*métodos miembro"* porque el objeto los posee por el solo hecho de ser miembro de una clase.

Los *demonios* son un tipo especial de métodos; relativamente poco frecuentes en los sistemas de programación orientada a objetos; que se activan automáticamente cuando sucede algo especial. Su ejecución no se activa mediante un mensaje, sino que se desencadena automáticamente cuando ocurre un suceso determinado: la asignación de un valor a una propiedad de un objeto, la lectura de un valor determinado, etc.

Los demonios, cuando existen, se diferencian de otros métodos por que no son heredables y porque a veces están ligados a una de las propiedades de un objeto, más que al objeto entero.

2.1.4 Polimorfismo

Es la posibilidad de construir varios métodos con el mismo nombre, pero con relación a la clase a la que pertenece cada uno, con comportamientos diferentes. Esto conlleva la habilidad de enviar un mismo mensaje a objetos de clases diferentes. Estos objetos recibirían el mismo mensaje global pero responderían a él de formas diferentes; por ejemplo, un mensaje "+" a un objeto ENTERO significaría suma, mientras que para un objeto STRING significaría concatenación.

2.2 Ingeniería concurrente

La definición más universalmente aceptada es la del reporte R-338 de la IDA (Institute for Defense Analysis), publicado en el verano de 1986, la cual define la ingeniería concurrente, como un esfuerzo sistemático para un diseño integrado, concurrente del producto y de su correspondiente proceso de fabricación y de servicio. Pretende que los desarrolladores, desde un principio, tengan en cuenta todos los elementos del ciclo de vida del producto, desde el diseño conceptual, hasta su disponibilidad incluyendo calidad, costo y necesidades de los usuarios.

Así pues, la ingeniería concurrente persigue un estudio sistemático, simultáneo, en el momento del desarrollo del producto, de las necesidades de mercado que va a cubrir, de los requisitos de calidad y costo en alcanzar, de los medios y métodos de fabricación, venta y servicio necesarios para garantizar la satisfacción del cliente en todo el ciclo de vida del producto.

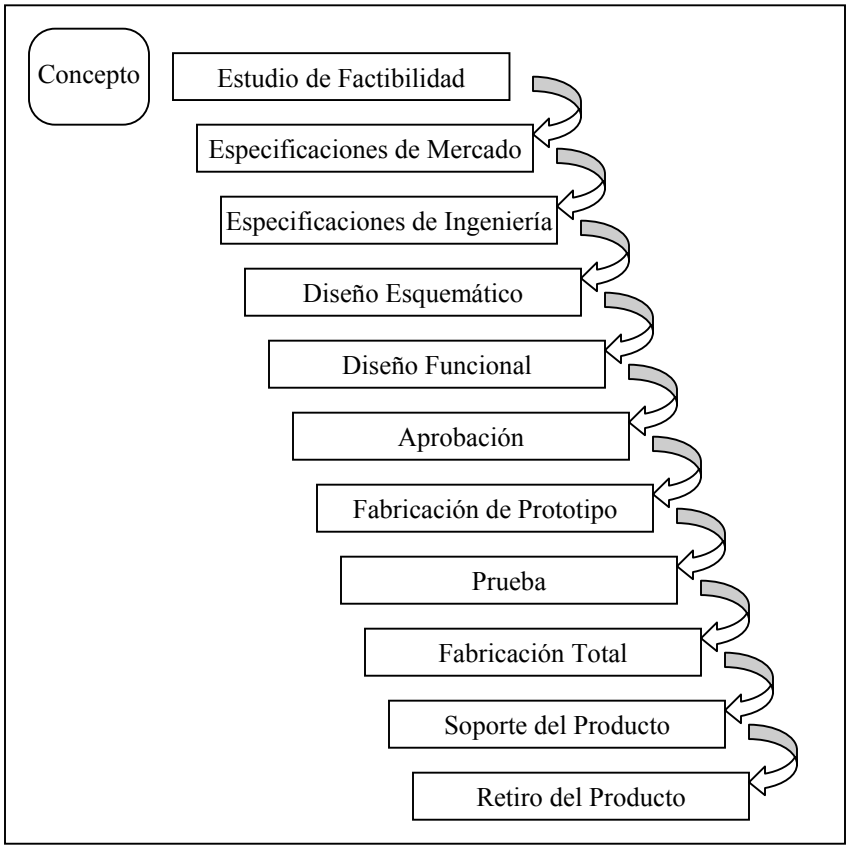

Figura 2.3 Ciclo de vida del producto, enfoque clásico o secuencial [7]

Precisa del trabajo coordinado y simultáneo de los diversos departamentos de la empresa: Mercadotecnia, Ingeniería del Producto, Ingeniería del Proceso, Producción, Calidad, Ventas, Mantenimiento, Costos, etc.

Sustituye el clásico entorno de trabajo en el desarrollo y fabricación del producto basado en un diagrama secuencial de actuación de los distintos departamentos, por un trabajo concurrente, simultáneo, en equipo, de todos a partir del mismo momento en que se inicia el proceso.

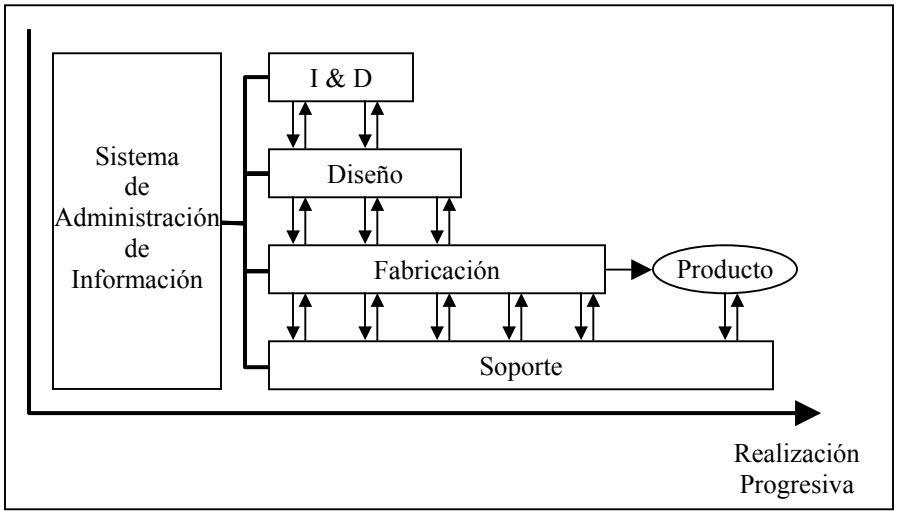

Figura 2.4 Ciclo de vida del producto, enfoque de ingeniería concurrente [7]

Esta metodología de trabajo recibe otros nombres:

- Ingeniería concurrente.
- Ingeniería simultánea.
- Equipos de diseño.
- Desarrollo integrado de producto.
- Ingeniería total.

De todos ellos escogemos el de ingeniería concurrente ya que este concepto pone más de manifiesto un esfuerzo común, una cooperación entre todos los agentes que intervienen.

La ingeniería concurrente es un nuevo enfoque, en pleno proceso de desarrollo, que incorpora una gran variedad de nuevas concepciones y metodologías de gestión de proyectos. Algunos de ellos son:

- DFF: Diseño para la función.
- DFM: Diseño para la manufactura.
- DFA: Diseño para el ensamble.
- DFQ: Diseño para la calidad.
- DFMT: Diseño para el mantenimiento.

Estas metodologías, y otras no citadas, pueden englobarse en dos orientaciones principales:

- Ingeniería concurrente en relación a la Productividad (Manufactura, Costo, Calidad, Comercialización)
- Ingeniería en relación al entorno (Ergonómica, Seguridad, Medio Ambiente, Reciclaje)

En ingeniería concurrente el producto debe ser programado para todo el ciclo de vida, debiendo, incluirse en todas las etapas posibles en el ciclo de desarrollo, como muestra la figura.2.5

En resumen los objetivos globales que se persiguen con la implementación de la ingeniería concurrente son:

1. Acortar los tiempos de desarrollo de los productos.
2. Elevar la productividad.
3. Aumentar la flexibilidad.
4. Mejor utilización de los recursos.
5. Productos de alta calidad.
6. Reducción en los costos de desarrollo de los productos.

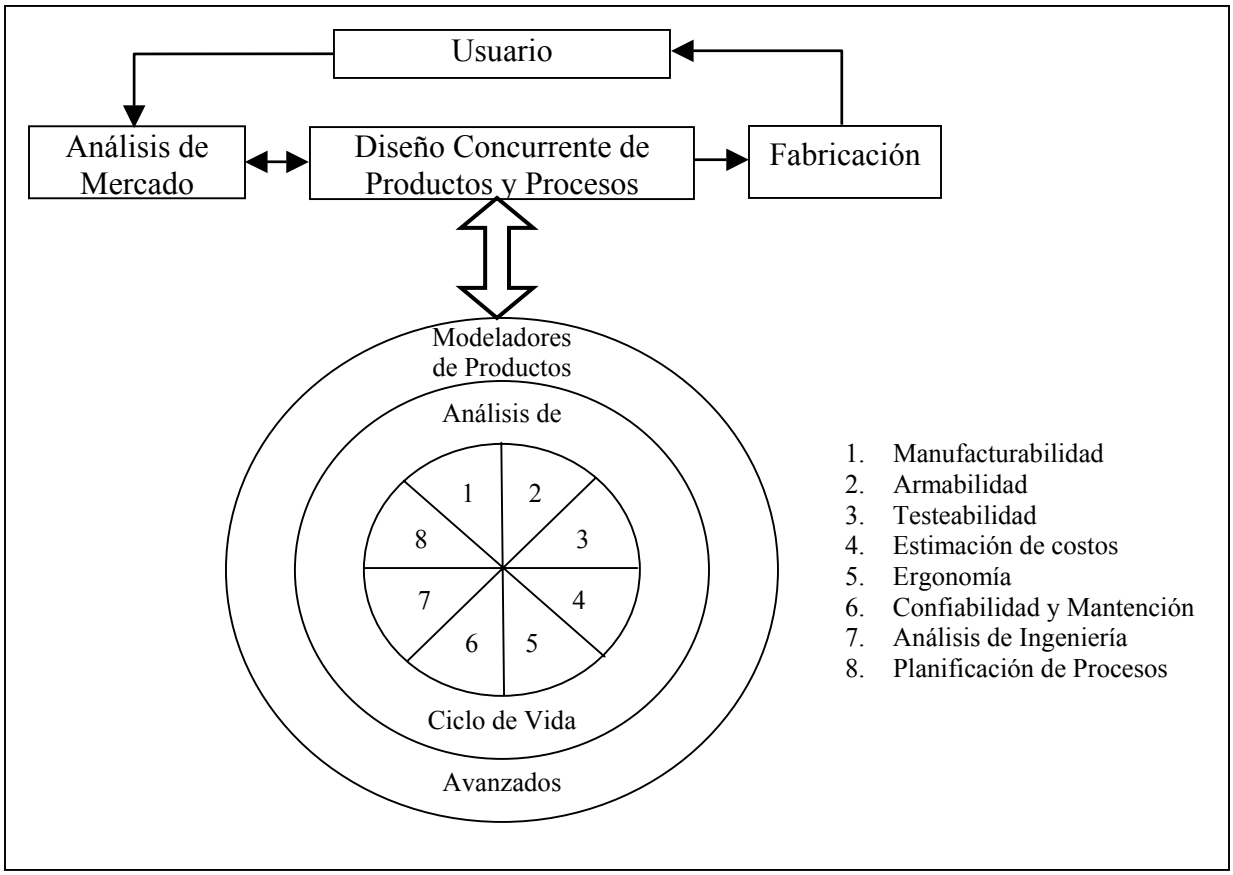

Figura 2.5 Ciclo de vida del producto usando el circuito de ingeniería concurrente [7]

2.3 Lenguaje de Modelado Adaptable (AML)

AML es un lenguaje de modelado orientado a ingeniería concurrente. Proporciona un prototipo para el modelado y organización del conocimiento de ingeniería requerido para integrar y automatizar ciclos enteros desde el diseño hasta la producción.

Este lenguaje está basado en los conceptos de la Programación Orientada a Objetos, en este paradigma de programación, los objetos interactúan con otros objetos y con ellos mismos; la diferencia es que en AML los objetos se pueden formar a partir de otros objetos.

Además tiene la capacidad de soportar funciones específicas que permiten al diseñador modelar un amplio rango de factores en un modelo sencillo. Por ejemplo, un modelo puede

visualizar los costos como propiedades de este objeto, esto le sirve al diseñador conocer que decisiones de diseño son afectadas por estos costos.

Algunas de las funciones soportadas por AML son:

- Completo soporte y portabilidad entre sistemas con plataformas UNIX y Windows.
- Soporta los formatos IGES / STEP (DXF).
- Soporte de diversos modeladores geométricos con completa compatibilidad de modelado.
- Constructor de Interfaz de Usuario nativo (GUI).
- Sintaxis común para diferentes módulos.
- Cálculo de dependencias en tiempo real.
- Objetos dinámicos .
- Modelado mediante alambre, superficies y sólidos .
- Dimensionamiento y acotamiento automático.

AML es un lenguaje que trabaja con una base de macros; es decir se logra que el programador diseñe un sistema con un lenguaje que le resulte muy familiar y descriptivo (esto con el fin de documentar lo menos posible el programa), y al momento de traducirlo o compilarlo[1] este código tan simple se expande para definir completamente el sistema. Es por esto que no se necesita declarar el tipo de variables que se utilizarán pues las reconoce automáticamente, para AML sólo existen los siguientes tipos de variables las cadenas (string), símbolos (symbol), listas (list), números (number) y objetos (object). Además las listas ya tienen los apuntadores de navegación implícitos al momento de ser creadas, es decir si quieres el primer elemento de la lista a, b, c; hay que dar esta instrucción si nos encontramos en la consola de AML:

AML>*(first '(a b c))*

[1] AML funciona con una combinación de compilación y traducción del código

Como se ha mencionado AML, utiliza conceptos de programación orientada a objetos como son las clases, objetos, herencia, por mencionar algunos; sin embargo nos introduce a conceptos nuevos como es el caso de los subobjetos (subobjects). Son precisamente los subobjetos los que logran que AML pueda generar nuevos objetos a partir de otros objetos definidos con anterioridad, y estos objetos nuevos a su vez pueden formar parte de otro objeto aún más complejo. Esta cadena puede seguir creciendo cuanto queramos, pero hay que tener cuidado de pasar los parámetros correctos de un subobjeto a un objeto para no perder el control de este.

AML funciona, como ya se mencionó, en plataformas UNIX y Windows NT. En ambientes UNIX, AML es soportado por Hewlett Packard, Sun, Silicon Graphics, e IBM y en PC's que están basadas en la plataforma Intel. Además las aplicaciones desarrolladas con AML son totalmente portables entre las dos plataformas, lo cual le da una gran flexibilidad para ser utilizado en ambientes de ingeniería concurrente.

2.3.1 Las capas virtuales

Una capa virtual o interfase virtual en AML es una interfaz de comunicación entre AML y alguna aplicación comercial para realizar algún tipo de análisis o desarrollo especifico en la que AML toma el control sobre esta aplicación. El modelo de la capa virtual construido en AML le permite al ingeniero enfocarse en resolver el problema, en lugar de los varios medios para resolverlo.

Como ejemplo se considera un diseño estructural y análisis de esté, donde para modelar un sólido se necesita crear un diseño, entonces la creación de una malla es necesaria para crear una malla de elemento finito, y finalmente un programa del análisis tendrá que generar una solución para esta. En cada fase hay varios programas y técnicas disponibles qué tienen su propia interfase y esto obliga al diseñador a conocer la sintaxis para cada una de estas herramientas.

El AML proporciona una interfase común a varios modeladores de sólidos además de los generadores para los diferentes tipos de malla y analizadores de elemento finito (FEA por sus siglas en inglés). Este se logra mediante las capas virtuales proporcionando una interfase común a estos. Otra ventaja que proporcionan las capas virtuales son: para un problema se pueden usar estrategias de solución diferentes. Por ejemplo, el mismo diseño geométrico podría crearse usando el AML SHAPES-based o el AML Parasolid-based como modelador de sólidos, y comparar exactitud y desempeño, o importar un archivo generado en ProEngineer o en otros sistemas de CAD.

Podrían crearse las capas virtuales similares encima de los otros componentes de AML asegurando la compatibilidad con los sistemas existentes, así como una interfase común para poder accederlos.

2.3.2 Arquitectura del sistema

El sistema consiste en varios módulos de AML (los conjuntos de clases y métodos) relacionando a los diferentes dominios de conocimiento cada uno realiza una función diferente como se ilustra en la figura 2.6 Todos los módulos han sido escritos utilizando el AML aunque estos se comuniquen con los programas externos a través de las capas virtuales. Los módulos adicionales pueden definirse y cargarse en AML para adaptar el lenguaje para un propósito específico. Como AML es modular, sólo los sistemas necesarios son cargados. Esto es, si un problema requiere el modelado de un armazón pero ningún gráfico, solo se cargaran los módulos necesarios para la construcción de ese armazón.

Las aplicaciones que requieren de diferentes aspectos del sistema construidas en AML, utilizan una interfaz de usuario común para el sistema. De este modo el usuario solo tiene que familiarizarse con una interfaz que es independiente de la aplicación; diseño, análisis, manufactura, e inspección, todas utilizan una interfase de AML común.

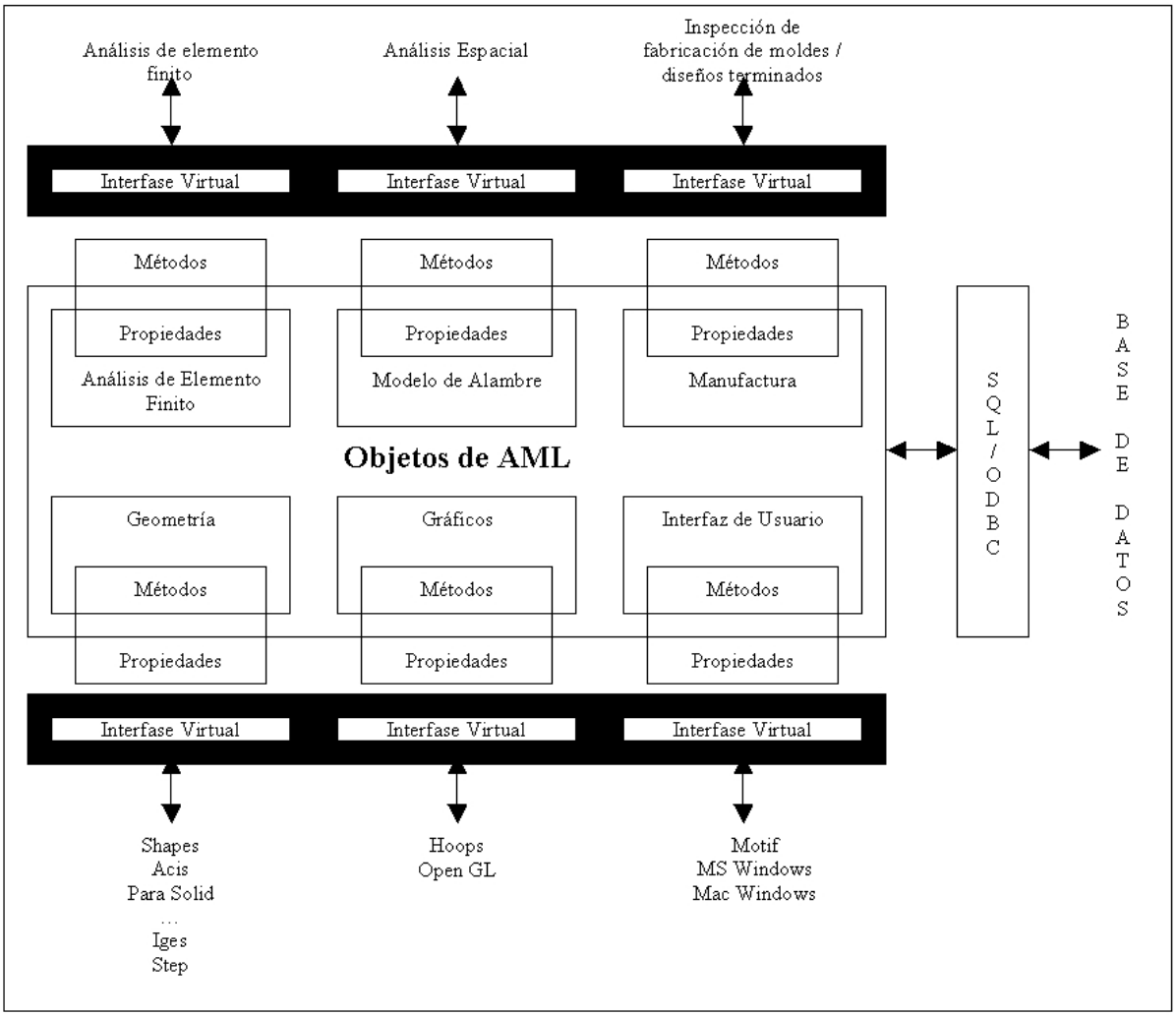

Figura 2.6. Arquitectura de AML [3]

2.3.3 La geometría

Las varias aplicaciones de AML incluso el CAD, layout y configuración, CAM, y FEM / FEA, tienen requisitos geométricos diferentes. Estos requisitos individuales son satisfechos a través de la capacidad de aumentar partes al modelo para satisfacer las diversas demandas de las varias aplicaciones, estas representaciones diferentes se manipulan a través de un modelo unificado. AML presenta varias clases/objeto para modelar primitivas simples y operaciones geométricas complejas. Las operaciones complejas son para hacer una combinación booleana, sólidos/superficies/alambres es decir; mediante primitivas (formas

geométricas básicas) e instrucciones complejas (operaciones entre estas geometrías) se pueden construir formas más complejas.

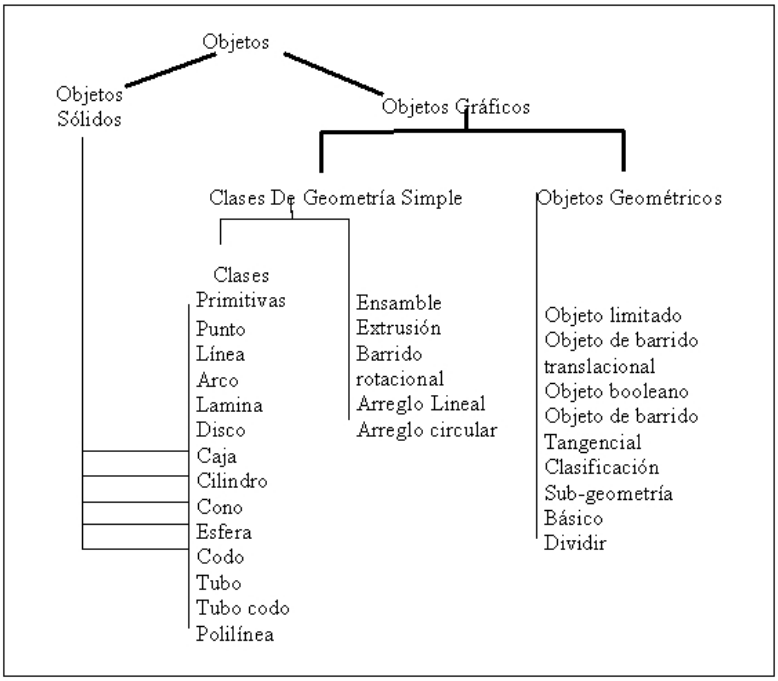

Figura 2.7 Capas geométricas de AML [3]

2.3.4 La Interfase del usuario

AML proporciona un juego completo de clases de la interfase incluyendo las formas, los botones de opción, casillas de verificación, formas de entrada; explotando así los menús soportados por las plataformas Unix/Motif y Windows/NT.

En resumen AML ofrece un ambiente de modelado flexible que puede utilizarse en el desarrollo de problemas de ingeniería. AML habilita la abstracción del modelado dentro de un conjunto de entidades entrelazadas que pueden modelarse en diversos escenarios. Estos problemas pueden tener una gran cantidad de atributos que se unen a través de una restricción unidireccional no cíclica del gráfico. AML también puede aplicarse a problemas que requieren un alto grado de visualización de entidades. También pueden satisfacerse aplicaciones que requieren de operaciones geométricas complejas.

2.4 Modelo del Producto.

Dentro del desarrollo de un nuevo producto intervienen diversos procesos para que el producto esté totalmente terminado, entre estos los procesos de ingeniería usan y crean información continuamente, mucha de esta información es generada por aplicaciones de software especializadas en algún tipo de análisis.

Debido a que esta información proviene de diversas aplicaciones, cada una de ellas tiene un determinado formato que solo puede ser reconocido por la aplicación que la creó; esto acarrea un grave problema de comunicación, cuando un proceso depende de la información de un proceso anterior y las aplicaciones para hacer los análisis son distintas.

Para resolver este tipo de problemas se han implementado algunas posibles soluciones que van desde el reproducir totalmente el modelo para cada una de las aplicaciones, generar protocolos de comunicación como IGES o el DXF para poder importar un modelo a otra aplicación; sin embargo, se presentó otro problema, no se podía asegurar que la versión en la que estaban trabajando los diferentes procesos era la misma, pues como la información residía en cada una de las máquinas, cada analista podría hacer modificaciones al modelo que no se verían reflejadas en las versiones anteriores del modelo.

Como respuesta a estas dificultades se idearon los sistemas integrados. Estos sistemas, a diferencia de los anteriores que utilizan las aplicaciones como centro de desarrollo del modelo; utilizan al modelo como el centro del desarrollo, es decir; existe una única fuente de información a la cual deberán acudir todas las aplicaciones para poder obtener la información necesaria para llevar a cabo su análisis.

Este tipo de sistemas presenta diversas ventajas como evitar que exista información duplicada, si en algún momento el modelo cambia los otros procesos podrán ver cual fue el aspecto que cambio, asegurando que todos trabajen con la misma versión del modelo, entre

otras. Cabe mencionar que este tipo de sistemas trabajan con una base de datos central a la cual tienen acceso todas las aplicaciones que se requieran.

El Modelo del Producto se apega a este tipo de sistemas, el propósito del modelo es describir completamente un producto tanto en sus características geométricas como no geométricas además de añadir otros aspectos relevantes como son la intención del diseñador, observaciones del diseñador, además de poder ampliarse a un posible Modelo de Manufactura y otros modelos.

Según Krause[2] una tecnología de modelado de producto debe cumplir los requisitos básicos siguientes:

- Información actualizada: Toda la información debe estar disponible en las diferentes fases del proceso y ser la misma permitiendo modificarla cuando sea necesario.
- Facilitar la documentación del producto: guarde resultados intermedios y finales.
- Apoyar la representación, exploración de proceso y alternativas del producto para reducir iteraciones costosas y flexibilidad de producto.

De acuerdo a van Der Net[3] un Modelo del Producto debe satisfacer tres necesidades básicas:

- Creación de una descripción completa del producto para todas las fases del diseño.
- Permitir la captura y el almacenamiento de los "intentos de diseño."
- Habilitar el análisis de manufactura en el diseño.

[2] Krause, F.L., Kimura, F., Kjellberg, T. 1993, Product Modelling. Annals of the CIRP
[3] van Der Net, A.J., De Vries, W. A. H., Delbressine, F. L. M. 1996,Relation-Based Oroduct Model for Integrating Design and Manufacturing Annals of CIRP

3 Desarrollo del Modelo del Producto

3.1 Modelo del Producto

3.1.1 Modelos de Información

Un Modelo es una representación abstracta de un fenómeno físico, con la finalidad de entenderlo. En la ingeniería, los modelos son de vital importancia, pues con ellos se debe trabajar para conocer y predecir el comportamiento de un sistema (entiéndase por sistemas un conjunto de elementos que interactúan entre si para conseguir un objetivo especifico). En el caso de la ingeniería en computación ayuda a tener una idea como será desarrollado el sistema y como se comunicará tanto con los diversos módulos que lo componen como con el exterior (el usuario, sistemas de control, etc.).

Según la Organización de Estándares Internacionales (ISO) un modelo es una representación de una entidad o sistema, describiendo solamente los aspectos que son relevantes en el contexto del propósito.

Un modelo de información es un modelo que ayuda a representar las características relevantes de un sistema, así como su comportamiento y los procesos involucrados en dicho sistema.

Molina[1] ha clasificado los métodos para la asistencia del modelado de información en cuatro grupos principales:

- Método de Modelado de datos. Describe la información relevante para un sistema.
- Método de Modelado de procesos. Describe como ciertas actividades son llevadas a cabo.

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

- Método de Modelado de comportamiento. Describe como se comporta el sistema a través del tiempo.
- Método de Modelado híbrido. Describe un sistema mediante una combinación de los métodos anteriores.

Todos estos métodos arrojan "modelos de información", solo que cada uno ellos está especializado en un área en particular, a excepción del híbrido que pretende equilibrar de manera óptima todos los anteriores.

El Modelo del Producto que será utilizado en esta tesis, ha sido desarrollado en base a las investigaciones que se han realizado en el proyecto MIM (Manufacturing Information Models –Modelos de Información para Manufactura). Este proyecto da un mejor entendimiento del Modelo de Manufactura y una nueva forma de estructurar los datos. En este proyecto se han generado dos principales líneas de investigación: una orientada al maquinado y procesos de manufactura, la otra se refiere a actividades relacionadas con el ensamble. Algunos de los casos de estudio se han citado ya en el parte introductoria de esta tesis.

En la figura 3.1 se muestra a grandes rasgos la estructura tanto del modelo del Producto como el de Manufactura que el Dr. Jesús Manuel Dorador utilizó para realizar diseños de ensamble y planeación de procesos de ensamble.

En esta tesis únicamente se utilizará el Modelo del Producto, para ser implementado en AML, dejando el desarrollo del modelo de Manufactura y Procesos de Ensamble para un estudio posterior.

Se ha elegido trabajar en este modelo debido a la fácil comprensión de su estructura y a que se puede contar con la asesoría del autor para su interpretación, además de que es muy fácil relacionarlo con la realidad.

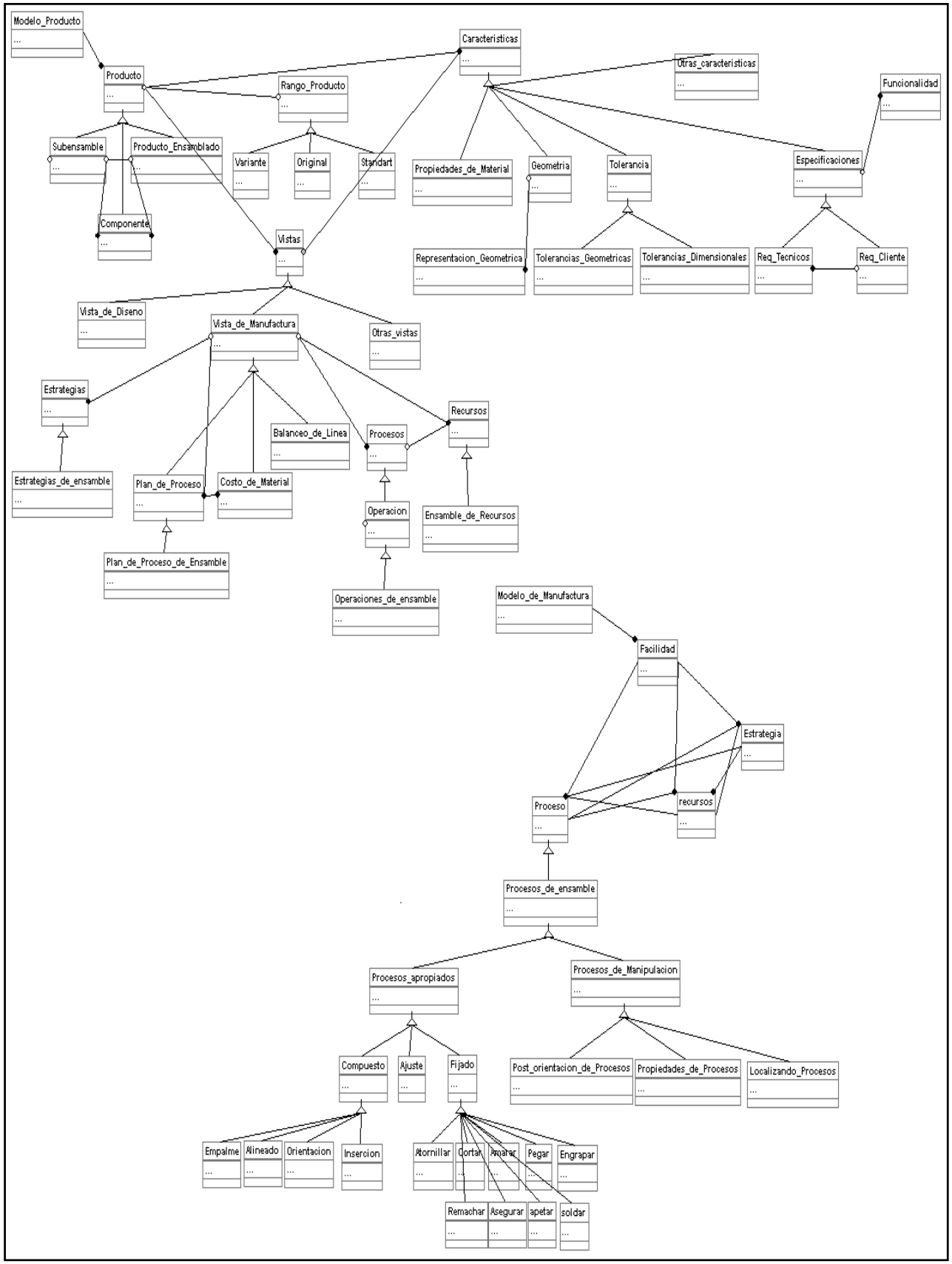

Figura 3.1 Modelo del Producto y Modelo de Manufactura [2]

3.1.2 Descripción del Modelo del Producto

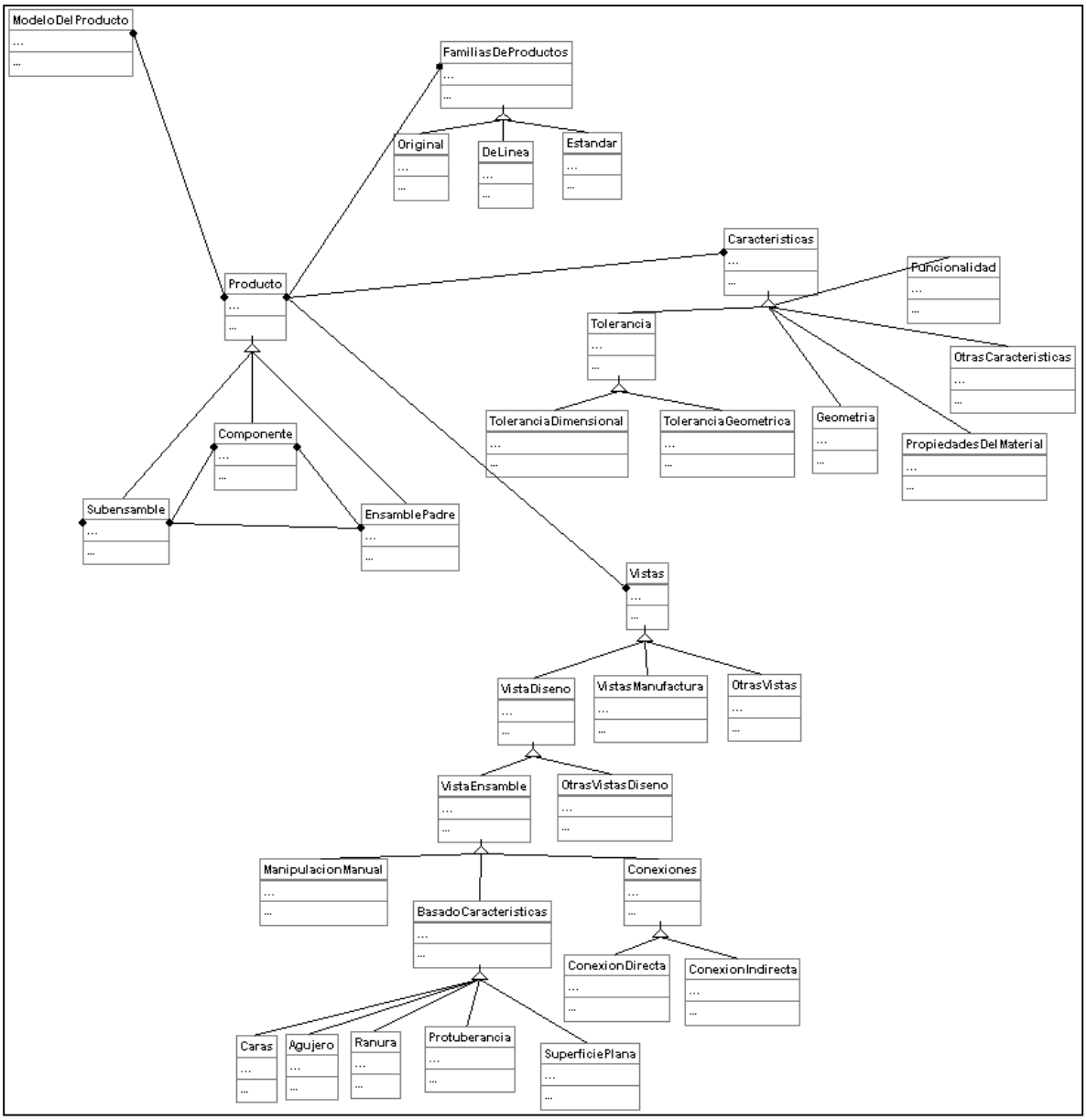

Figura 3.2 Modelo del Producto [2]

En la figura 3.2 se muestra el Modelo del Producto simplificado que se pretende implementar en AML. En las secciones posteriores se describirá de manera general la composición de este modelo.

Como su nombre lo indica el Modelo del Producto contiene la información de un producto; con toda esta información debe poderse hacer una descripción completa de un producto es decir, debe contener además de la información geométrica, información sobre tolerancias, propiedades de material, funcionalidad y otras características; entre estas características se encuentra la intención del diseñador, entre otras.

Para que pueda manejar diversos tipos de productos y cada uno de estos productos tenga la información que lo describa, el Modelo del Producto tiene una clase que se encarga de identificar y almacenar cada uno de estos productos, esta clase es mostrada en la figura 3.3 de lo que se puede entender que el modelo del producto puede contener muchos productos pero un producto puede ser descrito por un solo Modelo del Producto; lo escrito anteriormente se deduce de la forma en que se lee un esquema escrito en UML. Para mayor referencia sobre UML consultar el Apéndice B.

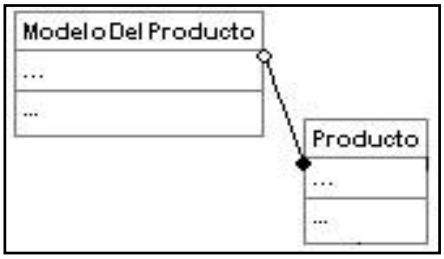

Figura 3.3 Relación entre el producto y el Modelo del Producto [2]

3.1.2.1 Producto

Expandiendo la clase Producto se puede ver que tiene tres clases hijas como se muestra en la figura 3.4.

Estas clases hijas, en la forma en que están relacionadas, describen como está formado un producto, la forma de interpretar la figura 3.4 seria la siguiente: un componente puede estar formado ya sea por un componente o por un Subensamble o por un Ensamble Padre (ensamble principal); ahora un Ensamble Padre a su vez puede estar formado por Subensambles y/o Componentes, de igual forma un Subensamble puede estar formado por

otros Subensambles y/o Componentes, de esto se puede entender que la entidad más simple que puede formar a un producto es el Componente y que un producto por muy complicado que sea puede ser descrito por estas tres únicas clases.

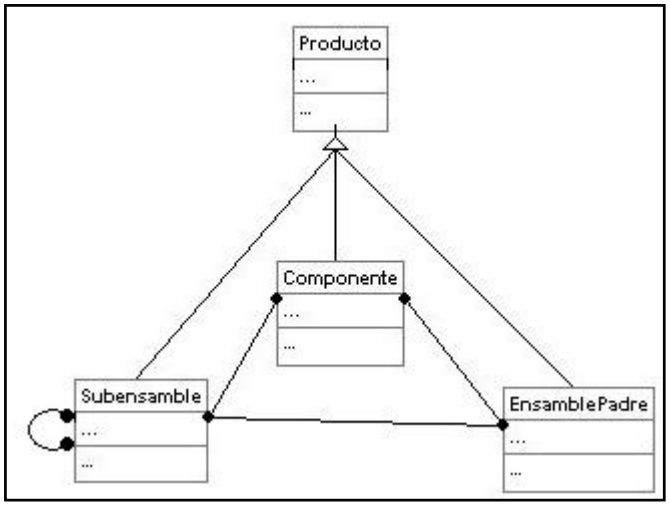

Figura 3.4 Detalle sobre la clase PRODUCTO [2]

Esta forma de describir un producto además de permitir la descripción de productos muy complejos, ayuda a llevar una secuencia en la construcción de dicho producto, esta construcción puede ser real o virtual (generada por computadora).

3.1.2.2 Familias de Productos

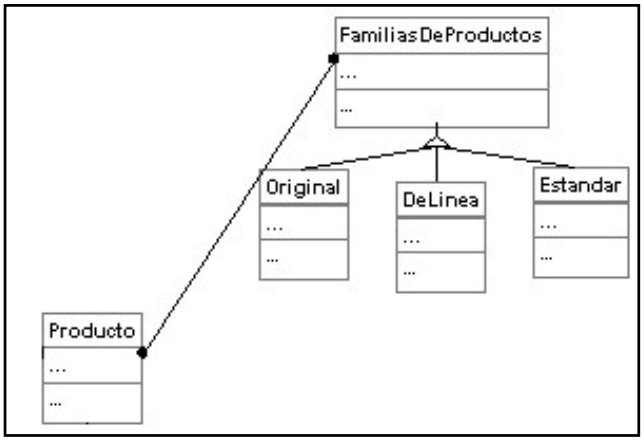

Figura 3.5 Detalle de la clase FAMILAS DE PRODUCTOS [2]

La concepción de familias de productos dentro del Modelo del Producto se ha llevado a cabo con el fin de clasificar los productos que tengan características similares; esto ocasiona que los productos se puedan ver desde clasificar con mayor facilidad y por tanto su búsqueda es mucho más sencilla y rápida.

La clasificación de estas familias se ha hecho de la siguiente forma: Original, de Línea y Estándar.

Una familia de productos Estándar, se refiere a todos aquellos productos existentes en el mercado y pueden ser utilizados sin que se les tenga que hacer ninguna modificación.

Una familia de productos de línea o también llamados Variantes, describe a todos aquellos productos que provienen de un producto estándar, pero que han sido modificados en algunas de sus características para poder adecuarlos a las necesidades del cliente.

Una familia Original se refiere a un producto totalmente nuevo que ha sido diseñado de acuerdo a las especificaciones del cliente, ya que no existe en el mercado un producto estándar o modificado que satisfaga sus necesidades.

Múltiples vistas en el Modelo del Producto

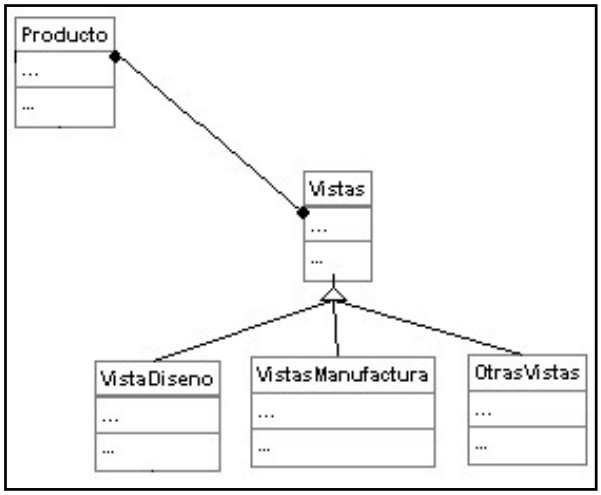

Figura 3.6 Detalle de la clase VISTAS [2]

Las vistas en el Modelo del Producto, se refiere a las diferentes formas de representar un producto, esto de acuerdo a quien va dirigida la información, es decir la información que se requiera en el proceso de ensamble puede no ser la misma que se requiera en el proceso de manufactura o en el de diseño. Con las vistas se intenta que llegue a los diferentes procesos solo la información que requiere, ni mas ni menos, esto con el fin de evitar confusiones en las diferentes etapas del ciclo de vida del producto.

3.2 Descripción del Caso de Estudio.

Para hacer la representación del Modelo del Producto se ha elegido la construcción de un mueble para cocina. En principio este mueble consta de un conjunto de 3 charolas que pueden ser colocadas a la distancia deseada, el mueble es desarmable, a excepción de los esquineros que están soldados, esta elaborado en su totalidad de acero inoxidable, la descripción de las partes que constituyen a este mueble se hace a continuación:

Para cada una de las charolas se tienen:

- Cuatro ensambles de esquineros

Esquinero L	Esquinero M	2 refuerzos
		

Figura 3.7 Componentes del Esquinero

Todos estos componentes han sido soldados como se muestra en la figura 3.8, formando de esta manera él esquinero para la charola.

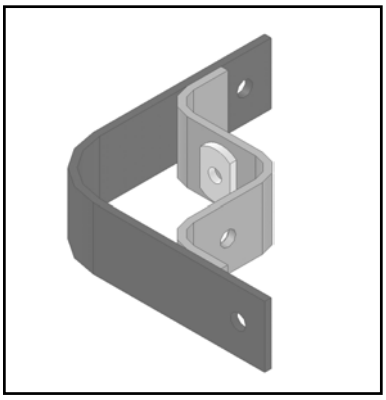

Figura 3.8 Ensamble de Esquinero

- Lámina recortada y doblada de la forma mostrada en figura 3.9.

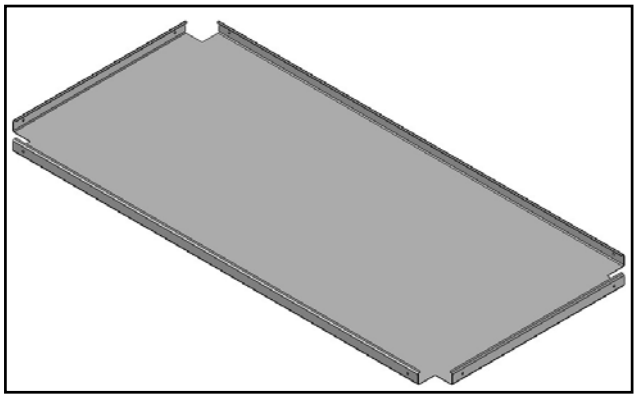

Figura 3.9 Lámina Que Formará La Base De La Charola

- 8 tornillos estándar, 8 tuercas y 8 rondanas para la fijación de los cuatro esquineros en la charola de forma mostrada en la figura 3.10.

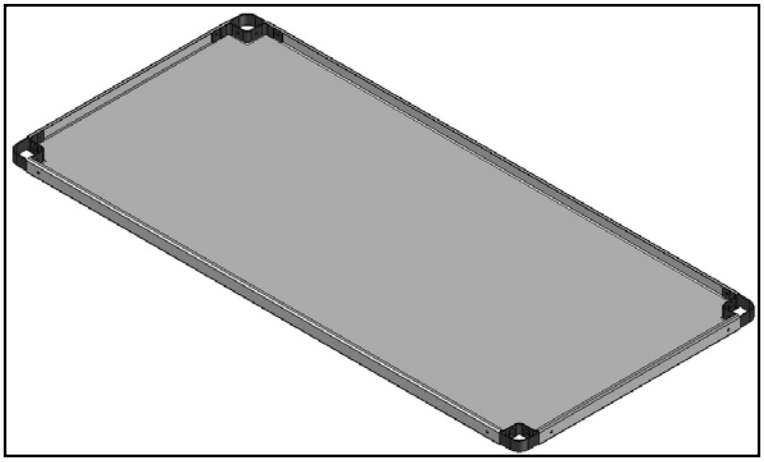

Figura 3.10 Ensamble De La Charola

Para el ensamble del mueble se utilizan:

- 4 tubos de la altura requerida, tomando en cuenta la altura de las ruedas del mueble y cerrados en uno de los extremos.
- 4 ruedas.
- 24 tornillos para la fijación de las charolas en los tubos (8 por cada charola).

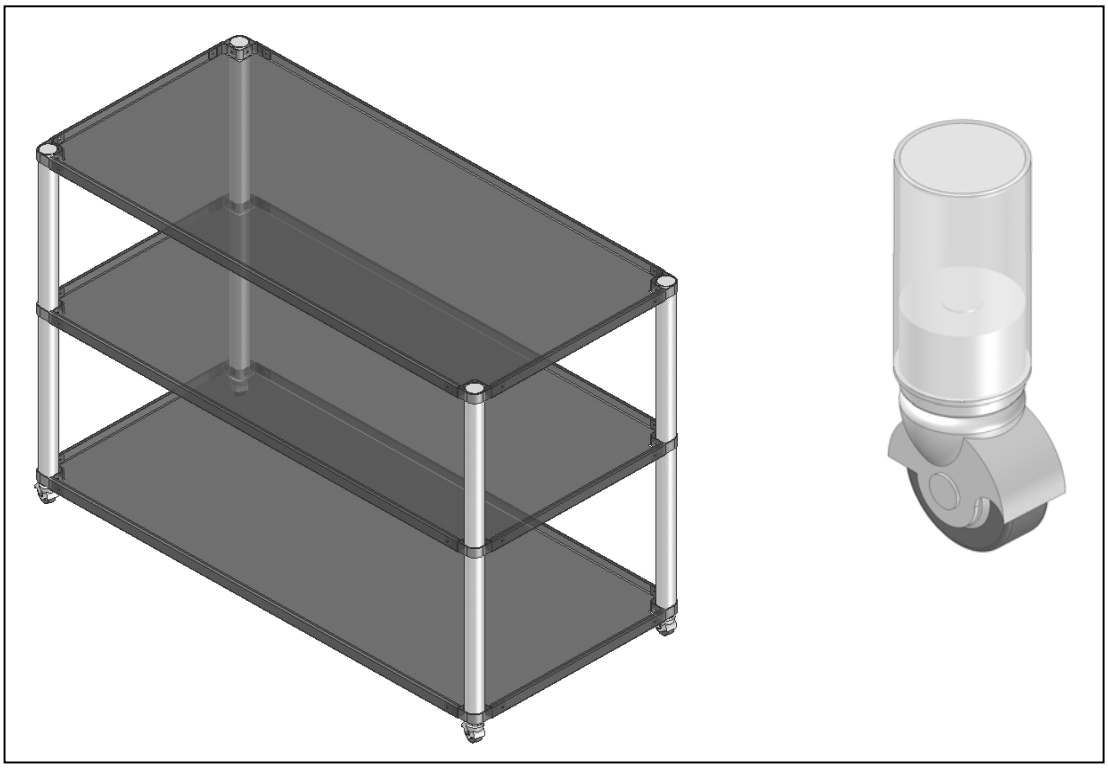

Figura 3.11 Ensamble del caso de estudio y detalle del tubo de soporte

El mueble descrito anteriormente servirá como referencia para desarrollar el modelo del Producto. Se le pretenden hacer algunas modificaciones, como colocar las tapas a los tubos de material plástico, poder agregar el número de charolas dependiendo de la altura del tubo y el espaciamiento entre ellas, tener la opción de colocar pestañas a la charola, tener la opción de colocar ruedas o bases fijas, entre otras. Esto con el fin de hacerlos lo más general posible, para que así se puedan generar después familias de este tipo de productos.

La construcción del mueble en AML se hará tratando de seguir el proceso de ensamble, es decir, primero se han de crear los componentes básicos haciéndolos paramétricos a los valores principales del mueble como son: el alto, largo, ancho, etc.

Una vez creados todos los componentes serán colocados en los sitios y orientaciones correspondientes para formar el mueble completo, a este mueble se le podrán asignar además de las características geométricas, propiedades de material, tolerancias, especificaciones del cliente, intención del diseñador, entre otras características. Toda esta información se deberá poder almacenar en tablas creadas con ayuda del propio AML, esto con el fin de simular una base de datos.

4 Implementación de Modelo del Producto Usando AML

Hasta el momento se han descrito a grandes rasgos las características del Modelo del Producto así como el modelado en AML. En la presente sección se describirá como se ha adecuado este modelo del producto a las capacidades y restricciones que presenta AML, siendo aplicadas en el caso de estudio descrito con anterioridad.

4.1 Características de Modelado en AML

A diferencia de otras herramientas de modelado, AML no ofrece un ambiente gráfico de desarrollo, sino que todo el proceso en el modelado se realiza mediante la escritura de código fuente. Esto puede dificultar el poder visualizar lo que se está modelando, cuando se está empezando a usar AML, sin embargo ésta deja de ser una dificultad conforme el programador se acostumbra a usar el lenguaje.

Como es de suponerse en este lenguaje todas las construcciones se llevan a cabo utilizando las clases, métodos y funciones existentes, sin embargo el programador puede crear sus propias clases, métodos, funciones o bien redefinir las existentes.

La técnica de desarrollo de un modelo no difiere en mucho de las utilizadas en otros sistemas de modelado, primero se tienen que crear piezas simples para poder formar ensambles o relaciones entre las piezas, la dificultad de está radica en que las formas primitivas con las que cuenta AML se vuelven cada vez más difíciles de definir debido a que en algunas de ellas se requieren de vértices para definir el contorno de la pieza, y aunado a que se carece de una representación gráfica que nos ayude a visualizar el contorno que se dibuja, ocasiona que algunas veces se tengan resultados poco satisfactorios, pero como se mencionó antes, es cuestión de familiarizarse con el sistema. Es por esto que se sugiere que se cuente con un plano o un croquis que ayude a la visualización que se desea modelar.

Otro aspecto importante es que las piezas se deben diseñar de la forma más general posible, es decir, que se desarrollen pensando en que pueden ser utilizadas en otra parte del modelo o incluso en otro modelo, y solo se tendrían que modificar las medidas de está.

4.1.1 Parametrización

Con el concepto de Parametrización quiere decir que el modelo debe ser totalmente definido con un mínimo número de medidas o variables independientes; estas variables serán las que definan tanto a las demás dimensiones de la pieza como a la posición del sistema coordenado de está y su posición dentro del espacio o el ensamble. En la figura 4.1 se muestra esta relación entre las dimensiones, y la posición de la pieza.

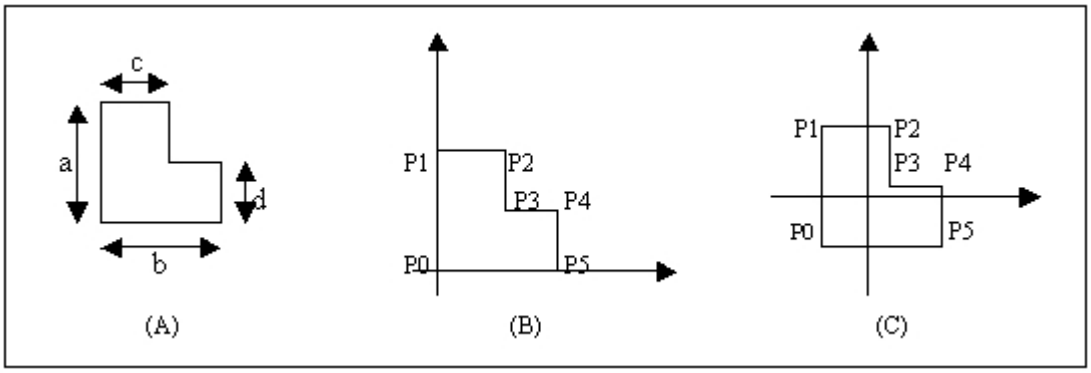

Figura 4.1 Algunas posibles localizaciones de un mismo perfil en AML

En la figura 4.1(A) se muestra el esquema del perfil que deseamos crear y los valores de las dimensiones mínimas para definirlo. En la figura 4.1(B) se representa el perfil en una de sus posibles posiciones, los puntos P0, P1, P2, P3, P4 y P5 son los vértices que servirán para unir las rectas del perfil, las coordenadas de estos puntos están referenciadas en base a las dimensiones del perfil y al origen del sistema coordenado, por ejemplo: P0 (0,0), P3 (c,d), P4 (b,c), etc. La figura 4.1(C) muestra el mismo perfil localizado en diferente posición, como se puede observar en este caso resulta más complicado definir los puntos en base a la dimensiones ya que implica hacer operaciones sobre ellas para obtenerlos.

Con esto se quiere ejemplificar que tan complicado se puede volver el diseñar un solo perfil en AML, es por eso que se recomienda generar las piezas preferentemente en la posición (no es la posición de la parte en el modelo completo) y orientación en la que serán utilizadas, esto con el fin de facilitar su ubicación en el momento de ser llamadas en el modelo completo.

Una de las ventajas que se presentaron en AML al modelar un producto es que gracias a los subobjetos se puede representar la parte del Producto (Componente, Subensamble y Producto Ensamblado) al momento de ir programando el modelo, ya que estos subobjetos representarían los componentes o subensambles, y el producto ensamblado sería la última clase desarrollada para la representación gráfica del modelo. La Figura 4.2 ayudará a ver de forma más clara lo expuesto anteriormente.

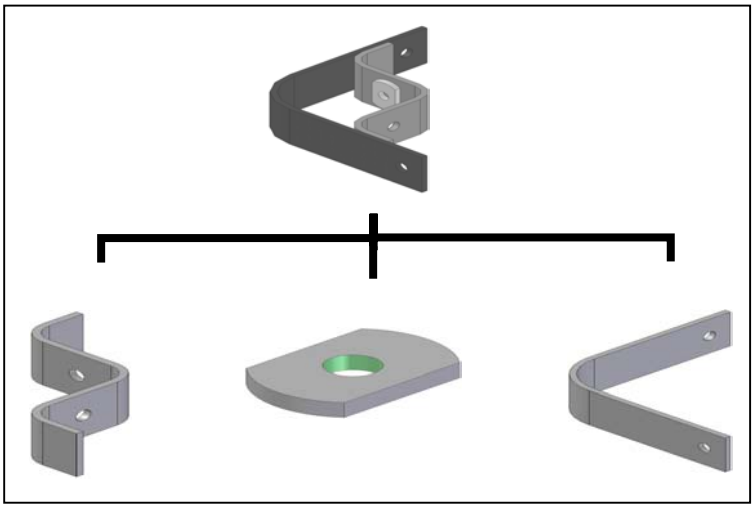

Figura 4.2 Subobjetos de una clase en AML

En la figura 4.2 se está mostrando el ensamble de un esquinero. En AML este esquinero sería representado por una clase "esquinero" con ciertas características (dimensiones, materiales, características no geométricas) que dependen de las clases de las que hereden, estas características son llamadas en AML propiedades, es mediante estas propiedades como se definirá tanto el esquinero como la posición de los elementos que lo forman. Los tres elementos mostrados bajo el esquinero son los componentes de este ensamble cada uno con sus propias características, a estos componentes es a lo que se conoce como subobjetos.

Las características de éstos subobjetos pueden ser alternadas haciendo que tomen el valor ya sea de una de las propiedades del ensamble o de las operaciones entre estas; lo mismo sucede con la posición de cada uno de los componentes en el espacio 3D, esta puede ser referida de acuerdo con los valores de las propiedades del ensamble.

A su vez si se requiere que este ensamble sea utilizado como subensamble para otro ensamble, lo único que hay que hacer es agregarlo como un subobjeto para la clase que representará el nuevo ensamble. De aquí que el ensamble final sea la clase que utiliza a todas las piezas; ya sea que estén dentro de una clase del tipo subensamble o como una clase del tipo componente; y que no es usada para formar otro ensamble.

Otras formas de utilizar los subobjetos es mediante la creación de piezas por medio de operaciones booleanas como son intersección, unión, etc. Este tipo de operaciones solo se puede hacer entre dos componentes a la vez y si se requiere operar con más de dos elementos se tiene que hacer mediante parejas; cabe resaltar que el resultado depende del orden en que se realicen las operaciones y los elementos relacionados en éstas. Un ejemplo de operaciones booleanas con más de dos operadores se ilustra en la figura 4.3.

Las ilustraciones 4.3(A), 4.3(B) y 4.3(C) representan en este caso las partes que formarán la pieza requerida. Las figuras 4.3(D) y 4.3(E) muestran los resultados de las operaciones de unión y diferencia realizada en diferente orden; en la primera se hizo primero la unión entre A y B y al resultado se le quitó la parte C; mientras que en la segunda se hizo primero la operación de diferencia entre A y C y al resultado se le unió B. Estos son solo dos de los posibles resultados esperados utilizando las partes y operaciones mencionadas.

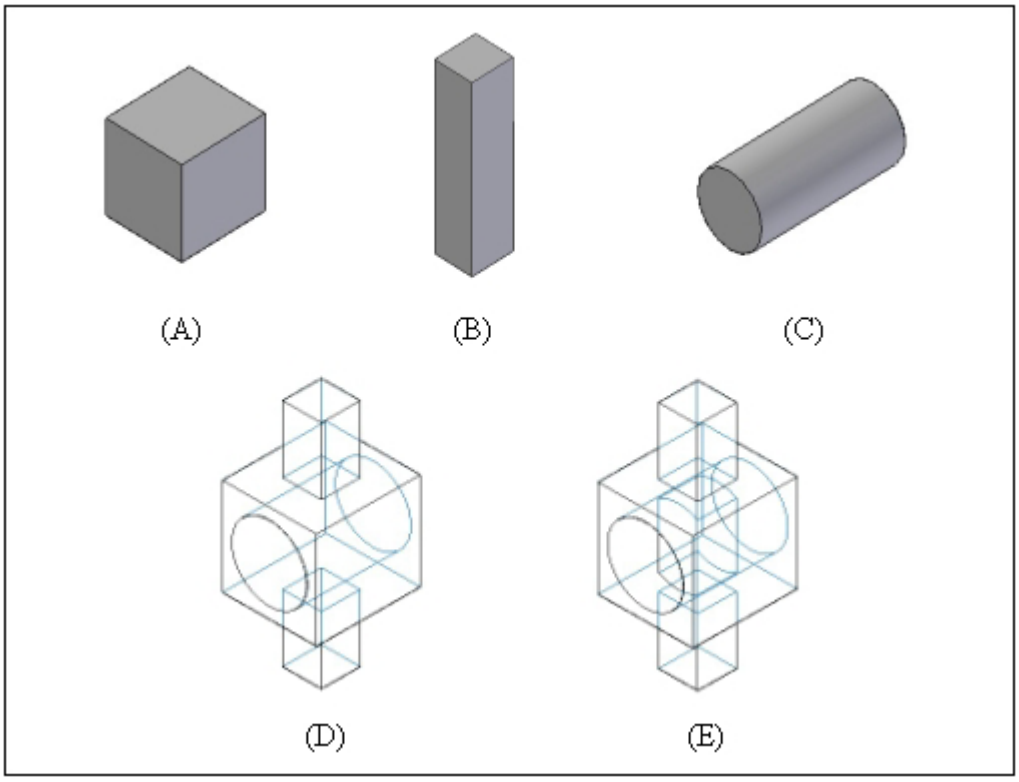

Figura 4.3 Operaciones booleanas en AML

4.2 Descripción de la implementación del Modelo del Producto usando AML

Para realizar la implementación del Modelo del Producto sólo se han considerado algunas clases de las presentadas en el capítulo 3 de esta tesis; donde se hizo la descripción del Modelo del Producto; y son presentadas en la figura 4.4.

Ahora bien la forma en que se implementarán estas clases, será en tres partes. La primera parte se encarga de la representación gráfica del modelo, es decir el aspecto visual que tendrá el modelo; la segunda parte se refiere a todos los aspectos que no estén involucrados directamente con la representación visual; y la tercera se encarga de la comunicación entre estas dos. La figura 4.5 muestra únicamente las dos primeras partes ya que la tercera solo se trata de una relación que es implementada en la programación en AML.

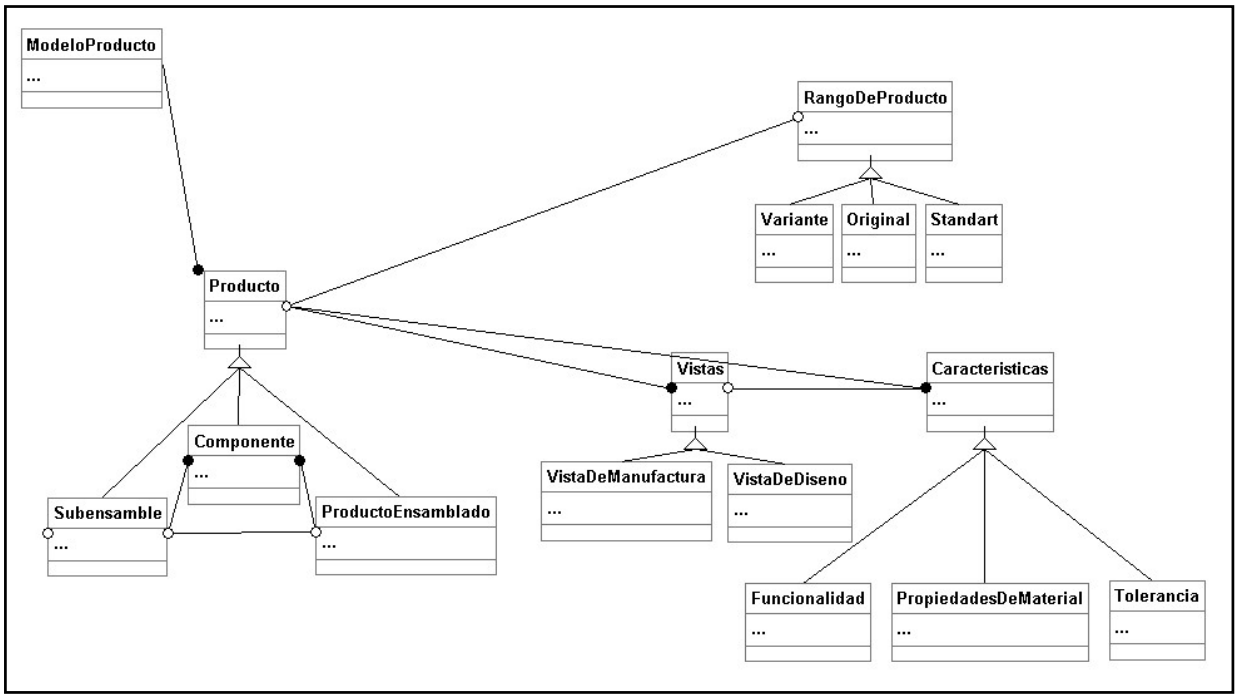

Figura 4.4 Modelo del Producto a ser implementado en AML

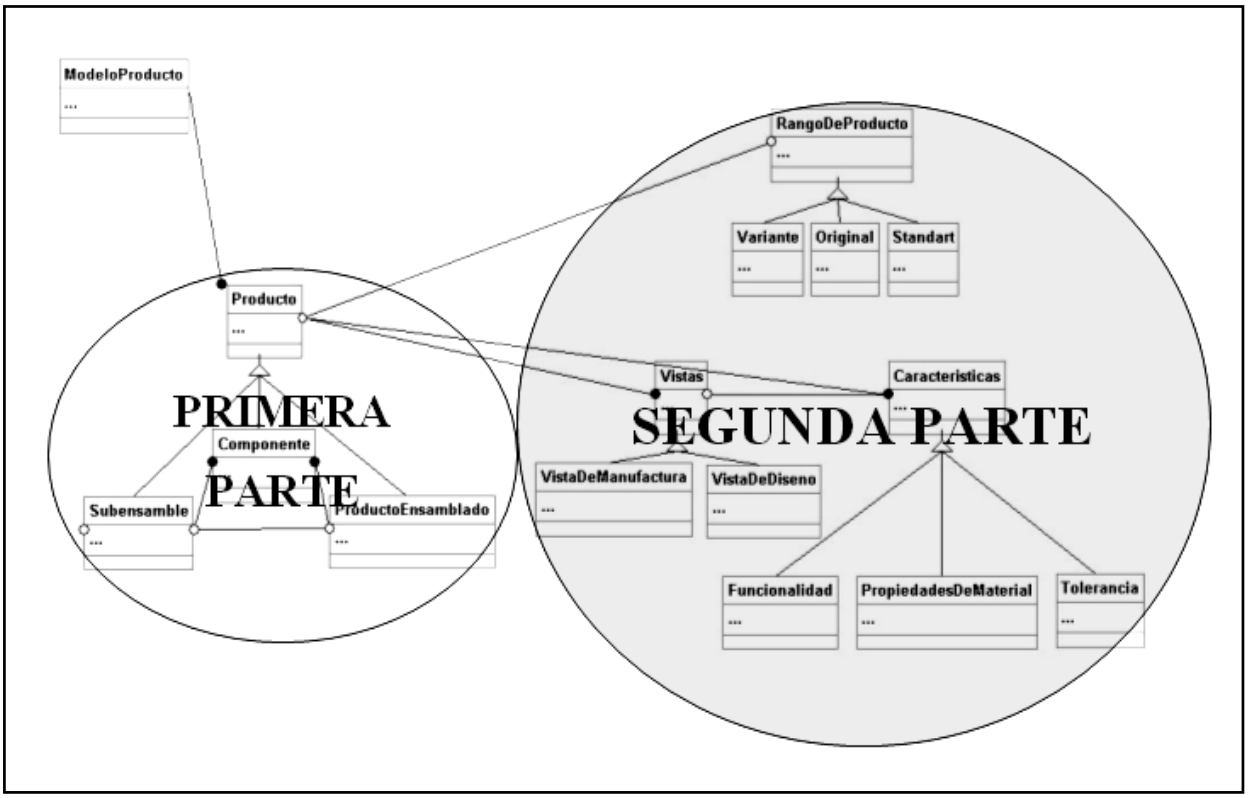

Figura 4.5 Secciones en que se implementará el Modelo del Producto

4.2.1 Implementación de la Primera Parte (Representación Visual)

La primera parte del modelo involucra todas las clases que dependen únicamente por una relación de herencia de la clase Producto, es decir solo contempla las clases: Componente, Subensamble, Producto Ensamblado y Producto. Esto se ha hecho con el fin de aprovechar la propiedad de los subobjetos que proporciona AML, es gracias a esta propiedad que al momento de programar un modelo, un subobjeto puede comportarse como un componente o un subensamble; la clase producto terminado, por la naturaleza de AML, se genera al programar el producto terminado, siendo sus subobjetos las partes o subensambles que se crearon con anterioridad.

A continuación se muestra parte del código del sistema en el que se ilustra con mayor claridad lo descrito anteriormente:

```
(define-class L-PERF
  :inherit-from (difference-object)
  :properties(
          espesor-l (default 3)
          largo-l (default 105)
          radio-tubo (default 20)
          ancho-l (default 30)
          d-tornillo (default 7.2)
          material-L (default 'acero-inox)
          object-list( list ^l-perf1 ^hoyo-y)
          )
  :subobjects(
          (l-perf1 :class 'difference-object
                  object-list(list ^base ^hoyo-x)
                  display? nil
                  )
          (base :class 'esq-l
                   espesor-l ^^espesor-l
                   largo-l ^^largo-l
                   radio-tubo ^^radio-tubo
                   ancho-l ^^ancho-l
                   solid? t
                   display? nil
                   )
          (hoyo-y :class 'cylinder-object
                  height ^^espesor-l
                  diameter ^^d-tornillo
```

```
                solid? t
                display? nil
                dist (- ^^largo-l ^diameter (/ ^diameter 2))
                dist2 (/ ^height 2)
                orientation(list
                        (rotate 90 :y-axis)
                        (translate (list ^dist2 ^dist 0.0))
                        )
                )
        (hoyo-x :class 'cylinder-object
                height ^^espesor-l
                diameter ^^d-tornillo
                solid? t
                display? nil
                dist (- ^^largo-l ^diameter (/ ^diameter 2))
                dist2 (/ ^height 2)
                orientation(list
                        (rotate 90 :x-axis)
                        (translate (list ^dist ^dist2 0.0))
                        )
                )
        )
)
```

En el código anterior se muestra la programación para obtener el diseño final de una de las partes de lo que será el esquinero, hay resaltar que para facilitar el modelado de las diversas piezas se realizaron bocetos y diagramas de estas para determinar la mejor forma de modelarlas, la estructura básica de esta clase es la siguiente: se define la clase *L-PERF*; esta clase será el resultado de una diferencia de otras clases antes definidas (ya sea que el programador las haya definido o que estén ya creadas dentro de AML); se especifican las propiedades de esta clase (*espesor-l, largo-l, radio-tubo, ancho-l, d-tornillo*), la propiedad object-list sirve para indicar los elementos que serán diferenciados (aquí es muy importante el orden en que se coloquen los elementos para obtener el resultado deseado); enseguida se especifican los subobjetos que serán utilizados para obtener la pieza terminada, en este caso se están utilizando cuatro subobjetos, de estos cuatro subobjetos el primero se está creando en la misma clase (*l-perf1*), de los otros tres uno de ellos (*base*) ha sido creado por el programador anteriormente y los otros dos ya existen en AML.

La forma en que se irán relacionando estos subobjetos es la siguiente: la clase *L-PERF* primero llama a los subobjetos que se han parametrizado con las propiedades de *L-PERF*,

cuando se cargan los subobjetos *base, hoyo-y y hoyo-x* estos cambian los valores por *default* de sus propiedades por los que se le están pasando para que se redimensionen y se reorienten según sea el caso. Lo que sucede con el subobjeto *l-perf1* cuando se carga es que debe esperar hasta que los subobjetos *base y hoyo-x* se hayan redefinido completamente para poder terminar de cargarse, esto debido a que es un subobjeto de tipo diferencia y que utiliza a estos subobjetos para poder definirse. Una vez que se han terminado de cargar todos los subobjetos la clase queda definida completamente después de operar la diferencia entre *l-perf1 y hoyo-y*. La explicación gráfica de lo expuesto anteriormente se muestra en la figura 4.6.

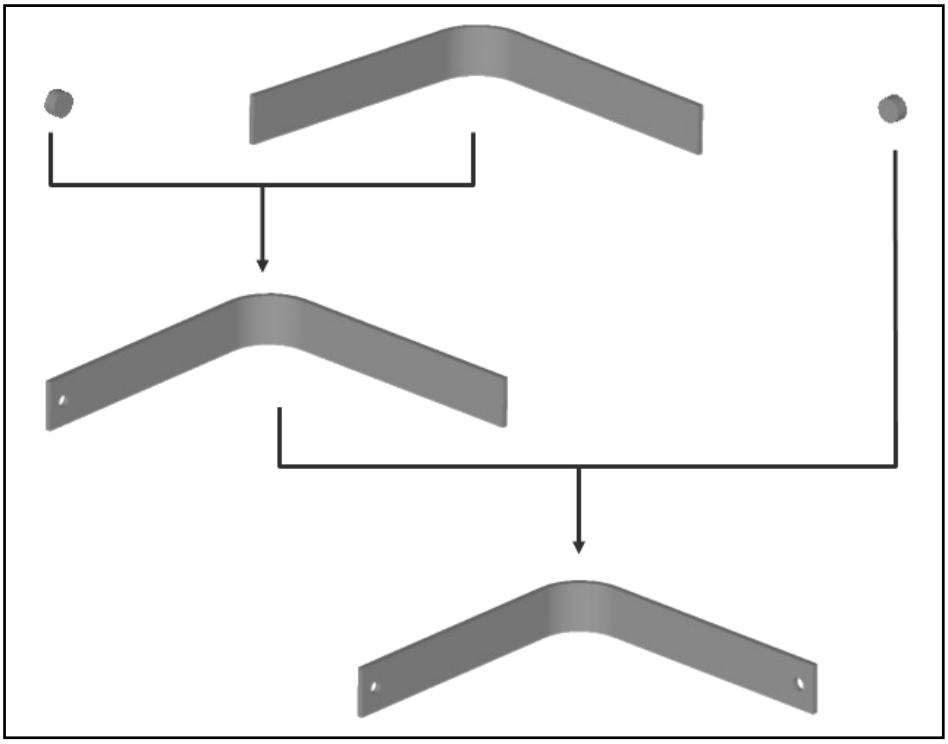

Figura 4.6 Descripción de la formación de una parte del esquinero

De forma análoga han sido programados todos los componentes del caso de estudio, la diferencia en el caso de los subensambles es que los subobjetos en este caso no se operan únicamente, se orientan las piezas que formaran el ensamble y se parametrizan respecto a las propiedades de la clase que identificará al ensamble, es de esta forma como se crea el producto ensamblado que se muestra en el Modelo del Producto.

Es por esta razón que se ha elegido la implementación en las tres partes antes descritas, como se puede observar, para realizar el diseño visual son necesarios valores numéricos para las diferentes piezas que forman el producto, sin embargo estos valores pueden ser aproximados, no es necesario que se tengan los valores finales del modelo para empezar a diseñarlo ya que estos valores se podrán consultar y asignar posteriormente de las diversas tablas que formen la segunda parte de la implementación.

Esta forma de implementar el modelo tiene la ventaja de poder reutilizar el código de las piezas e incluso de ensambles realizados con anterioridad para diseñar un modelo totalmente nuevo.

4.2.2 Implementación de la Segunda Parte (Implementación de Tablas)

En esta parte de la implementación se han creado las diferentes tablas que se encargarán de almacenar la información concerniente al Modelo del Producto. La estructura básica del manejo de tablas es la siguiente:

```
(define-class TEST-PIECES-TABLE-CLASS-1
  :inherit-from (flat-file-record-table-class)
  :properties (
               table-filename "Tabla1.txt"
               table-prefix "Tabla1"
               table-system-path (logical-path :tesis)
               file (concatenate 'string ^table-system-path ^table-filename)
               size 100
               record-format (list
                                 :integer    ;;; 0.-   index
                                 :string     ;;;
                                 '(:real 3)  ;;;
                                 '(:real 3)  ;;;
                                 '(:real 3)  ;;;
                                 '(:real 3)  ;;;
                                 '(:real 3)  ;;;
                                 '(:real 3)  ;;;
                                 '(:real 3)  ;;;
                                 '(:real 3)  ;;;
                                 '(:real 3)  ;;;
                                 '(:real 3)  ;;;
                                 '(:real 3)  ;;;
```

```
                    '(:real 3)  ;;;
                    '(:real 3)  ;;;
                    :symbol     ;;;
                )
            primary-key-field 0
            field-delimiter ':tab
        )
    )
```

Como se puede ver para crear una tabla se crea una clase que herede de la clase flat-file-record-table-class, esta clase contiene el formato general para la definición de cualquier tabla; como en cualquier otra clase se definen enseguida las propiedades: table-filename indica el nombre del archivo en donde se almacenarán los registros de la tabla; table-prefix aquí se coloca únicamente el nombre del archivo (sin extensión); table-system-path nos indica la ruta donde se encuentra el archivo donde se almacena la tabla, esta ruta puede ser absoluta o ser referida a una unidad definida en el logical-path uno de los archivos de configuración de instalación que usa AML para consultar las rutas de acceso, en el modelado de este sistema se opta por la segunda opción y se recomienda que se haga de esta forma pues se puede modificar la ruta de la tabla según nuestra conveniencia; *file* nos indica la ruta completa para poder localizar el archivo de la tabla, como se puede ver, esto se consigue concatenado las propiedades table-system-path y table-filename; enseguida se define el número máximo de registros que puede tener una tabla; *record-format* en esta propiedad se indican en forma de lista los campos que tendrá el registro, que en este caso son dieciséis campos, esto se hace indicando uno de los cuatro tipos de variable que puede almacenar un campo; primary-key-field, en esta propiedad se indica cual de los campo será la llave primaria, hay que tener en cuenta que los campos se empiezan a enumerar desde cero, y que el campo cero es por default la llave primaria; field-delimiter en esta se especifica el tipo de delimitador que se usara para diferenciar los campos, este delimitador puede ser uno de los tres símbolos siguientes: tabulación (‘:tab), espacio (‘:space) y coma (‘:comma). Con esto queda completamente definida la estructura que tendrá la tabla.

Después de definir la estructura de la tabla se procede a definir dos métodos y una función para que actúen sobre esta tabla, los cuales se muestran a continuación.

El primero de estos métodos es:

```
(define-method     GET-MAX-INDEX      TEST-PIECES-TABLE-CLASS-1
()
 (let*
    (
     (index-list   (select-table (the) :fields '(0)))
     )
   (if index-list
         (apply 'max (loop for i in index-list append i))
    -1)
  )
 )
```

En el primer renglón se escribe el nombre del método seguido de la clase a la cual se va a aplicar el método y se colocan entre paréntesis los argumentos si es que se requieren; en el segundo renglón se encuentra let* que indica que se creará la variable local index-list donde se almacena el valor del campo 0 de la tabla; en el sexto renglón se condiciona a que exista al menos un registro en la tabla para poder aplicar la función máximo a la tabla en el campo 0, esto regresa el número de registros existentes en la tabla. Este método sirve cuando se agrega un nuevo registro en la interfaz de usuario, debido a que dicho registro deberá ser colocado al final del último registro.

El segundo de los métodos es:

```
(define-method      get-table-record        TEST-PIECES-TABLE-CLASS-1 (index)
 (select-table (the TEST-PIECES-TABLE-CLASS-1)
           :record-test 'compare-table-piece-index
           :fields '(0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15)
           :test-param (list index)
           )
 )
```

Este método obtiene un registro de la tabla indicada en la definición de este; select-table selecciona la tabla a utilizar y se asignan los valores a las propiedades que se mencionan a continuación: :record-test indica el nombre de la función de búsqueda de un registro, esta función debe estar previamente definida por el usuario o existir en AML cargando

automáticamente el primer argumento para esta función; :fields indica el número de campos que serán seleccionados de la tabla, en caso de que se colocará '(1 4) solo dos campos en este caso 1 y 4 serían seleccionados; :test-param indica el segundo argumento de la función a utilizar en :record-test que en este caso seria el argumento que recibe el método (index).

Finalmente la función es:

```
(defun compare-table-piece-index (record parameters)
  (equal (nth 0 record) (nth 0 parameters))
  )
```

Esta función se encarga de comparar los campos de la interfaz con los campos del archivo que contiene los datos, utiliza dos argumentos: el primero corresponde a la propiedad :record-test del método anteriormente descrito, el segundo corresponde a la propiedad :test-param; equal se encarga de revisar que el contenido del registro 0 de la tabla sea igual al contenido del registro 0 de la interfaz, en caso de que sea diferente indicará que dicha tabla no corresponde con la solicitada en la interfaz o que la tabla no existe.

4.2.3 Implementación de la Tercera Parte (Unificación del Sistema)

En esta sección se explicará la forma en que se ha realizado la unión entre los dos partes del Modelo del Producto explicadas anteriormente. A continuación se presentarán y explicarán fragmentos del código usado para este fin.

```
(define-class INTERFAZ-CARAC-GEOM
  :inherit-from (DATA-MODEL-NODE-MIXIN geom-copy-object)
  :properties(
              properties-available? t
              catalog-object nil
              main-form (default nil)
              pieces-table-source-object nil
              pieces-records-list-1 (default nil)
              piece-record nil

              (select-piece-action :class 'access-property-class
```

```
                                    label "Selecciona Caracteristicas Geometricas "
                                    access-object (the superior superior)
                                    access-method 'SELECT-PIECES-METHOD
                                    )
            (index :class 'editable-data-property-class
                    label "ID"
                    available? ^^properties-available?
                    formula (if ^piece-record (nth 0 ^piece-record) (1+ (GET-MAX-INDEX ^pieces-table-
source-object)))
                    )
            (descripcion :class 'editable-data-property-class
                        label "Descripción"
                        available? ^^properties-available?
                        formula (if ^piece-record (nth 1 ^piece-record) "Versión del Producto")
             (largo-ch :class 'editable-data-property-class
                        label "Largo de la Charola"
                        formula (if ^piece-record (nth 4 ^piece-record) 1500)
                        )
            (ancho-ch :class 'editable-data-property-class
                        label "Ancho de la Charola"
                        formula (if ^piece-record (nth 5 ^piece-record) 700)
                        )
                        ...
                        ...
                        ...
(validar :class 'access-property-class
                        label "Validación de campos"
                        access-object (the superior superior)
                        access-method 'VALIDA-METHOD
                        )
             (add-piece-action :class 'access-property-class
                label "AGREGAR"
                access-object (the superior superior)
                access-method 'ADD-PIECES-METHOD
                )
             (update-piece-action :class 'access-property-class
                label "ACTUALIZAR"
                access-object (the superior superior)
                access-method 'UPDATE-PIECES-METHOD
                )
             (delete-piece-action :class 'access-property-class
                label "BORRAR"
                access-object (the superior superior)
                access-method 'DELETE-PIECES-METHOD
                )
            property-objects-list (list
                                    (the superior select-piece-action self)
                                    (the superior index self)
                                    (the superior descripcion self)
                                    (the superior largo-ch self)
                                    (the superior ancho-ch self)
                                        ...
                                        ...
                                        ...
                                    (the superior validar self)
                                    (the superior add-piece-action self)
```

```
                                    (the superior update-piece-action self)
                                    (the superior delete-piece-action self)
                                    )
            largo (default (/ ^largo-ch ^escala))
            ancho (default (/ ^ancho-ch ^escala))

            largo-min-mueble !ancho-ch

 :subobjects(
            (charolas :class 'CHAROLAS
                        alto-mueble ^^alto-mueble
                        num-char ^^num-char
                        largo ^^largo
                        ancho ^^ancho
                        alto ^^alto
                        bisel ^^bisel
                        calibre ^^calibre
                        d-tubo ^^d-tubo
                        d-tornillo ^^d-tornillo
                        espesor ^^espesor
                        margen ^^espesor
                        quantity (if (< ^^num-char ^^char-max)  ^^num-char ^^char-max)
                        spacing  (if (< ^^espaciado ^^h-min)
                                    ^^h-min
                                   (if (< ^^h-max ^^espaciado)
                                     ^^h-max
                                     ^^espaciado)
                                   )
                        )
            (tubo1 :class 'tubocompleto
                    escala ^^escala
                    rad-tubo (/ ^^d-tubo 2)
                    espesor-tubo ^^calibre-tub
                    altura-mueble ^^alto-mueble

                    tipo-pata ^^tipo-pata
                    referencia (list ^^distx ^^disty 0.0)
                    )
            (tubo2 :class 'tubocompleto
                    ...
             )
            (tubo3 :class 'tubocompleto
                  ...
                    )
            (tubo4 :class 'tubocompleto
                    ...
                    )
            )
  )
```

El código mostrado tiene dos fines, el primero generar la interfaz gráfica de usuario para este tipo de características y el segundo unificar la representación geométrica del producto completo

Lo primero se consigue generando un listado de propiedades que heredan de clases especiales para la representación grafica de la interfaz de usuario, tales como editable-data-property-class, mediante las cuales el usuario introduce, edita ó consulta todos los valores de las características geométricas del modelo, todas estas propiedades son operadas por medio de un factor de escala para tener una representación visual adecuada de mueble; además se han creado cinco botones; el primer botón ejecuta un método que permite seleccionar la versión del producto con la que se desea trabajar; el segundo botón creado se encarga de correr un método que sirve para verificar que los valores que el usuario introduce estén dentro de los rangos permitidos; y los últimos tres botones ejecutan métodos para la manipulación de los datos en la tabla de características geométricas (agregar, actualizar y borrar).

El segundo propósito se consigue con ayuda de los subobjetos. Para crear al producto ensamblado se ha creado un arreglo con el número de charolas que el usuario requiere; esto se hace con un ensamble previo de la charola y con ayuda de la operación de arreglo lineal con la que cuenta AML; y con el posicionamiento de los subensambles de los cuatro tubos completos previamente realizados. Cabe aclarar que los valores que sirven para definir a los subensambles utilizados han sido referenciados a los valores de las propiedades que se han operado con el factor de escala.

A continuación se muestra y explica parte de los códigos para los métodos usados en esta clase.

```
(define-method SELECT-PIECES-METHOD INTERFAZ-CARAC-GEOM ()
  (let*
     ((record-list             !pieces-records-list-1)
      (options-list            (append (list (list "Nuevo registro" 'new))
                                       (loop for record in record-list
                                         for i from 0 to (1- (length record-list))
                                         collect (list (format nil "~a ~a" (nth 0 record) (nth 1
record)) i))))
      (selected-option         (pop-up-menu options-list))
      )
    (when selected-option
      (if (equal selected-option 'new)
```

```
        (change-value !piece-record nil)
        (change-value !piece-record (nth selected-option record-list))
        )
    (update-work-area !main-form)
    )
  )
 )
```

El método anterior se encarga de recuperar los datos de una determina versión del producto o bien crear una nueva versión, esto lo consigue de la siguiente forma. Se le pasa una lista de los campos existentes de la tabla que es recibida en la variable *record-list*; se asigna a la variable option-list el resultado obtenido al recorrer todos los registros de la tabla presentando únicamente los campos *0* y *1* que corresponden a el *ID* y a la *Descripción*; se le presenta al usuario un menú desplegable con el contenido de *option-list* con la función *pop-up-menu* de AML, quedando en espera de que el usuario seleccione una de las opciones listadas cuyo resultado se almacena en la variable *selected-option*. Si *selcted-option* existe se compara con el símbolo *'new*, en caso de que se a igual a *'new* se cargan los valores por default de las propiedades del modelo, de lo contrario se cambian estos valores por los del registro seleccionado de la tabla, y se actualiza la presentación de la interfaz.

```
(define-method ADD-PIECES-METHOD INTERFAZ-CARAC-GEOM ()
 (let*
    ((record          (get-table-record !pieces-table-source-object !index))
     (new-record        (when (null record)
                           (list !index !descripcion !escala !num-char !largo-ch
                                 !ancho-ch !alto-ch !bisel-ch !calibre-ch !d-tornillo-ch
                                 !espesor-ch
                                 !d-tubo-m !calibre-tub-m !alto-mueble-m !espaciado-ch
                                 !tipo-pata)))
    )

   (if record
        (message-box "Error" (format nil "EL REGISTRO CON EL ID ~a YA EXISTE." !index))
    (progn
        (add-record   !pieces-table-source-object    new-record)
        (save-table    !pieces-table-source-object)
        (message-box "Mensaje" "¡REGISTRO AGREGADO SATISFACTORIAMENTE!")
        (smash-value (the   pieces-records-list-1 (:from   !catalog-object)))
        )
    )
  )
 )
```

El código mostrado anteriormente permite almacenar un nuevo registro en la tabla de características geométricas, el proceso que está realizando este método se describe a continuación. Primero se busca en la tabla de características geométricas (*pieces-table-source-object*) el número de registro que se desea almacenar (*index*) y el resultado de esta búsqueda se almacena en la variable *record*; si el valor de *record* es *null,* es decir no encontró el registro en la tabla se crea una lista con los valores ordenados de los campos que se desean almacenar; si el valor de *record* existe entonces se envía un mensaje de error, de otro modo se añade un nuevo registro a la tabla de características geométricas (*pieces-table-source-object*) con los valores de los campos almacenados en la lista *new-record*, se salva la tabla, se muestra un mensaje de éxito al agregar el registro y se actualiza la vista de la forma de características geométricas del programa.

```
(define-method VALIDA-METHOD INTERFAZ-CARAC-GEOM ()
  (let*
     ((sep-char-max (abs (/ (- !alto-mueble-m (* !alto-ch !num-char)) (- !num-char 1))))
      (sep-char-min (* !alto-ch 2))
      (largo-min-mueble !ancho-ch)
      (d-tubo-max (/ !ancho-ch 6))
      (d-tubo-min (/ !d-tubo-m 10))
      (num-char-max (- (truncate !alto-mueble-m (* 3 !h-min)) 1) )
      (num-char-min (default 1))
      (espesor-esq-min (* !calibre-ch 3))
      (espesor-esq-max (* !calibre-ch 8))
      (alto-m-max (* !largo-ch 1.5))
      (alto-m-min (/ !ancho-ch 2))
      )

     (if  (OR (> !largo 1000) (> !ancho 1000) (> !alto-mueble 1000) )
            (message-box "Escala" (format nil "Incrementar el factor de escala para la correcta
visualización"))
          )
     (if (< !espaciado-ch sep-char-min)
            (message-box "Separación entre Charolas" (format nil "La separación es muy pequeña, el
mínimo pertimitido es: ~a" sep-char-min))
          (if (> !espaciado-ch sep-char-max)
             (message-box "Separación entre Charolas" (format nil "La separación es muy grande, el
máximo pertimitido es: ~a" sep-char-max))
           )
          )
     (if (< !largo-ch largo-min-mueble)
            (message-box "" (format nil "El largo del mueble es muy pequeño, el mínimo pertimitido es: ~a"
largo-min-mueble))
          )
     )
  )
```

El código del método presentado anteriormente sirve para verificar que los valores que introduce el usuario de las características geométricas estén dentro de un rango válido; estos rangos se calculan en base a otros parámetros que da el usuario como son: alto del mueble, alto de la charola, largo, ancho, entre otros; si se introduce algún valor que esté fuera del rango se muestra el máximo o el mínimo, según sea el caso, de la variable que esta fuera de rango; esto se consigue comparando los valores máximos y mínimos calculados con los valores que introduce el usuario mediante rutinas if anidadas como se puede observar en el código anterior.

Las demás tablas que forman la base de datos así como las representaciones de éstas en el entorno de usuario se han hecho de forma análoga a la explicada anteriormente, donde cada una de estas tablas es creada en un archivo distinto y la operación de validación solo ha sido desarrollada para las características geométricas.

La última clase del código, que se muestra a continuación, reúne todas las clases que sirven para crear y manipular a las tablas definiendo de esta forma al producto completo. Es esta clase la que gobernará todo el sistema por lo que es desde aquí donde se definen las tablas en las que se trabajará y los campos válidos del registro para cada una de las tablas. El objetivo del último subobjeto es mostrar dos botones que sirven para poder ver el contenido tanto de la tabla que contiene las características geométricas de todas las versiones del producto como una tabla de posibles materiales del mueble.

```
(define-class MUEBLE-PRINCIPAL
  :inherit-from (object)
  :properties(
              main-form (the interface forms data-model-main-form)
              (pieces-table-1 :class 'TABLA-CARAC-GEOM)
               ...
               ...
               ...
              pieces-records-list-1 (select-table ^pieces-table-1 :fields '(0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
15))
             ...
             ...
             ...
              )
  :subobjects(
              (caracteristicas-geometricas :class 'INTERFAZ-CARAC-GEOM
```

```
        pieces-table-source-object ^^pieces-table-1
        catalog-object (the superior superior)
        )
    ...
    ...
    ...
    (Reportes :class 'MOSTRAR-TABLAS
        pieces-table-source-object ^^pieces-table-1
        ...
        ...
        ...
        )
    )
)
```

4.3 Posibilidades de crecimiento

Hasta este momento el desarrollo que se ha hecho sobre modelos de información en AML abarca solamente parte del Modelo del Producto. En esta tesis sólo se incluyeron las clases que se consideraron fundamentales para hacer un primer acercamiento entre los modelos de información y el lenguaje de programación AML, esto deja un amplio campo de trabajo para futuros proyectos, entre estos están la implementación completa del Modelo del Producto, el Modelo de Manufactura, el desarrollo de un manejador de bases de datos utilizando exclusivamente AML y diversos sistemas de análisis numérico.

Cabe hacer mención que lo expuesto en esta tesis solo abarca las partes que son útiles para el desarrollo del sistema, y que aún quedan diversas áreas existentes en AML que no se han explorado todavía, por lo que este estudio se podría considerar como una introducción a la investigación del potencial que puede tener AML.

5 Descripción del Sistema Desarrollado en AML para el Modelo del Producto

5.1 Inicialización y configuración de AML.

AML proporciona una interfaz de usuario la cual permite cargar y compilar el código de manera que se pueda observar lo que se esta modelando. Sin embargo esta interfaz no sirve para modelar directamente como ocurren otros sistemas de modelado por ello es necesario indicar todas las instrucciones de modelado desde el código fuente.

El sistema del caso de estudio consta, como ya se mencionó, de un mueble de charolas. En esta sección se explicará a grandes rasgos la forma de utilizar el sistema desarrollado, como modificar los componentes principales del mueble, tales como el número de charolas, el diámetro del tubo, las bases del mueble, además de cómo generar nuevas versiones de este producto y como almacenar los cambios en las tablas correspondientes.

Al iniciar el ambiente de trabajo de AML aparece la siguiente pantalla:

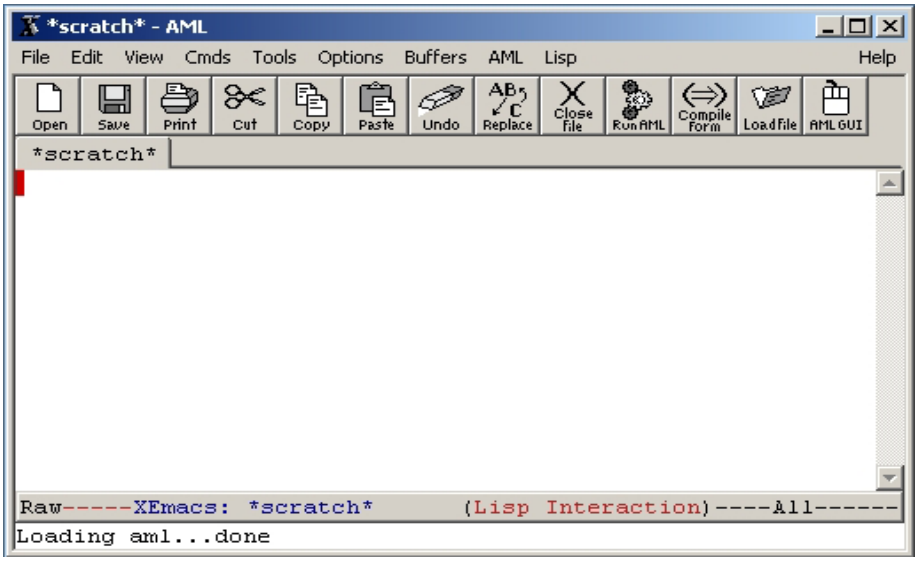

Figura 5.1 Arranque de AML

Enseguida se ejecuta AML ya sea pulsando "*(run aml)*", sin comillas, en la línea de comandos o bien haciendo click en el icono mostrado en la Figura 5.2

Figura 5.2 Ejecución de AML

Al hacer esto se cargan automáticamente todas las actualizaciones y parches del AML, al terminar el sistema está listo para trabajar con él.

En el caso de este sistema es necesario cargar la interfaz de modelado de AML esto debido a que en código fuente se tienen algunas instrucciones para añadir luces y orientación del CANVAS de AML. Esto se hace de la siguiente forma

Al hacer click en el icono "*AML GUI*", Figura 5.3, se carga la interfaz grafica de AML, mostrando la pantalla de bienvenida ilustrada en la Figura 5.4

Figura 5.3 Cargando interfaz gráfica De AML

Figura 5.4 Pantalla de bienvenida de AML GUI

Después de hacer click en el botón OK nos muestra dos barras de herramientas que indicando que se está en el GUI de AML (Figura 5.5). Lo que sigue ahora es abrir la "*AML Main Modeling Form*", esto se logra de la forma mostrada en la Figura 5.6.

Figura 5.5 Barras de herramientas de AML GUI

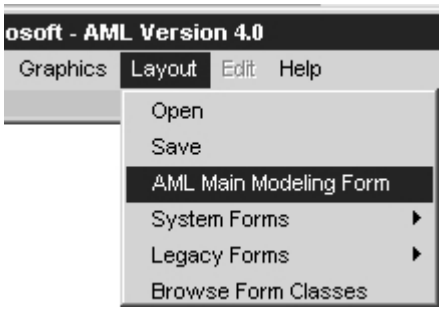

Figura 5.6 Abriendo el "AML Main Modeling Form"

La Figura 5.7 muestra el ambiente gráfico en el que se carga la aplicación desarrollada, esto trae la ventaja de no tener que desarrollar toda la interfaz de usuario sino que únicamente se colocan los elementos conforme se vayan necesitando.

Hay que resaltar que, al abrir este ambiente de trabajo la forma en que están orientados los ejes coordenados es como se muestra en la Figura 5.8, además de carecer de luces dentro del CANVAS, lo que dificulta una adecuada apreciación de la representación gráfica del producto.

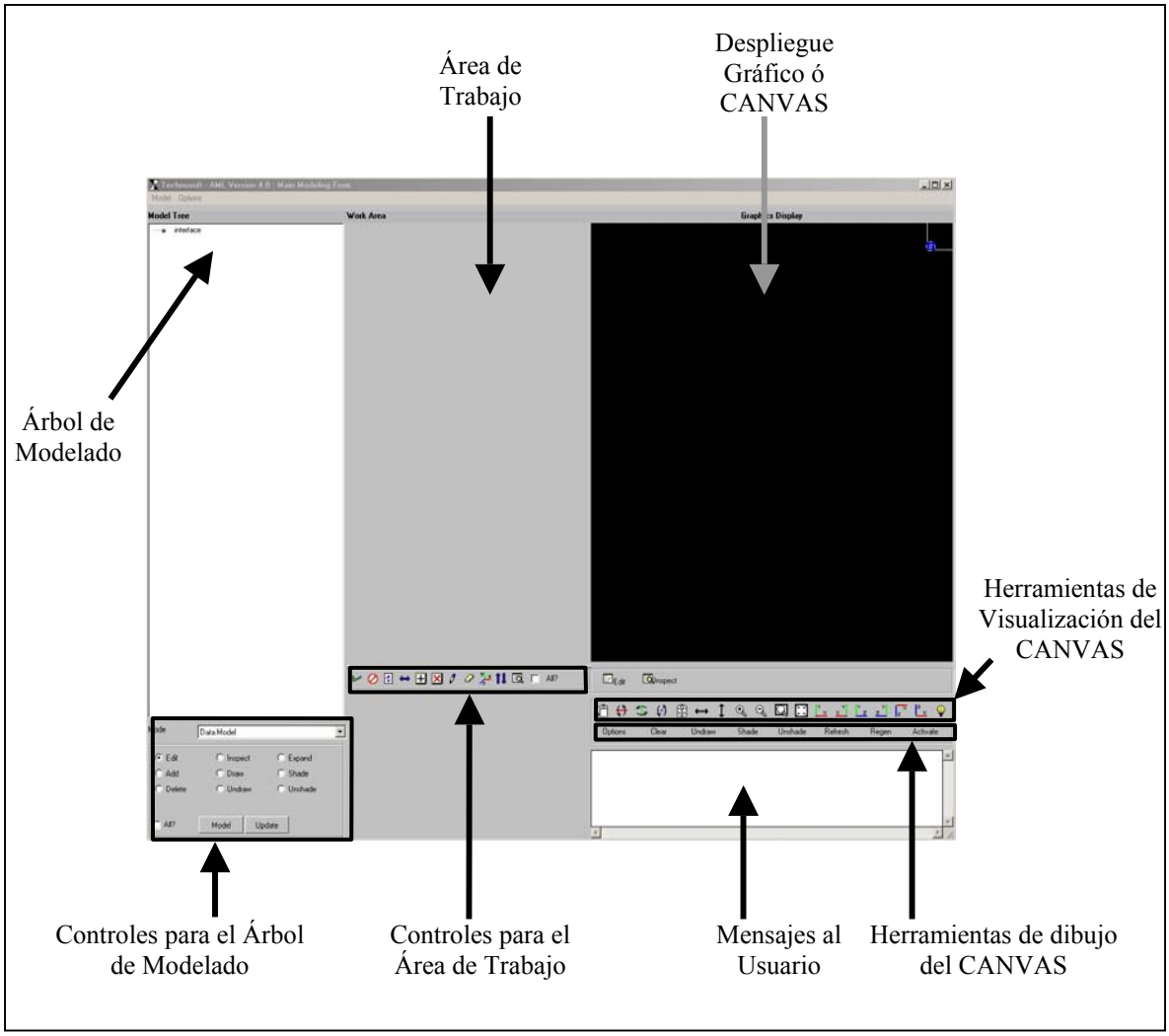

Figura 5.7 AML Main Modeling Form

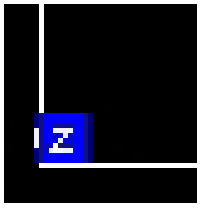

Figura 5.8 Orientación del CANVAS al inicio

5.2 Carga del sistema desarrollado para el Modelo del Producto

Para cargar el sistema desarrollado se hace clic en el icono "*Load File*" de la barra de herramientas, como se muestra en la Figura 5.9, y se especifica la ruta donde se encuentra

Figura 5.9 Cargando el sistema

el código fuente del sistema que se identifica por la extensión " *.aml* ", como se ilustra en la Figura 5.10.

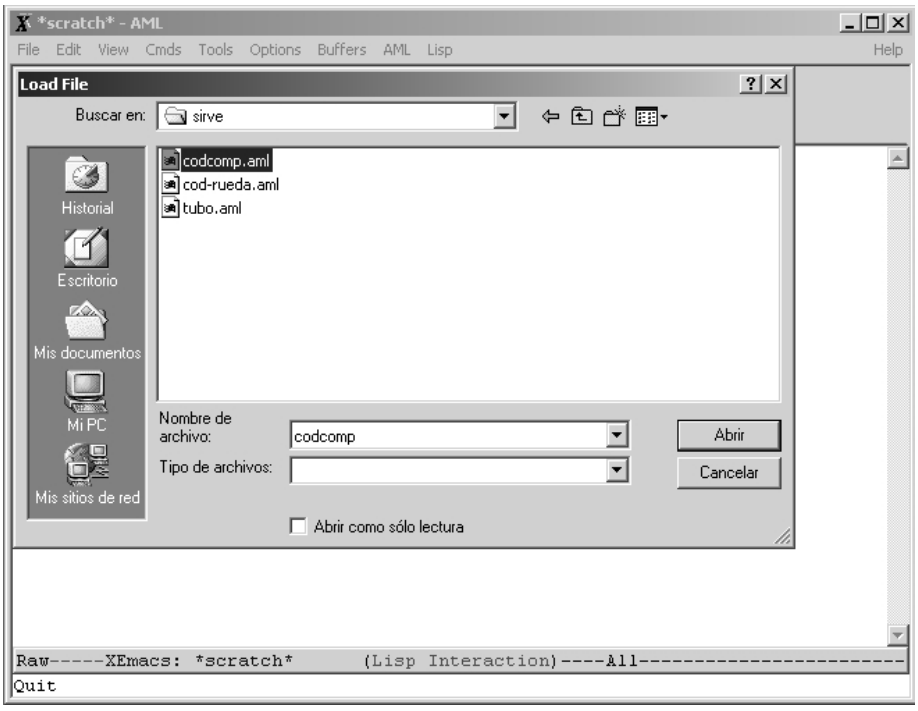

Figura 5.10 Cargando código del sistema

En el caso de que el sistema esté separado en varios archivos se repite esta operación hasta que se cargan completamente todas sus partes. Si el sistema no presenta errores al cargarse, se muestra una pantalla como la siguiente:

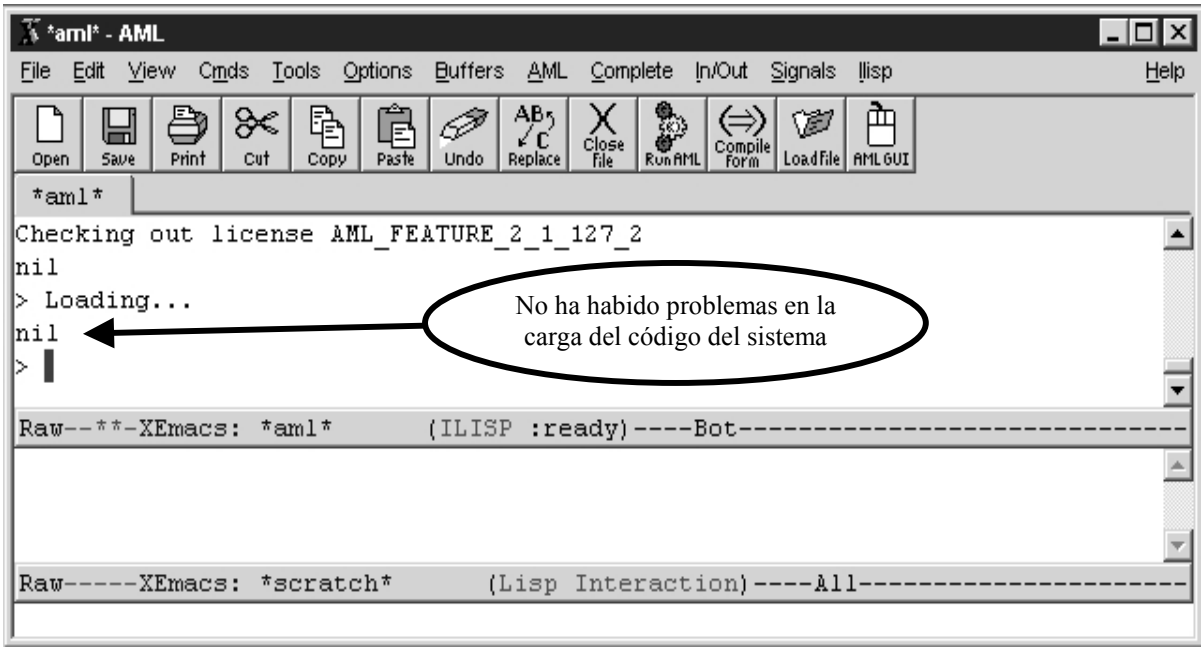

Figura 5.11 Abriendo el "AML Main Modeling Form"

Al cargar el sistema automáticamente en el CANVAS se ha reorientado el sistema de ejes coordenados (Figura 5.12) y se han agregado luces y ajustado la vista de cámara para una correcta visualización del producto.

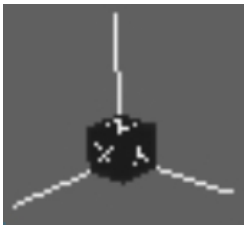

Figura 5.12 Sistema coordenado del CANVAS reorientado

Para poder agregar el modelo del producto en la interfaz de usuario y poder trabajar con él se requiere que los controles del árbol de modelado y los controles del área de trabajo estén configurados de la forma mostrada en la Figura 5.13.

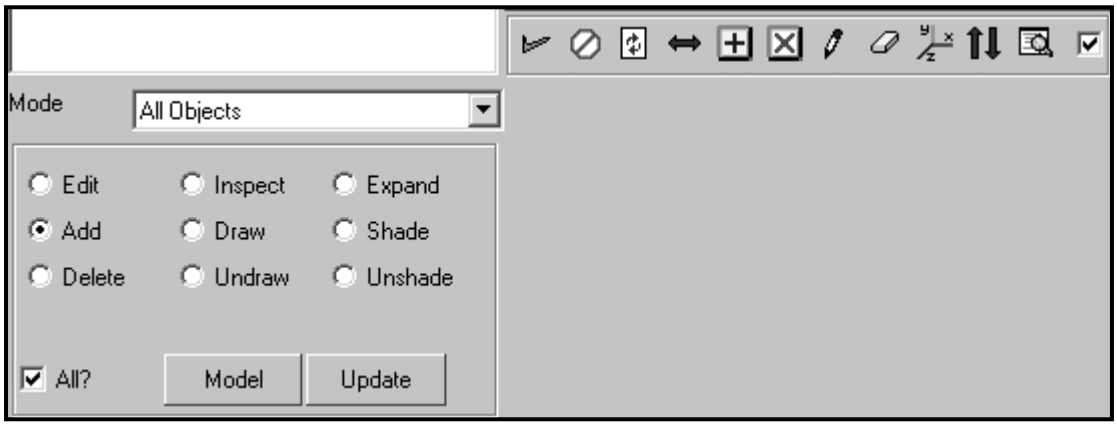

Figura 5.13 Configuración de los controles del árbol de modelado y del área de trabajo

Hasta este momento se ha configurado AML para poder cargar el sistema desarrollado, esto último se puede inicializar de dos formas. La primera es activar "*Add*" en el cuadro de controles del árbol de modelado y hacer click "*Interface*" dentro del árbol (Figura 5.14a), la otra es haciendo click con el botón derecho en "*Interface*" para que aparezca un menú emergente donde se selecciona "*Add*"(Figura 5.14b); no importa cual de las dos opciones se elija, en ambos se obtiene la pantalla mostrada en la Figura 5.15 donde se solicita el nombre de la clase a ser agregada.

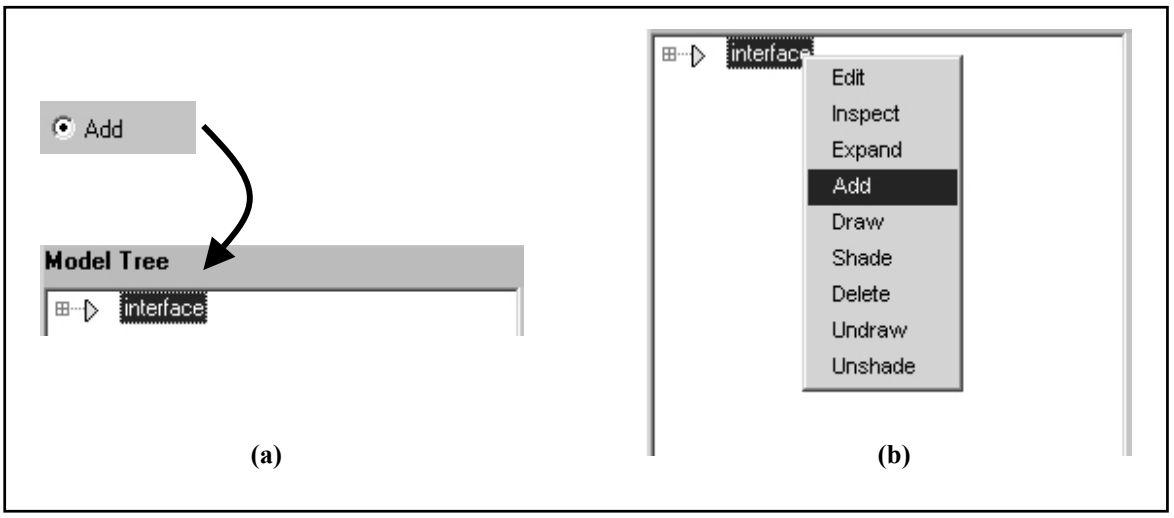

Figura 5.14 Opciones para añadir el modelo

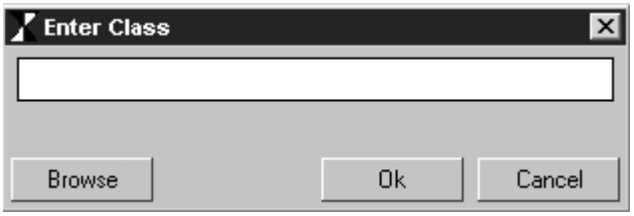

Figura 5.15 Solicitando nombre de la clase a agregar

Si se conoce el nombre exacto de la clase que se desea agregar se puede teclear en el cuadro de texto, pero si no se está completamente seguro se hace click en el botón "*Browse"*. Al hacer esto se muestra una lista de todas clases que tiene AML cargadas, muestra tanto las que carga automáticamente el sistema cuando se inicia, como las que el usuario ha cargado. Para cargar el sistema desarrollado se tiene que buscar en esta lista la clase "*mueble-principal*" (Figura 5.16); en el caso de que no aparezca y se esta seguro de haberla cargado de la forma mostrada al inicio de esta sección únicamente hay que hacer click en el botón "*Update"* para actualizar la lista y buscar nuevamente la clase; una vez seleccionada la clase se hace click en el botón "*Ok*" de la ventana "*Select Class*" y de la ventana "*Enter Class*".

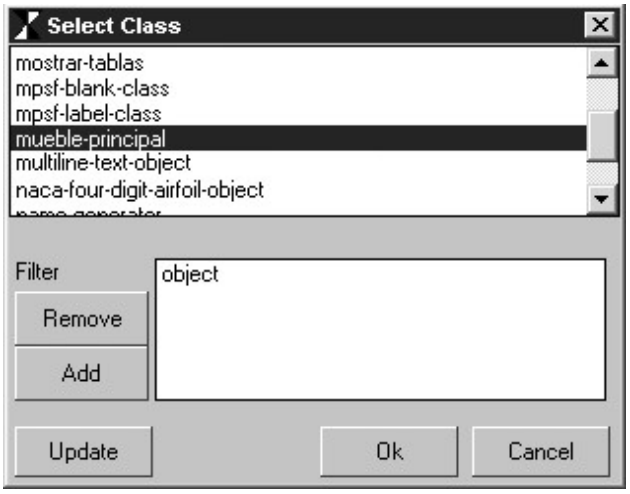

Figura 5.16 Buscando la clase a agregar

Lo que solicita en seguida es nombre con el que identificará a la clase en el sistema, este nombre es el que aparece en el árbol de modelado (Figura 5.17). Al terminar de introducir el nombre y hacer click en el botón "*Ok*" se ha completado la carga del sistema, esto se puede corroborar porque en el árbol de modelado se ha añadido un nuevo elemento con el

nombre introducido en la ventana "*Prompt*" lo que indica que el sistema esta listo para trabajar con él.

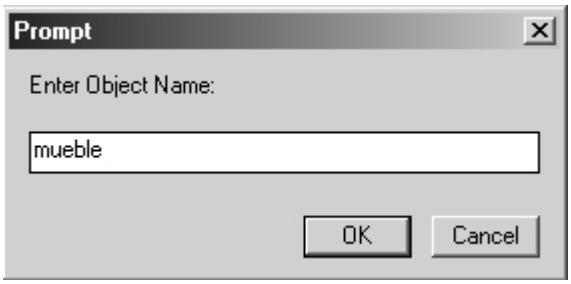

Figura 5.17 Introduciendo identificación del objeto

5.3 Trabajando con el sistema desarrollado

Para poder trabajar con el sistema hay expandir el objeto agrado para que muestre todos los subobjetos que forman al objeto principal. Al igual que cuando se agrega un objeto ya sea que se seleccione la opción "*Expand*"del cuadro de controles del árbol de modelado o del menú emergente al hacer click derecho en el objeto que se desea expandir se obtiene el mismo resultado (Figura 5.18).

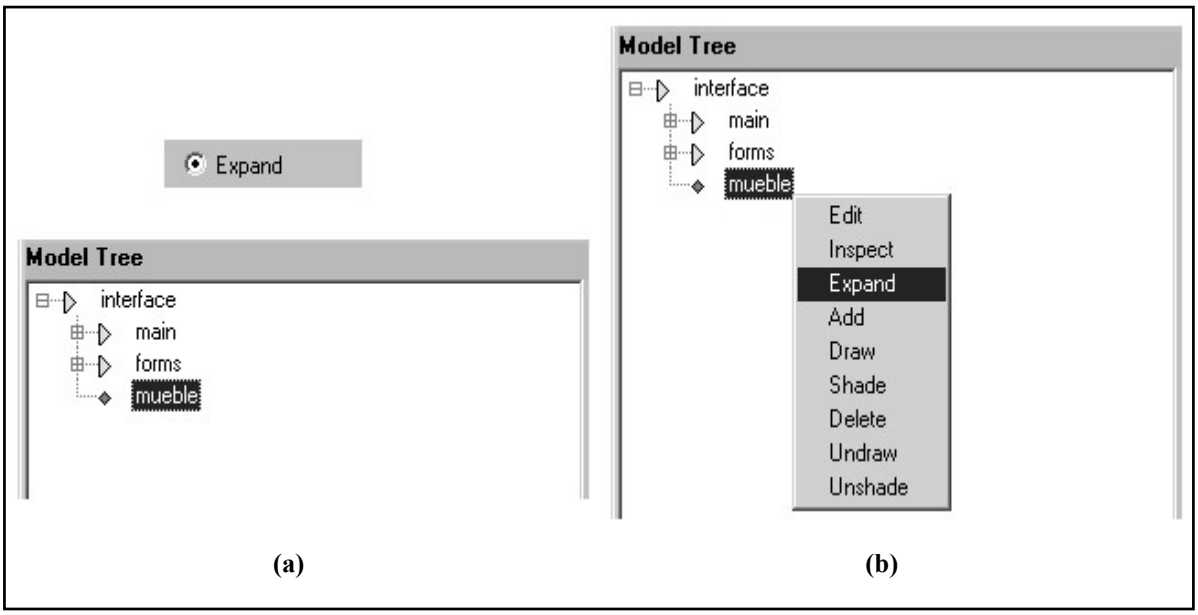

Figura 5.18 Expandiendo un objeto

Para ver el contenido modificable de estos subobjetos se tiene que hacer uso de la función *edit* de AML (Figura 5.19), esto muestra el área de trabajo una representación grafica en la que se pueden editar valores del subobjeto seleccionado (Figura 5.20)

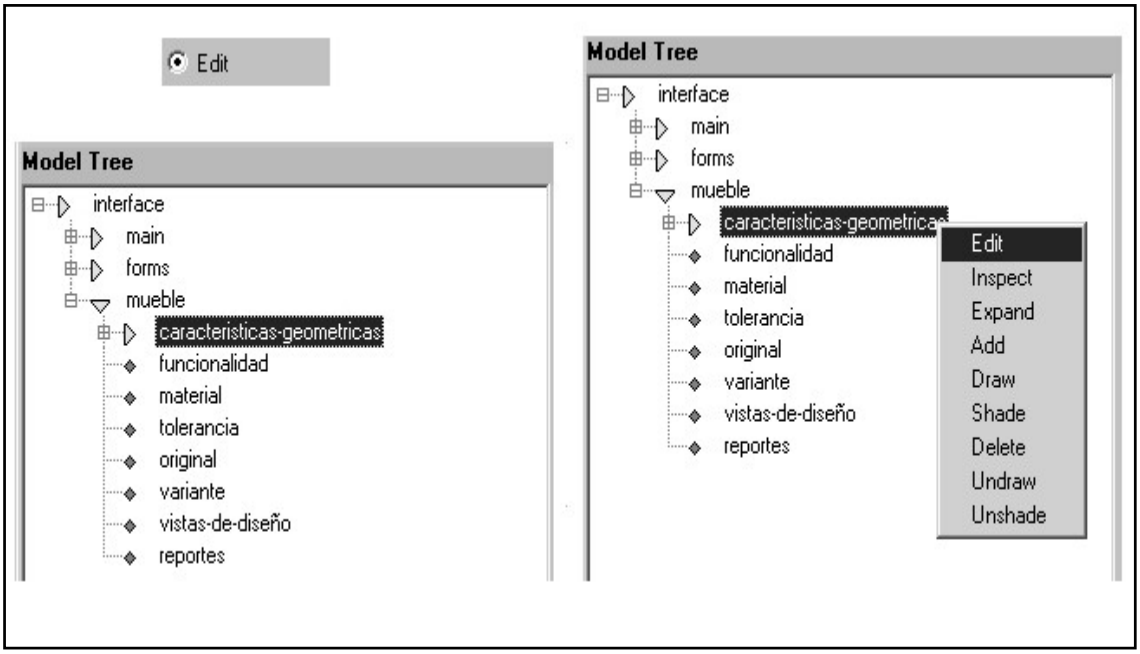

Figura 5.19 Editando un objeto

Al mostrar por primera vez los valores editables de cualquiera de los subobjetos, excepto del subobjeto "*reportes*", el sistema se comporta como si el usuario deseara crear una nueva versión del producto cargándose de esta forma todos los valores por omisión para este y colocando en el campo "*id*" el número que identificara a esta última versión del producto.

Al mostrar por primera vez los valores editables de cualquiera de los subobjetos, excepto del subobjeto "*reportes*", el sistema se comporta como si el usuario deseara crear una nueva versión del producto cargándose de esta forma todos los valores por omisión para este y colocando en el campo "*id*" el número que identificará a esta última versión del producto.

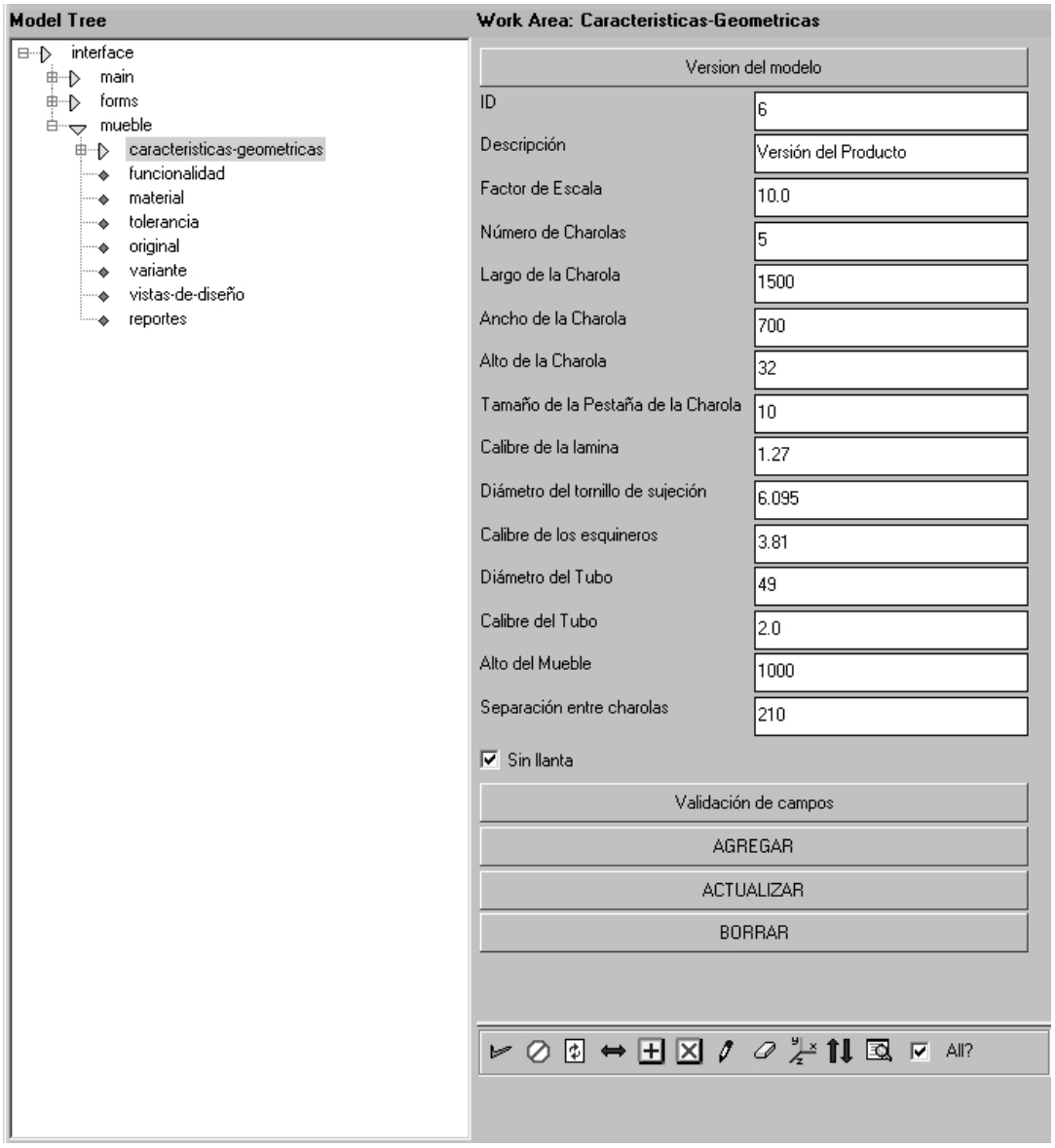

Figura 5.20 Propiedades editables del subobjeto "*caracteristicas-geometricas*"

Para poder trabajar en alguna versión almacenada del producto solo hay que hacer click en el botón "*versión del producto*", con el botón secundario del ratón se elige una de las versiones del producto mostradas y se actualizan los valores de las propiedades modificables del producto (Figura 5.21). Sin embargo estos valores se deben de propagar por todo el producto, para hacer esto solamente se tiene que hacer click en el icono de aceptación (Figura 5.22), la aceptación de los valores de las propiedades se tienen que

realizar cada vez que se haga alguna modificación en cualquier propiedad ya sea tanto en una nueva versión como en una ya existente.

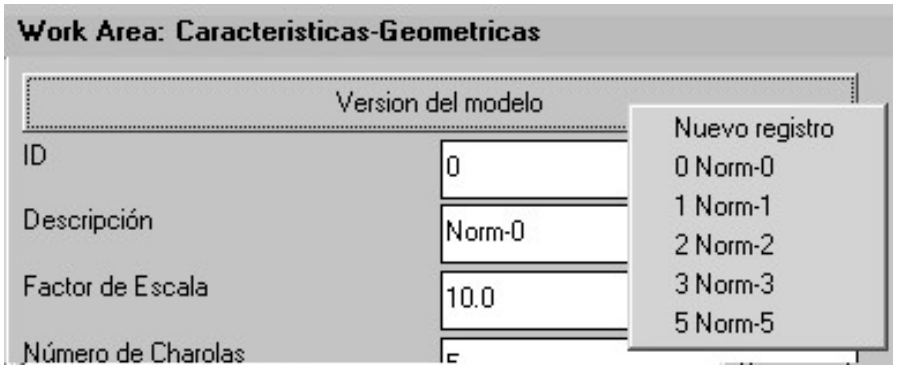

Figura 5.21 Seleccionando otra versión del producto

Figura 5.22 Aceptación de valores para el producto

Para poder visualizar la representación gráfica del producto únicamente se tiene que hacer click en el icono "*draw*" (Figura 5.23) esto hará que el objeto seleccionado en el árbol de modelado se dibuje en el *CANVAS* (Figura 5.24); hay que resaltar que para que un objeto sea dibujado debe contener propiedades gráficas o bien heredar de un objeto que tenga éstas propiedades, y que dependiendo de ellas se podrán dibujar, ya sea como sólidos en el *CANVAS* o como formas gráficas que ayuden a desarrollar una nueva interfaz de usuario.

Figura 5.23 Dibujando el objeto seleccionado

En el caso de que la visualización del producto no sea muy favorable, el área de graficación cuenta con las herramientas típicas de visualización (Figura 5.25), al hacer click en los íconos estos quedan temporalmente bloqueados para que al hacer nuevamente click y sin soltar el botón derecho del ratón se pueda hacer uso de la herramienta, al soltar el botón o hacer click con el botón derecho del ratón el comando se termina o cancela.

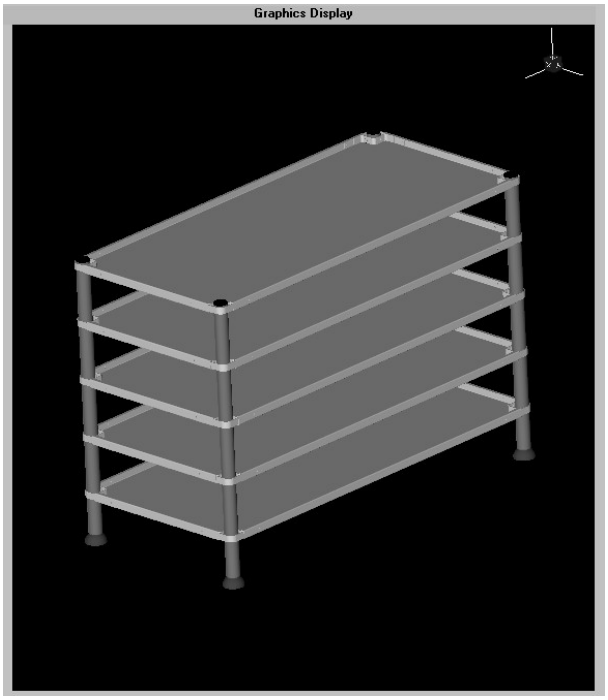

Figura 5.24 Objeto dibujado en el CANVAS

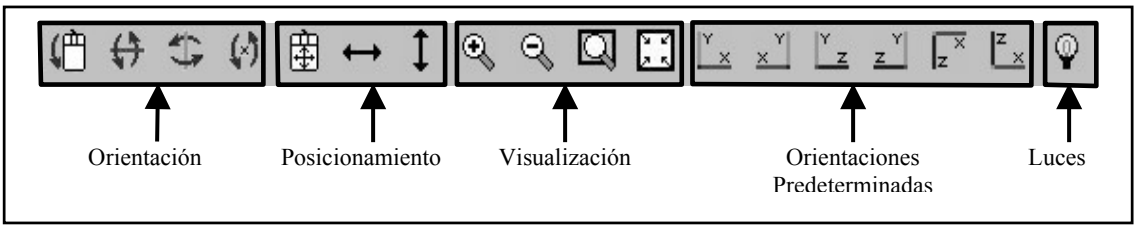

Figura 5.25 Herramientas de visualización del CANVAS

Además de las anteriores herramientas el área del CANVAS se cuenta con herramientas adicionales que nos ayudan a modificar ciertos aspectos de visualización del producto dibujado (Figura 5.26). Entre estas se encuentran para cambiar el fondo del área de dibujo, ajustar la cámara de visualización, remover luces, entre otras, en la Figura 5.27 se ilustra la forma en que se cambia el fondo del área de dibujo, esto con el fin de apreciar mejor los detalles de producto.

Figura 5.26 Herramientas avanzadas del CANVAS

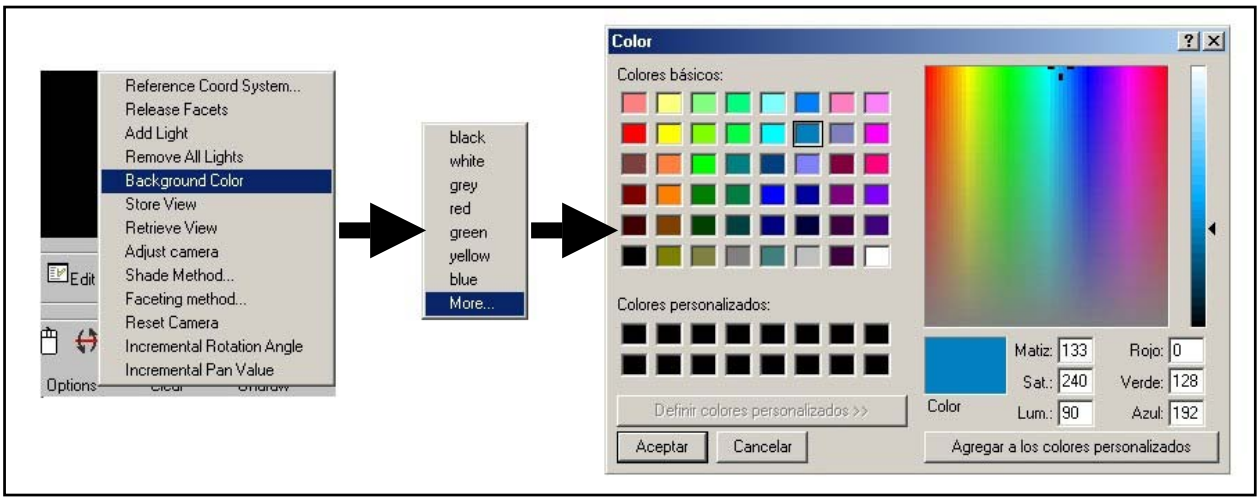

Figura 5.27 Cambiando el fondo del área de Dibujo

Al crear una nueva versión o modificar una versión existente del producto, no se puede estar seguro de que los valores que el usuario proporciona sean válidos para esa versión del producto; es por esto que se ha creado un método el cual se encarga de obtener los valores máximos y mínimos de las propiedades geométricas del producto, en caso de que los valores no se encuentren en estos rangos, se mostrarán mensajes de error que indiquen la propiedad que no cumple con el rango. Para poder accesar a este método únicamente hay que hacer click en el botón "*Validación de campos*" que se encuentra enseguida del listado de propiedades geométricas. En la Figura 5.28(a) se muestra un listado de valores, algunos de ellos erróneos para el producto, al validar los campos se despliegan los mensajes mostrados en las Figuras 5.28 (b) y 5.28 (c).

En la Figura 5.28(a) también se pueden apreciar los tres botones que sirven para poder accesar y manipular la tabla que contiene la información geométrica del producto, las acciones que desempeñan son: agregar una nueva versión del modelo, actualizar una versión del modelo y borrar una versión del modelo.

La manipulación de los subobjetos restantes que forman al producto se hace de forma parecida a la de características con la excepción de que no cuentan con una representación en el CANVAS. En el caso del subobjeto reportes solo se presentan un par de botones

(Figura 5.29), estos botones lo único que hacen es una representación de la tabla de características geométricas y de la tabla de materiales (Figuras 5.30 y 5.31).

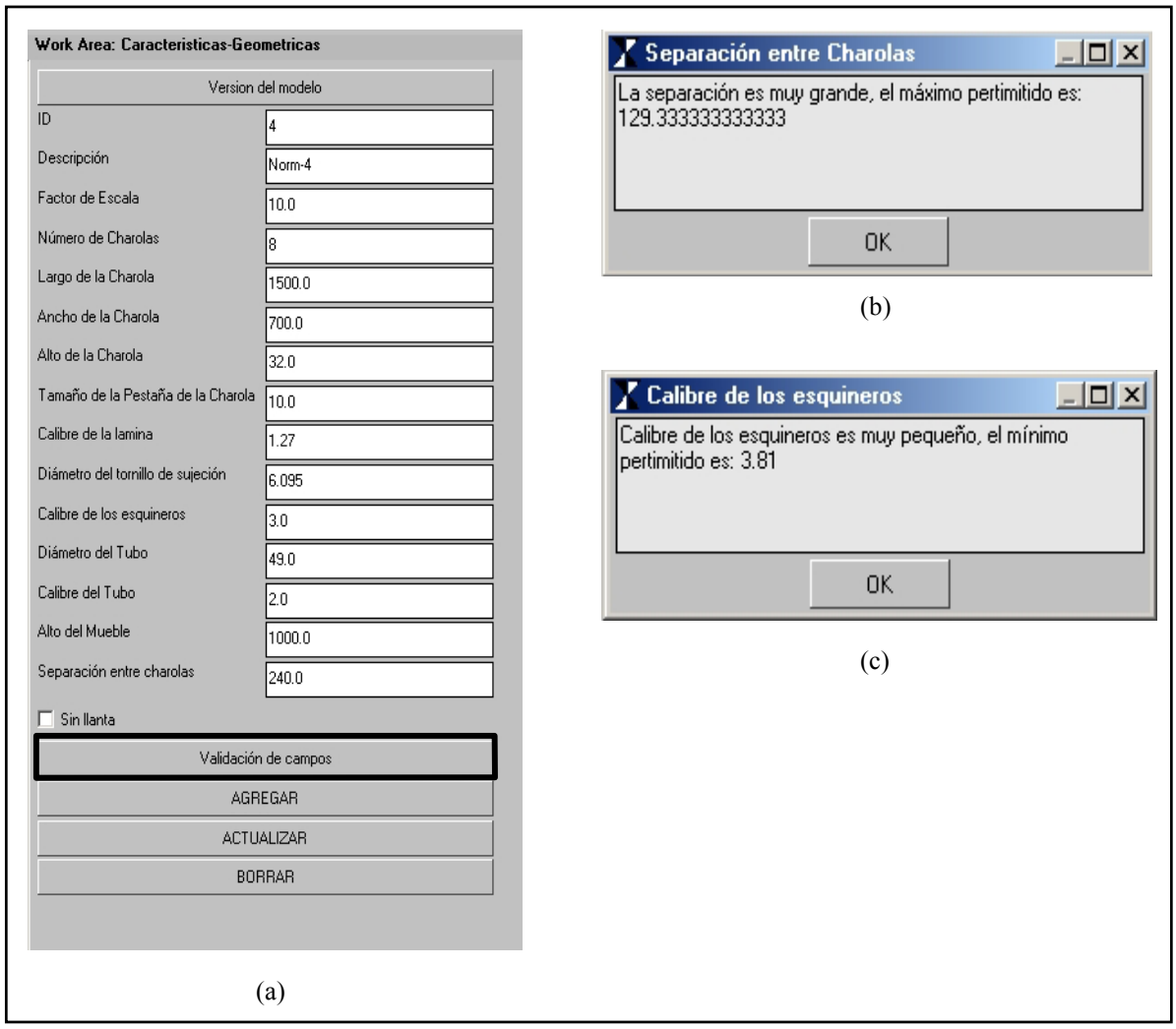

Figura 5.28 Objeto dibujado en el CANVAS

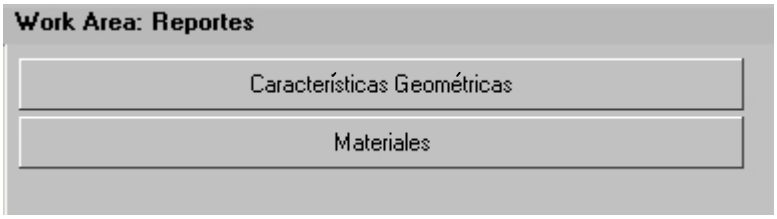

Figura 5.29 Subobjeto reportes

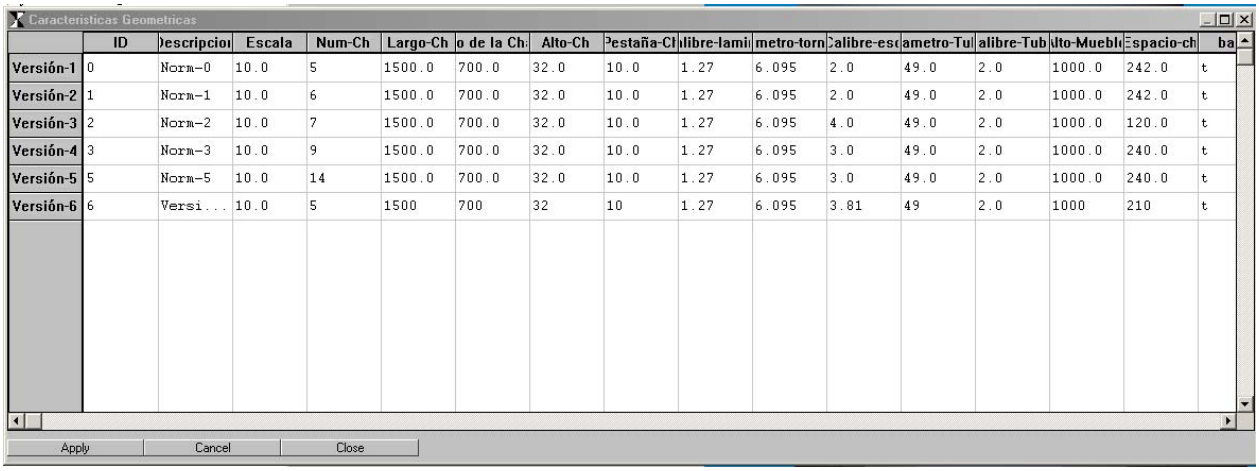

Figura 5.30 Tabla de características geométricas

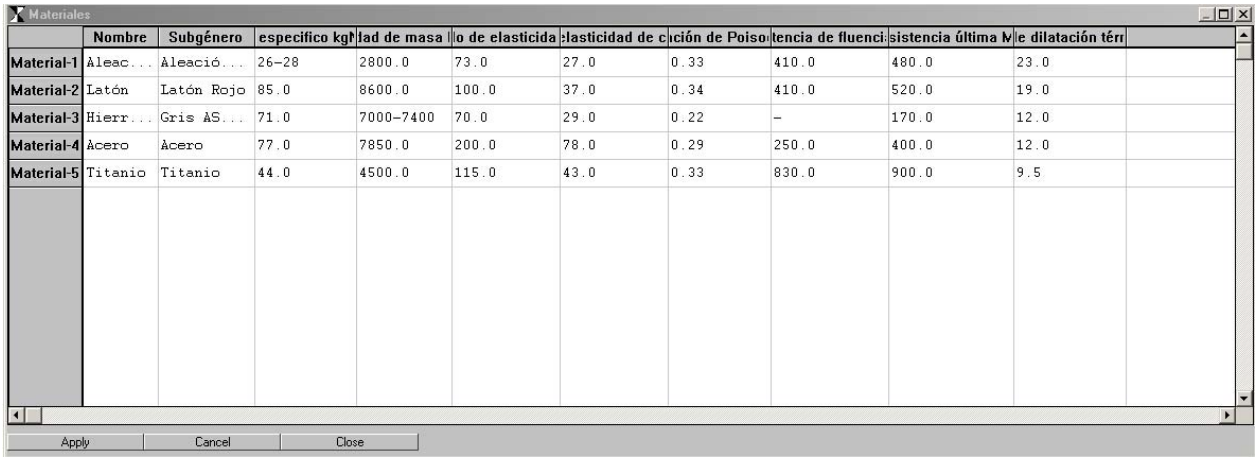

Figura 5.31 Tabla de materiales

En las Figuras 5.32 y 5.33 se muestra un ejemplo de cómo esta cambiando la representación gráfica del producto al cambiar los valores de sus propiedades

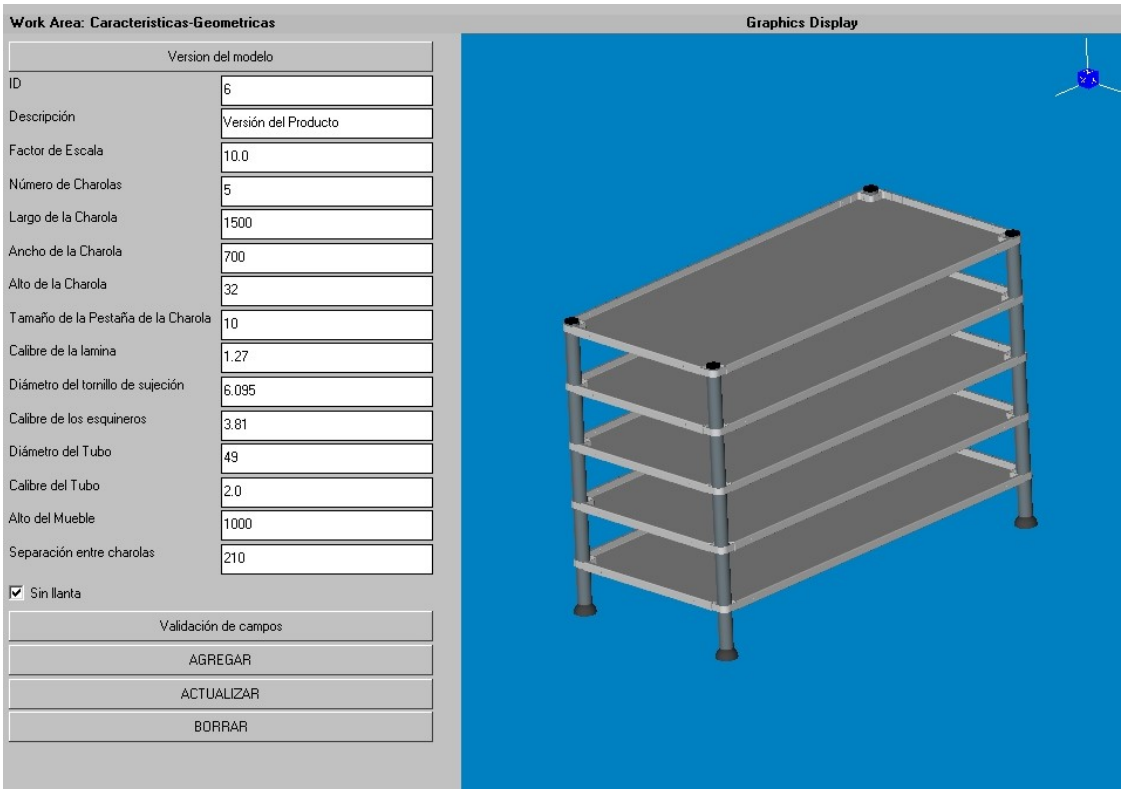

Figura 5.31 Producto con los valores por omisión

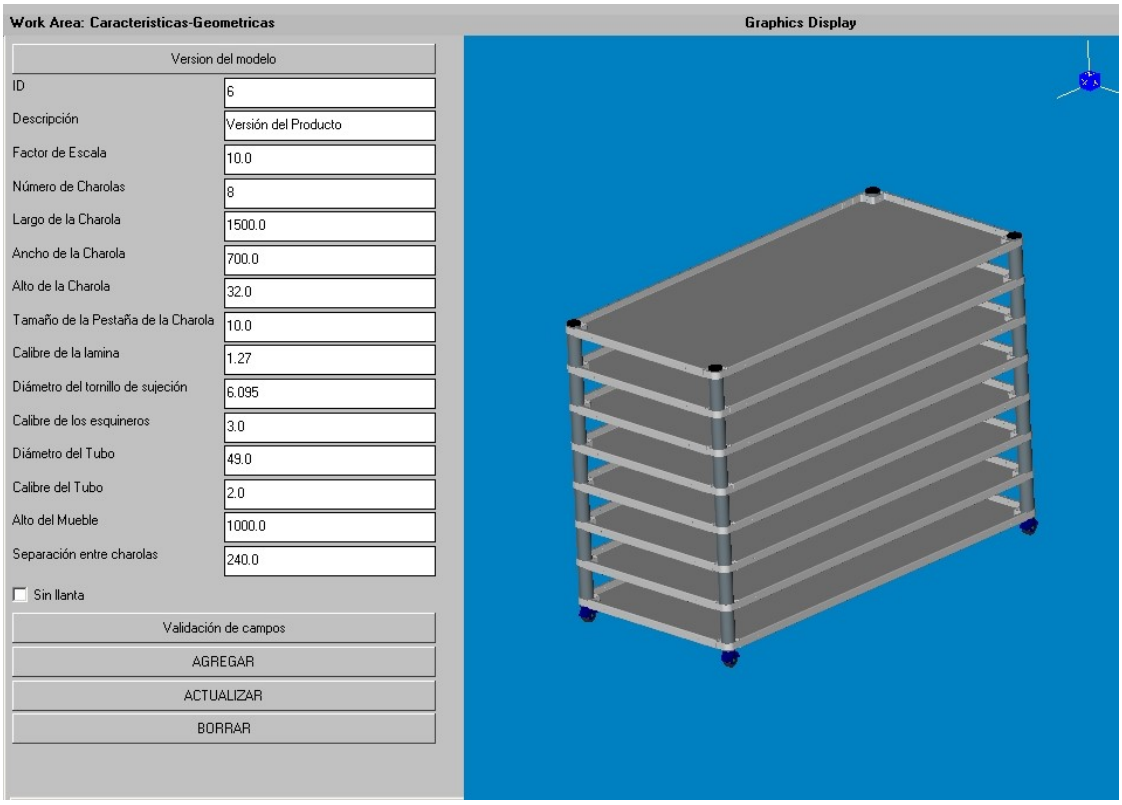

Figura 5.31 Producto con los valores modificados

Conclusiones

En el presente estudio se investigó la factibilidad de implementación del Modelo del Producto haciendo uso del Lenguaje de Modelado Adaptable (AML).

Debido a las características que presenta AML en la manipulación de objetos, generación de sólidos y manipulación de tablas se decidió intentar una implementación del Modelo del Producto.

Para esta implementación se escogió la versión del Modelo del Producto utilizada por el Dr. Jesús Manuel Dorador en su tesis doctoral "Product And Process Information Interaction in Assembly Decision Support Systems", debido a la facilidad de comprensión y a que se disponía de mayor información sobre este. Para hacer esta implementación se simplificó el Modelo del Producto a sus clases más básicas para facilitar el desarrollo del sistema.

Al diseñar la primera parte del sistema, es decir, la construcción del mueble, se cumplió con los requerimientos de tener un ensamble principal, subensambles y componentes ayudando los subobjetos a simplificar el modelado y diseño del sistema en esta etapa.

Una vez que se consiguió modelar gráficamente el caso de estudio en AML, se implementaron las tablas y métodos que manipularían a estas, esto con el fin de simular una base de datos para poder almacenar la información del producto.

Debido a que AML es un lenguaje descriptivo el programador posee un mayor control sobre variables de tipo geométrico y no geométrico, esto es por que él mismo programa las clases, métodos, objetos y subobjetos con sus propiedades, por lo que sabe donde y para que esta usando cada uno de los objetos.

Se logró de forma satisfactoria la manipulación de datos, agregando, borrando, actualizando y validando los registros de las tablas en la interfaz y almacenando los datos en tablas dentro de archivos .txt.

La metodología para modelar producto en AML puede presentar algunas dificultades, sobre todo si lo que se modela es un producto que consta de muchas piezas. La dificultad de esto radica no tanto en el modelado de la pieza ya que con el auxilio de diagramas de esta se simplifica su modelado, la dificultad se presenta cuando se quiere ensamblar todo el producto ya que se tienen que reorientar las piezas y desplazarlas a la posición en la que estarán dentro del ensamble o subensamble. Esta reorientación se lleva a cabo mediante la manipulación de los objetos por medio de código y el resultado se muestra en la interfaz de visualización de AML; esto se debe a que esta interfaz de visualización no sirve para la manipulación o modificación de los objetos, por lo que si se desea hacer algún cambio dentro de una pieza o un ensamble se tiene que regresar al código, modificarla y cargar nuevamente el código para poder ver los cambios realizados.

Otra de las restricciones de AML se presenta al tratar dar dimensiones mayores a las 1000 unidades (cm, mm, etc) al hacer esto, se pueden tener resultados no deseados en la representación grafica a pesar de que la programación sea correcta, es por esto que si se desea utilizar dimensiones mayores a las 1000 unidades se deberá de utilizar un factor de escala para una correcta visualización.

En los manuales de AML se menciona que se puede trabajar con las bases de datos SQL y ODBC (Open DataBase Connectivity), sin embargo para la utilización de estas bases de datos se requiere programar librerías de enlace dinámico (DLL) pero la documentación sobre este tema enfocado a AML es muy escasa. Es por esto que se opto por la implementación de tablas en archivos de texto.

En resumen se puede decir que el modelado de sistemas de información utilizando AML aunque no es sencillo, es posible. Esto trae como ventaja la independencia de sistemas de representación geométrica, programación y bases de datos. Sin embargo esto a su vez podría volverse un problema si es que en algún momento AML deja de funcionar o bien la empresa deja de dar soporte sobre este.

Apéndice A Introducción a AML

AML cuenta con una gran variedad de constructores previamente definidos, el más importante de estos constructores es el ***define-class*** el cual es usado para definir la estructura de una nueva clase, este mismo constructor se ha utilizado para definir las clases que son usadas en la interfaz.

Algunas consideraciones que se deben tener en cuenta para la definición de una clase son las siguientes:

1. Especificar la(s) clase(s) de la cual la nueva clase debe heredar, a la(s) cual(es) se le(s) llama super-clases.
2. Especificar las propiedades de la nueva clase y sus formulas (atributos).
3. Especificar los subobjetos de la nueva clase llamados (children).

Hay que tener en cuanta que si en las propiedades o en los subobjetos se hace referencia a otros objetos o clases, estos últimos deben haberse definido previamente y que los valores que tengan las propiedades por omisión podrán ser sobrescritos si es que así se requiere. Esto no quiere decir que la nueva clase definida no pueda poseer nuevas propiedades, de hecho las propiedades de los subobjetos pueden depender de estas propiedades.

Como se deduce del primer punto una clase siempre tiene que heredar de otra clase sin embargo; si no se tiene o no se quiere que heredé de otra clase, en la definición de esta se debe tratar como un objeto, como se muestra en el ejemplo:

```
(define-class material-object
  :inherit-from (object)
          :properties (
            material
            density  1
            volume   0.3
            weight   (* ^density ^volume)
                   )
  :subobjects(
```

```
        )
      )

(define-class WOODEN-OBJECT
  :inherit-from (material-object clase-1)
        :properties (
          color    'red
          material 'wood
          density  0.03
                )
      )
```

Aquí se están definiendo dos clases: en la primera se define la clase material-object que no está heredando de ninguna clase, es por ello que se coloca la palabra object en los paréntesis de inherit-from, esta clase cuenta con las propiedades de material, densidad, volumen y peso con sus respectivos valores; en los cuales no es necesario declarar el tipo de variable que son, ya que AML los identifica automáticamente; un caso especial es el peso, ya que se calculará en base a las propiedades de densidad y volumen.

En la segunda se define una clase llamada WOODEN-OBJECT la cual hereda de material-object todas las propiedades así mismo define la propiedad de color, asigna el valor de material y reescribe el valor de densidad. Si en algún momento se quiere que esta clase herede de otra clase además de material-object, solo hay que agregarla en forma espaciada entre los paréntesis de inherit-from con lo que hereda los atributos de ambas clases.

Como se puede ver en ambas clases no se están utilizando subobjetos pues en el primero el campo de subobjetos se encuentra vacío, y en el segundo no existe este campo, de lo cual se deduce que no es necesario especificar dicho campo si no se usarán subobjetos.

Aquí se puede ver con más claridad como funciona el cálculo de dependencias, pues si tenemos una clase (clase1) que ha heredado ciertas propiedades de otras clases o en su definición hace referencia a propiedades de otras clases y alguna de estas cambia en cualquiera de estas clases, la propiedad de la clase1 cambia automáticamente, si es que no se determina otra cosa (si se modifica algo en un objeto, los objetos que dependen de él también se van a modificar).

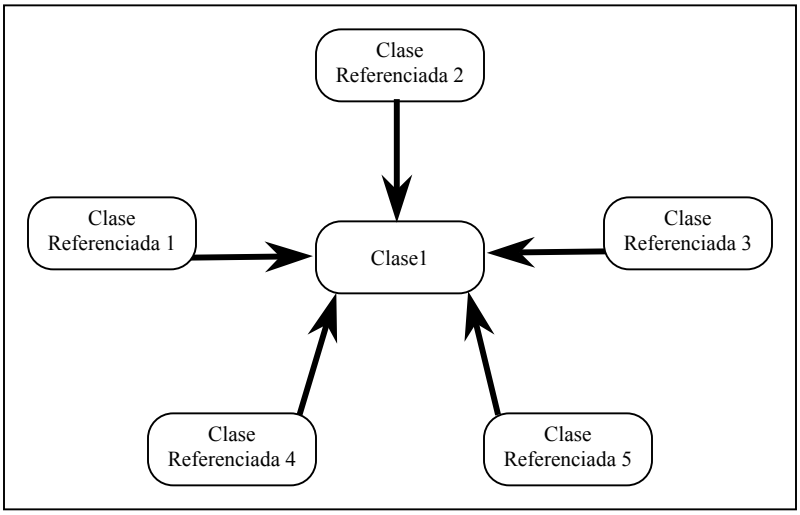

Figura A.1 Dependencias Entre Clases

Niveles de objetos.

Dentro de AML, dependiendo de la forma en que se construya una clase, se tendrán diversos niveles de objetos para esa clase. Mediante el conocimiento de los niveles podemos hacer referencias a propiedades que se encuentren en niveles distintos. Para ilustrar estos niveles y referencias se tiene el siguiente ejemplo:

Supóngase que se tiene la siguiente jerarquía de una clase mostrada en la Figura A.2

Como se puede observar una correcta tabulación es una forma de facilitar la identificación de los diversos niveles. Se puede acceder a atributos, definidos con anterioridad y no necesariamente en el mismo nivel, mediante uno de los nuevos constructores definidos en AML.

El constructor “the” nos ayuda para dar la ubicación exacta de la variable a la cual se requiere hacer referencia. Por ejemplo, ubicándonos en el cuadro anterior, si queremos que

la propiedad “span” de “WING-0001” tenga el mismo valor que la propiedad “wing-span” de “AIRPLANE” la forma de referenciar esto es:

WING-0001

span (the superior superior superior wing-span)

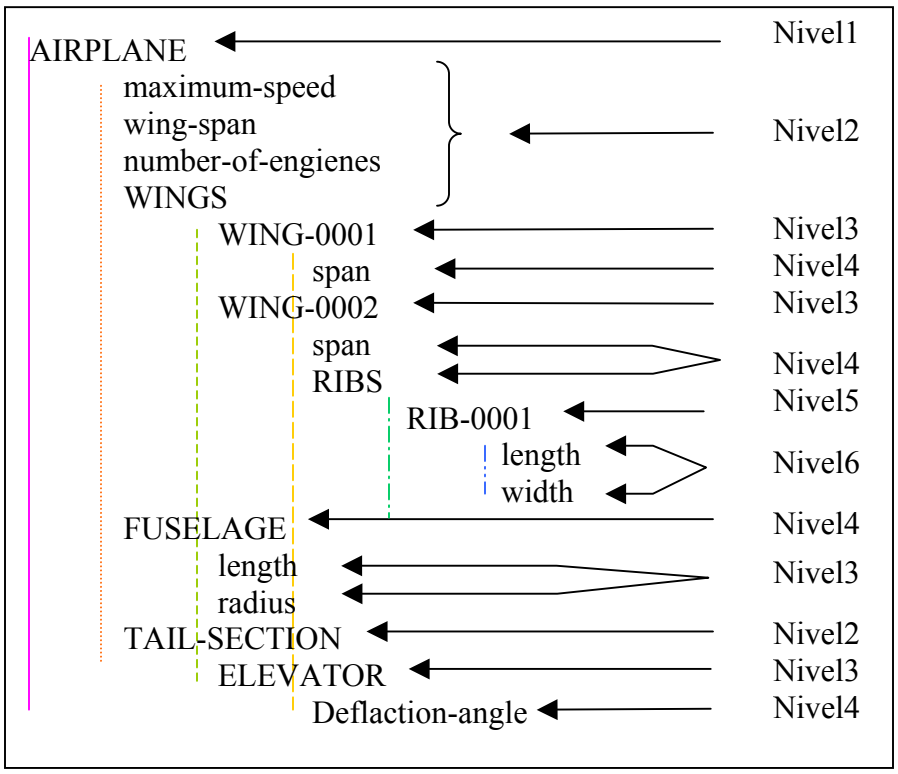

Figura A.2 Niveles De Jerarquía

En esta línea se ve la palabra superior esto nos indica que tenemos que ir al nivel inmediato superior para continuar la referencia. Como se puede apreciar se tiene que subir tres niveles y en el último nivel ir a la propiedad wing-span para tomar el valor.

Esta misma instrucción se puede abreviar utilizando “ ^ ” como si fuera superior por lo que se puede escribir como:

WING-0001

span ^^^wing-span

Este método abreviado solo se puede utilizar cuando se hacen búsquedas en forma ascendente, si se desea hacer una referencia a una propiedad en forma descendente es necesario escribir toda la ruta de acceso sin utilizar los métodos abreviados como ocurre en el siguiente ejemplo:

WING-0002

span (the superior superior WING-0001 span)

En este ejemplo primero se ascienden dos niveles y después se desciende por WING-0001 para llegar a span.

Otra forma de abreviar una referencia ascendente es mediante el símbolo " ! " que significa "the", volviendo al primer ejemplo se puede escribir como:

WING-0001
span !wing-span

Mientras que con la instrucción ^ la búsqueda de la referencia se inicia en el nivel inmediato superior, en este caso la búsqueda se inicia en el mismo nivel en el que se hace la referencia. Otra diferencia es que si no se coloca el numero exacto de ^ para la liga el programa marcara error, mientras que con ! la búsqueda se realizara en forma ascendente hasta encontrar la primera coincidencia de la propiedad referenciada, esta forma de referencia implica una búsqueda por lo que consume un mayor tiempo de procesamiento.

Clases generales de AML

AML cuenta con una gran variedad de clases pero todas se pueden clasificar en dos categorías: las clases básicas y las clases geométricas como se muestra en la Figura A.3

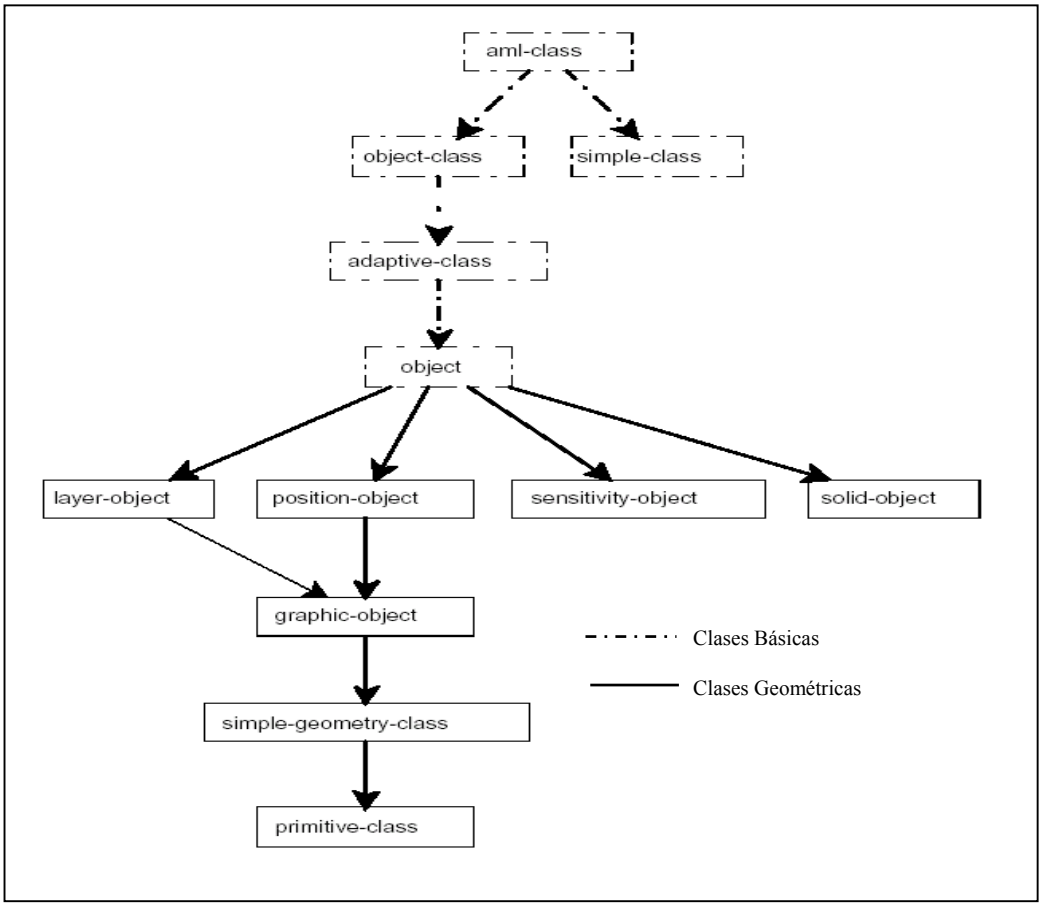

Figura A.3 Clases generales de AML[3]

aml-class

Cada clase de AML hereda de esta clase. Esto es utilizado para uso interno proporcionando a todas las clases la funcionalidad requerida. En otras palabras se puede considerar como la clase patrón en la cual se basarán todas las demás.

object-class

Esta clase proporciona propiedades que se necesitan para el uso interno y la habilidad de rastrear las dependencias. Estas propiedades pueden ser las referenciadas pero no inicializadas por ninguna subclase.

simple-class

Es usada para crear objetos que no tengan ningún mecanismo de dependencia, ni propiedades, ni fórmulas, ni subobjetos. Esto significa que este objeto no dependera de nada y nada dependerá de este objeto. Si esta clase es heredada a una subclase que también herede al object-class sera mediante este que se establescan las dependencias. El propósito de esta clase es proporcionar un mecanismo para almacenar valores en el modelo y no en las variables globales.

adaptive-class

Esta clase proporciona la habilidad de tener una dependencia que notifica cuando las propiedades y los subobjetos son agregados.

Cuando select-object se usa en una propiedad, ninguna dependencia es establecida para rastrear cuando un objeto es agregado. La propiedad add-objects proporciona un medio para establecer una dependencia que se notificará cada vez que un objeto es agregado o borrado.

object

object es la clase de más alto nivel jerárquico que puede ser usado por un usuario es el campo de inherit-from, en otras palabras si el usuario quiere crear una clase que no dependa de otra clase se colocará en el campo de inherit-from la palabra object, con esto se consigue una clase llamada super-class.

layer-object

Esta clase agrega la propiedad de capas (layer) a los objetos gráfico (graphic-objects). Esta propiedad puede usarse para manejar las diferentes capas de objetos en un modelo.

position-object

position-object proporciona a los objetos la habilidad de ser orientados en el espacio. Todos los objetos gráficos heredan de la clase graphic-objects que a su vez hereda de position-object, esto es lo que nos da la facilidad de poder orientar cada uno de los elementos gráficos en base a un sistemas de coordenadas que puede ser también referanciado a un sistema de coordenadas global.

sensivity-object

Este objeto se usará para controlar la selección interactiva de un objeto al desplegarlo en la pantalla. Es decir cuando se hace la representación gráfica del objeto se puede definir si el usuario podrá seleccionar el objeto y el color en que se colocara si es que se puede seleccionar; por omision se pueden seleccionar todos los objetos y se colocan el color rojo al seleccionarlos.

solid-object

Proporciona la facilidad de tratar a un objeto como un sólido o solo como una superficie, por omisión se tratan todos los objetos gráficos como sólidos.

graphic-object

Todos los objetos que esten relacionados con alguna geometría y/o gráficos, deberan heredar de graphic-object, hay que tener en cuanta que por omisión todos los elementos gráficos serán creados tomando como centro el origen global (0.0 0.0 0.0).

Entre las propiedades que se pueden elegir en esa clase se encuantran: el color, si desea que el objeto se despliegue o no, el tipo de tratamiento que se dar al gráfico (sombra, malla, superficies,etc), el tipo de línea con la que se creará, entre otras.

simple-geometry-class

Esta clase se usa por las primitivas del sistema para crear representaciones de marco de alambre rápidas. Cuando sea posible el sistema usará la representación geométrica simple en lugar del verdadero el modelo geométrico para ahorrar tiempo. El sistema usa automáticamente la representación geométrica apropiada para que el usuario no tenga que involucrarse sobre el modelo geométrico usado.

primitive-class

Todos los objetos geométricos son dibujados con una representación simple heredada de esta clase. Cuando la vedadera geometria se requiere se puede mostrar, pero si se usa la geometría simple se representa mas rápido el dibujo.

Funciones en AML

AML tiene implementadas diferentes tipos de funciones, entre otras se incluyen las funciones matemáticas, funciones sobre vectores y listas, y funciones booleanas y de operaciones sobre cadenas:

Funciones Matemáticas

ESCALARES	TRIGONOMÉTRICAS	VECTORES Y MATRICES
+	acos	add-vectors
-	acosd	angle-between-2-vectors
*	asin	cross-product
/	asind	dot-product
abs	atan	identity-matrix
average	atand	mid-point
ceiling/fceiling	cos	multiply
degrees-to-radians	cosd	multiply-vector-by-scalar
double-float	sin	negate-vector
expt	sind	normalize
float	tan	roughly-same-vector
floor/ffloor	tand	round-point
half		subtract-vectors
insure-between		vector-length
log		
max		
min		
mod		
radians-to-degrees		
rem		
roughly-equal		
round/fround		
round-real		
signum		
sqrt		
truncate/ftruncate		
twice		

GEOMETRICAS	TANGENTES E INTERSECIONES	TRIMS
3-point-arc	arc-tangent-to-line-circle	split-arc-by-2-points
3-point-arc-info	arc-tangent-to-line-point	split-arc-by-point
3-point-circle-info	arc-tangent-to-two-lines	split-circle-by-2-points
3-point-on-same-line	circle-circle-intersection	trim-arc/arc
arc-mid-point	circle-tangent-to-line-circle	trim-arc/circle
circles-are-in-same-plane	circle-tangent-to-two-circles	trim-arc/line
distance-from-point-to-plane	circle-tangent-to-two-lines	trim-circle/arc
get-arc-start/end-angle	line-circle-intersection	trim-circle/circle
line-is-in-plane	line-line-intersection	trim-line/arc
line-parallel?	line-tangent-to-circle	trim-line/circle
offset-line	line-tangent-to-two-circles	trim-line/line
point-along-vector	test-flags	
points-distance		
point-is-in-plane		
point-line-distance		
point-on-arc		
point-on-line-segment		
project-point-to-face		
project-point-to-line		
rotate-point		
vector-component-in-plane		
vector-rotation-angle		
vectors-are-parallel		
x-rotate-point		
y-rotate-point		
z-rotate-point		

Funciones de listas, vectores, secuencias y asociación de listas

LISTAS	VECTORES	SECUENCIAS	ASOCIACIÓN DE LISTAS
append	aref	copy-seq	assoc
append-list	empty-vector	elt	cons
butlast	vector	find	copy-alist
copy-list	vector-append	length	rassoc
first	vector-pop	position	
intersection	vector-push	remove	
last	vector-push-new	remove-duplicates	
list	vector-to-list	replace	
list-to-vector		reverse	
member		search	
multiple-value-list		sort	
nth		Subseq	
push			
pushnew			
rest			
set-difference			
set-exclusive-or			
subsetp			
substitute			
union			

Funciones booleanas y operaciones sobre cadenas

COMPARACION ENTRE NÚMEROS	COMPARACIONES GENERALES	COMPARACIÓN DE CADENAS	COMPROBACIÓNDE TIPOS
=	eq	string=	evenp
/=	eql	string/=	floatp
<	equal	string-equal	integerp
>	equalp	string<	listp
<=	typep	string>	minusp
>=		string<=	null
		string>=	numberp
			oddp
			plusp
			stringp
			symbolp
			type-of
			zerop

Además de estas funciones el usuario puede desarrollar sus propias funciones y métodos utilizando una sintaxis muy parecida a la usada en la programación estructurada como ocurre en las funciones condicionales (case, if, etc) o los ciclos de código (for, while, do, repeat, etc).

Tablas

AML utiliza tablas para poder describir fácil y eficientemente la administración de datos, sobre todo al tratar con pequeños volúmenes de información, permitiendo cargarlos y/o almacenarlos en archivos tipo ASCII.

A continuación se dará una breve descripción de los métodos y las clases utilizadas en el manejo de tablas.

record-table-object (clase)

Esta es una tabla que organiza la información en registros, cada registro se constituye de varios campos, el record-table-object es sumamente rápido en el acceso a los registros para modificarlos, no esta optimizado para búsqueda secuencial a través de los registros. El record-table-object utiliza una llave primaria para identificar cada registro. Cada registro tendrá un único valor de llave primaria y propiedad primary-key-field es usada para especificar que campo de registro será usado para representar la llave primaria. El valor de una llave primaria puede ser algún valor AML (número, lista, símbolo, objeto, instancia, etc).

Nota: cuando se almacena una lista, cada uno de los elementos de la lista es almacenado en un campo.

add-record record-table-object (método)

Añade un registro a la tabla especificada en *record-table-object.*

clear-table record-table-object (método)

Borra todos los registros de la tabla especificada en *recod-table-objec.*

retrieve-record record-table-object (método)

Recupera el registro de una tabla especificada en *recod-table-objec* mediante la coincidencia del valor de la llave primaria con el campo de búsqueda.

delete-record record-table-object (método)

Borra el registro de una tabla especificada en *recod-table-objec* mediante la coincidencia del valor de la llave primaria con el campo de búsqueda.

update-record record-table-object (método)

Actualiza el registro de una tabla especificada en *recod-table-objec* mediante la coincidencia del valor de la llave primaria y el campo de búsqueda; incluso el valor de la llave primaria.

select-table record-table-object (método)

Pregunta por un *record-table-object* y regresa una lista de con las coincidencias de la pregunta.

load-table record-table-object (método)

Carga tabla que se encuentra en la ruta especificada, la información dentro conservara el orden en que se fueron almacenando los registros, cabe mencionar que para poder cargar una tabla antes debió ser salvada usando el método *save-table.*

save-table record-table-object (método)

Salva el estado actual de una tabla dentro del archivo en la dirección especificada.

flat-file-record-table-class (clase)

Un *flat-file-record-table-class* es un *record-table-object* que puede cargar un archivo de formato texto y poblarlo automáticamente. El archivo de texto deberá tener un registro por línea y los campos de cada registro deberán ser separados por un delimitador específico. El delimitador se define en la propiedad *field-delimiter*. Exceptuando el coma como delimitador, las ocurrencias adyacentes de delimitadores (tabulador, espacio) serán consideradas como un solo delimitador. El método *save-table* ha sido redefinido sobre *flat-*

file-record-table-class para guardar la tabla en el mismo formato en que se especifico en los valores de las propiedades y no en el formato definido dentro de *record-table-object.*

El método *select-table* ha sido redefinido para esta clase para devolver un registro en el orden en que aparece dentro del archivo de texto.

insert-record flat-file-record-table-class (método)

Este método inserta un registro dentro de *flat-file-record-table-class* en una posición específica. Esta inserción es equivalente a *add-record;* solo que el posicionamiento se ve reflejado en la forma en que se construye la tabla, por lo que cuando se salve o se cargue la tabla se conservara el orden que se le haya dado y no el orden de inserción.

Interfase GUI de AML

En esta sección se dará una breve introducción a los principales comandos para el desarrollo de una interfaz de usuario en AML, esto con el fin de hacer al sistema mas amigable al usuario.

model-interface-property-class

Esta es la super-clase de todas las clases del modelo de la interfaz GUI. Generalmente esta clase no es instanciada por el usuario.

Sus propiedades son:

- available?: Bandera (t o nil) especificando si la propiedad esta disponible sobre los valores actuales de las otras propiedades del modelo de datos.
- label: Cadena que especifica la etiqueta de la propiedad.

access-property-class

Esta clase es usada para propiedades en cuya fórmula se llama a un método específico en un objeto específico. Esto es manejado por el access-object y access-method. El usuario no debe redefinir la fórmula o cambiar el valor de un access-property-class; la entrada del usuario se restringe a especificar un access-object y un access-method.

Una propiedad de acceso se representa en la GUI por un botón de acción que activa el access-method.

Sus propiedades son:

- Access-method Nombre del método de acceso que la fórmula activa. El método de acceso debe definirse en la clase (o una super-clase) del access-object.
- Access-object Instancia del Objeto al que access-method es llamado. Botones que ejecutan un método.

Figura A.4 Representación de la clase access-property-class

data-model-node-mixin

Esta clase es diseñada para representar un nodo de un árbol de un modelo de datos que contiene aplicaciones específicas de la descripción sobre el nodo que representa. También maneja las propiedades de ese nodo que son una parte del modelo de datos y ésta debe ser una parte de la aplicación GUI, es decir cuando se hace la expansión de una clase o subobjeto en la interfaz de usuario se puede con los elementos expandidos. Las clases definidas por el usuario heredan de data-model-node-mixin. La aplicación de la forma estándar del modelo application-forms (descrita mas adelante) hace uso de los nodos del árbol de construcción del modelo para la generación automática de la interfaz GUI. La

clase application-form que incluye data-model es diseñada para la interfase del data-model-node-mixin.

Sus propiedades son:

- available? El valor por defecto es t. Cuando es nulo, no permite el acceso GUI a property-object-list del nodo del árbol de modelo de datos.
- label Etiqueta del nodo es desplegada en el árbol de modelo de datos GUI.
- property-objects-list Lista de instancias de model-interface-property-class que define a las propiedades de data-model de este nodo. Las propiedades incluidas en esta lista estarán disponibles en la forma data-model-node-mixin automáticamente generadas. La lista sigue el mismo formato de property-objects-list descrito con ui-multipleproperty-subform-class.
- subnode-object-list Instancia un subobjeto de este nodo que es automáticamente demandada cuando el árbol del modelo GUI es desplegado. La fórmula por defecto es:
- ((children (the superior) :class 'data-model-node-mixin))
- subobject-class-name Esta propiedad especifica la clase del objeto que estará disponible al agregar un subobjeto a este nodo de un modelo de datos GUI. . El nombre de la clase acepta un símbolo, una lista de nombres de la clase para permitir más de 1 clase o una lista de clases en pares.

Ejemplo:

'(box-object cylinder-object) or '(("Box" 'box-object) ("Cylinder" 'cylinder-object))

editable-data-property-class

Esta clase se usa para propiedades de uso general cuyos valores serán editados por el usuario. Son generalmente propiedades de entrada.

Sus propiedades son:

- validation-function Nombre de la función usada para validar la entrada del usuario. Esta función es definida por el usuario y debe tener un argumento que es el valor de entrada proporcionado por el usuario. Esta función debe devolver un valor no nulo para que la entrada sea aceptada. La función predefinida es nula (no función) qué acepta toda la entrada del usuario.
- validation-error-message Cadena que especifica el mensaje del error que el estallido-altos cuando la función de aprobación es ejecutada y regresa un valor nulo.

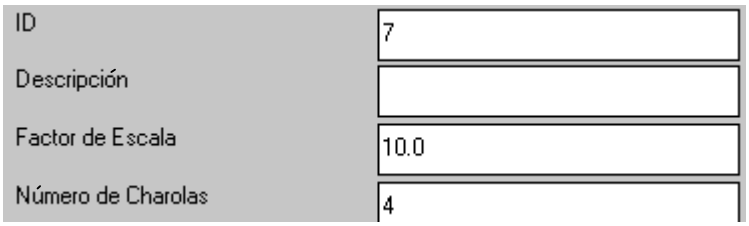

Figura A.5 Representación de la clase editable-data-property-class

computed-data-property-class

Esta clase es usada para propósitos generales en el que los valores no serán supuestamente editados por el usuario. Sus propiedades son generalmente de salida.

Figura A.6 Representación de la clase computed-data-property-class

flag-property-class

Esta clase es usada para propiedades "bandera" que esperan un valor de t o nil. Sus propiedades son generalmente representadas en la GUI con una casilla de verificación.

Figura A.7 Representación de la clase flag-property-class

data-matrix-property-class

Esta clase es usada para propiedades que esperan un valor en una lista de listas representadas como una matriz o una tabla de datos. Cada lista dentro de la lista del paréntesis representa un registro.

Sus propiedades son:

- Mode La forma automática de representar la tabla por defecto es 'standart: Si el modo es 'standart la matriz de datos aparece en línea (aparece sin ser llamada). La otra forma es el modo 'pop-up, esta propiedad hace que aparezca un botón en la interfaz que muestra una matriz que es la representación de la tabla de datos.
- columns-labels-list: Lista de cadenas especificando las etiquetas de las columnas (campos) de la matriz (tabla).
- Rows-labels-list Lista de cadenas especificando las etiquetas de renglones (registros) de la matriz (tabla).

Figura A.8 Representación De La Clase data-matrix-property-class

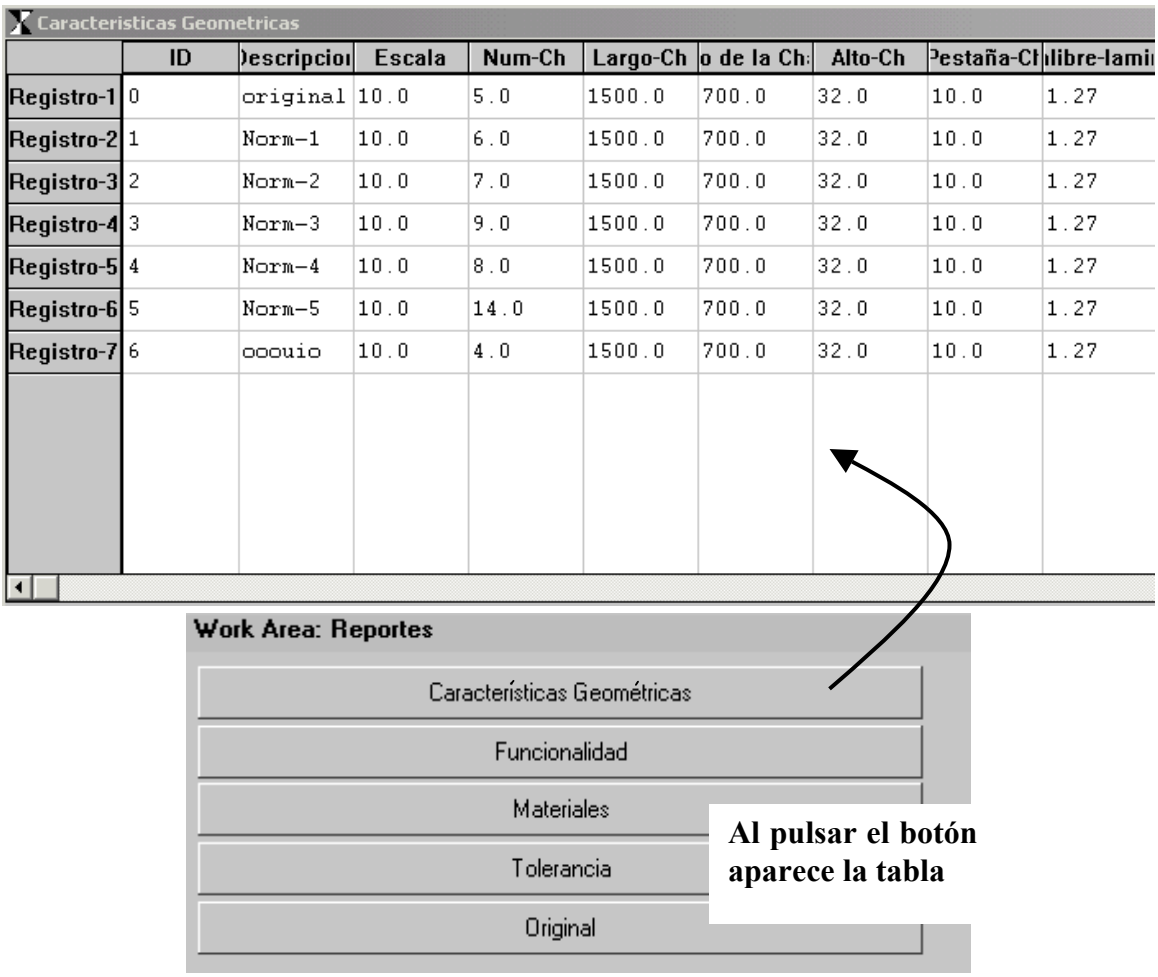

Figura A.9 Representación de la clase data-matrix-property-class

Agrupando Expresiones

Los siguientes constructores son usados para agrupar más de una expresión en una sola expresión agrupada, y también para controlar el valor que regresa. Al agrupar expresiones es frecuente usar funciones condicionales simples como el "*if*", o bien trabajar con las enunciados "*then" y else* del *if.*

progn

Este constructor agrupa una lista de expresiones para ejecutarlas secuencialmente y regresa la última expresión del grupo.

Por ejemplo:

```
AML> (defun regresa-mayor (a b)
      (if (> a b)
            (progn
                  (format t "Valor a es mayor, entonces el valor a es")
                  a
                  )
            (progn
                  (format t "Valor b es mayor, entonces el valor b es")
                  b))
      )

AML> (regresa-mayor 5 6)

      Valor b es mayor, entonces el valor b es
6
AML>
```

prog1

Este constructor agrupa una lista de expresiones para ejecutarlas de forma secuencial y regresa la primera expresión del grupo. Su formato es similar a la instrucción progn.

geom-copy-object [Class]

La clase geom-copy-object crea la geometría para ser copiada de un objeto. La geometría copiada depende de la geometría original y los cambios hechos al original son reflejados en la copia.

Sus propiedades son:

- source-object Una instancia de la cual creará una copia de la geometría.

Apéndice B Introducción a UML

Lenguaje Unificado de Modelado (Unified Modeling Language, UML) es una especificación de notación orientada a objetos. Se basa en las anteriores especificaciones BOOCH, RUMBAUGH y COAD-YOURDON. Cada uno de estos métodos ya estaba evolucionado y se unieron para seguir con esa evolución en vez de hacerlo por separado eliminando la posibilidad de cualquier diferencia que confundiría más aún a los usuarios. UML divide cada proyecto en un número de diagramas que representan las diferentes vistas del proyecto. Estos diagramas juntos son los que representan a la arquitectura del proyecto.

El propósito de UML es visualizar, especificar, construir y documentar sistemas orientados a objetos.

La visualización, especificación, construcción y documentación de un sistema con gran cantidad de software requiere que el sistema sea visto desde varias perspectivas. Diferentes usuarios (usuarios finales, analistas, desarrolladores, integradores de sistemas, encargados de la documentación técnica y jefes de proyectos) siguen diferentes agendas en relación al proyecto, y cada uno mira a ese sistema de forma diferente en diversos momentos a lo largo de la vida del proyecto.

Con UML, se puede diseñar un sistema que no este centrado únicamente en el diagrama de clases, que solamente nos muestra una visión estática del sistema. UML introduce nuevos diagramas que representan una visión dinámica del sistema, es gracias a esto que se puede ver como se propagarían los errores o que partes deben ser sincronizadas.

Gracias a UML se puede unificar el lenguaje de modelado para el desarrollo del sistema, esto nos trae como beneficio el hecho de hacer solo un tipo de documentación, ya que cualquiera que tenga conocimientos de UML podrá entender el desarrollo del sistema.

UML es hoy en día un lenguaje de modelado estándar ya que se puede aplicar a cualquier tipo de proyecto ya que no depende de las características de los objetos, es decir, lo mismo se puede aplicar a un proyecto de computación que a un proyecto arquitectónico. Lo importante de este lenguaje es identificar perfectamente los elementos que describirán el sistema y como se relacionaran entre si.

Aprender a aplicar UML de un modo eficaz comienza por crear un modelo conceptual del lenguaje, lo cual requiere aprender tres elementos principales: los bloques básicos de construcción de UML, las reglas que dictan como pueden combinarse esos bloques y algunos mecanismos comunes que se aplican a lo largo del lenguaje.

La explicación se basará en los diagramas, en lugar de vistas o anotación, ya que son estos la esencia de UML. Cada diagrama usa la anotación pertinente y la suma de estos diagramas crean las diferentes vistas. Las vistas existentes en UML son:

- Vista casos de uso: Comprende aquellos casos que describen el comportamiento del sistema tal y como es percibido por los usuarios finales, analistas y encargados de las pruebas.
- Vista de diseño: Comprende las clases, interfaces, y colaboraciones que forman el vocabulario del problema y su solución. Soporta principalmente los requisitos funcionales del sistema, entendiendo por ello los servicios que el sistema debe proporcionar a sus usuarios finales.
- Vista de procesos: Comprende los hilos y procesos que forman los mecanismos de sincronización y concurrencia del sistema. Cubre principalmente el funcionamiento, capacidad de crecimiento y rendimiento del sistema.
- Vista de implementación: Comprende aquellos componentes y archivos que se utilizan para ensamblar y hacer disponible el sistema físico. Se ocupa principalmente de la gestión de configuración de las distintas versiones de un sistema, a partir de componentes y archivos un tanto independientes y que pueden ensamblarse de varias formas para producir un sistema en ejecución.

- Vista de despliegue: Contiene los nodos que forman la topología sobre la que se ejecuta el sistema. Se ocupa principalmente de la distribución, entrega e instalación de las partes que constituyen el sistema físico.

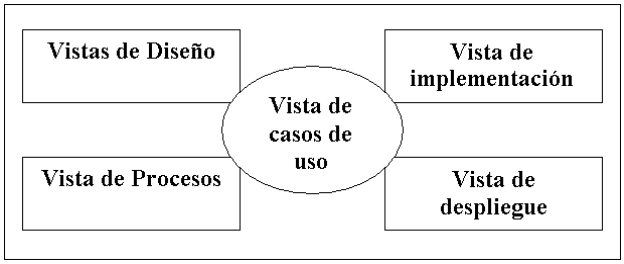

Figura B.1 Vistas existentes en AML.[1]

Se dispone de dos tipos diferentes de diagramas los que dan una vista estática del sistema, es decir, el aspecto de un sistema en el que se destaca su estructura y los que dan una visión dinámica, es decir, aspecto de un sistema que destaca su comportamiento.

Los diagramas estáticos son:

- Diagrama de clases: muestran las clases, interfaces, colaboraciones y sus relaciones. Son los más comunes en el modelado de sistemas orientados a objetos y dan una vista estática del proyecto.

  Es un tipo especial de diagrama y comparte las propiedades comunes al resto de los diagramas (un nombre y un contenido gráfico que es una proyección de un modelo). Lo que distingue a un diagrama de clases de los otros tipos de diagramas es su contenido particular.

  Los diagramas de clases también pueden contener paquetes o subsistemas, los cuales se usan para agrupar los elementos de un modelo en partes más grandes. A veces se colocan instancias en los diagramas de clases, especialmente cuando se quiere mostrar el tipo de una instancia.
- Diagrama de objetos: Muestra las instancias y como se relacionan entre ellas. Se da una visión de casos reales.

- Diagrama de componentes: Muestra la organización de los componentes del sistema. Un componente se corresponde con una o varias clases, interfaces o colaboraciones.
- Diagrama de despliegue.: Muestra los nodos y sus relaciones. Un nodo es un conjunto de componentes. Se utiliza para reducir la complejidad de los diagramas de clases y componentes de un gran sistema. Sirve como resumen e índice.
- Diagrama de casos de uso: Muestra los casos de uso, actores y sus relaciones. Además de quien puede hacer que y cuales relaciones existen entre acciones (casos de uso). Son muy importantes para modelar y organizar el comportamiento del sistema.

Los diagramas dinámicos son:

- Diagrama de secuencia, Diagrama de colaboración: Muestran a los diferentes objetos y las relaciones que pueden tener entre ellos, los mensajes que se envían entre ellos. Son dos diagramas diferentes, que se puede pasar de uno a otro sin perdida de información, pero que nos dan puntos de vista diferentes del sistema. En resumen, cualquiera de los dos es un Diagrama de Interacción.
- Diagrama de estados: Muestra los estados, eventos, transiciones y actividades de los diferentes objetos. Son útiles en sistemas que reaccionen a eventos.
- Diagrama de actividades: Es un caso especial del diagrama de estados. Muestra el flujo entre los objetos. Se utiliza para modelar el funcionamiento del sistema y el flujo de control entre objetos.

Como podemos ver el número de diagramas es muy alto, en la mayoría de los casos excesivos, y UML permite definir solo los necesarios, ya que no todos son necesarios en todos los proyectos.

Ciclo de vida del desarrollo del software.

Existen cuatro fases del ciclo de vida del desarrollo del software: iniciación, elaboración, construcción y transición.

La iniciación es la primera fase del proceso, cuando la idea inicial para el desarrollo se lleva al punto de estar (al menos internamente) suficientemente bien fundamentada para garantizar la entrada en la fase de elaboración.

La elaboración es la segunda fase del proceso, cuando se definen la visión del producto y su arquitectura. En esta fase se expresan con claridad los requisitos del sistema y se utilizan para crear una sólida base arquitectónica. Los requisitos de un sistema pueden variar desde enunciados de carácter general hasta criterios precisos de evaluación, especificando cada uno un comportamiento funcional o no funcional y proporcionando una referencia para las pruebas.

La construcción es la tercera fase del proceso, cuando el software se lleva desde una base arquitectónica ejecutable hasta su disponibilidad para la comunidad de usuarios. Los requisitos del sistema y especialmente sus criterios de evaluación son constantemente examinados frente a las necesidades del proyecto, y los recursos se asignan al proyecto de forma apropiada para atacar los riesgos.

La transición es la cuarta fase del proyecto, cuando el software es puesto en las manos de la comunidad de usuarios. Durante esta fase el sistema es mejorado continuamente, se erradican errores de programación, y se añaden características que no se incluían en una versión anterior.

Bibliografía y Referencias

Libros

[1] G.Booch, J. Rumbaungh, I. Jacobson; El Lenguaje Unificado De Modelado, ed. Addison Wesley Iberoamericana; Madrid 1999.

[2] Dorador González Jesús Manuel, Product and Process Information Interaction in Assembly Decision Support System, Junio 2001

[3] aml3-3-reference-manual.pdf, TechnoSoft Inc. 7 de marzo de 2002

Sitios en Internet

[4] http://www.sercobe.es/espejo/produccion/ingconcu/tutorial/ingconcu.htm última consulta 14 de enero de 2004

[5] http://usuarios.lycos.es/oopere/uml.htm última consulta 15 de enero de 2004

[6] http://www.monografias.com/trabajos/objetos/objetos.shtml última consulta 10 de marzo de 2003

[7] http://members.tripod.com/admusach/doc/ingconc.htm última consulta 14 de enero de 2004