INSTITUTO TECNOLOGICO Y DE ESTUDIOS
SUPERIORES DE MONTERREY
CAMPUS MONTERREY
DIVISION DE GRADUADOS EN ELECTRONICA,
COMPUTACION, INFORMACION Y COMUNICACIONES

GENERACION AUTOMATICA DE PREDICTORES DE TIEMPO DE MAQUINADO PARA FRESADORAS DE ALTA VELOCIDAD

T E S I S

PRESENTADA COMO REQUISITO PARCIAL
PARA OBTENER EL GRADO ACADEMICO DE

MAESTRA EN CIENCIAS
ESPECIALIDAD EN TECNOLOGIA INFORMATICA

HILDA PATRICIA PARRA DIAZ

DICIEMBRE DE 2000

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS MONTERREY

DIVISIÓN DE GRADUADOS EN ELECTRÓNICA, COMPUTACIÓN, INFORMACIÓN Y COMUNICACIONES

GENERACIÓN AUTOMÁTICA DE PREDICTORES DE TIEMPO DE MAQUINADO PARA FRESADORAS DE ALTA VELOCIDAD

TESIS

PRESENTADA COMO REQUISITO PARCIAL PARA OBTENER EL GRADO ACADÉMICO DE

MAESTRA EN CIENCIAS

ESPECIALIDAD EN TECNOLOGÍA INFORMÁTICA

HILDA PATRICIA PARRA DÍAZ

DICIEMBRE DE 2000

GENERACIÓN AUTOMÁTICA DE PREDICTORES DE TIEMPO DE MAQUINADO PARA FRESADORAS DE ALTA VELOCIDAD

Por

Lic. Hilda Patricia Parra Díaz

Tesis

Presentada a la División de Graduados en Electrónica, Computación, Información y Comunicaciones

Como requisito parcial para obtener el grado académico de

Maestra en Ciencias
con la Especialidad en Tecnología Informática

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

Diciembre de 2000

A Dios por quien soy y por quien vivo

A mis padres Pedro y Arminda

A mis hermanos Sergio, Nelia Noemí y Pedro Eloy

A mis sobrinos Oscar Iván, Larisza Noemí y Sergio Ulises

Gracias

Reconocimientos

Al Lic. Guillermo Jiménez Pérez por su paciencia y guía para la realización del presente trabajo.

Al Dr. Ciro Rodríguez González por su apoyo durante el proceso de implementación de la herramienta realizada en esta tesis.

Al Ing. Miguel de Jesús Ramírez Cadena por la revisión final de la tesis y el apoyo otorgado.

A mis compañeros de clases que siempre estuvieron dispuestos a sacarme de dudas y brindarme toda la ayuda que necesité.

A mis compañeros de trabajo por todo el apoyo que me brindaron y muy especialmente a Patricia Garza Gallardo, por la amistad que me ha demostrado y porque siempre estuvo dispuesta a apoyarme en mis actividades laborales, a fin de que pudiera realizar actividades de la tesis.

A mis amigos Angelina Muñoz Soto, Teresa Barragán Sánchez, Cuauhtémoc Cázares Flores y Vladimir Zahuindanda Rivera Arzola por los maravillosos momentos que he compartido con ustedes y por enseñarme que los amigos son también una bendición de Dios.

A toda mi familia, gracias por el cariño y apoyo que siempre me han brindado y que ha sido una de las principales motivaciones de mi vida.

A Dios por el inmenso amor que sin duda me tiene y por haberme permitido realizar una meta más en mi vida.

Hilda Patricia Parra Díaz

instituto Tecnológico y de Estudios Superiores de Monterrey
Diciembre de 2000

Resumen.

El presente documento muestra la utilización de los enfoques generativos implementados en el dominio de los sistemas de predicción de tiempos de maquinado en fresadoras de alta velocidad.

Describe cómo realizar un análisis del dominio a fin de determinar la variabilidad de los diferentes sistemas miembros de la misma familia de aplicaciones con el propósito de definir una arquitectura que permita manejar las diferencias y reutilizar las partes comunes que se identificaron durante el análisis del dominio.

Basándose en la arquitectura definida, se implementó una herramienta de generación de sistemas que automatiza el proceso de creación de nuevos miembros de la familia a partir de definiciones abstractas de los mismos.

Para la realización del trabajo aquí descrito se usaron herramientas y tecnologías como los compiladores, la programación orientada a objetos y el lenguaje de modelación de sistemas UML.

Índice General

Índice de Tablas

Índice de Figuras

CAPÍTULO I

INTRODUCCIÓN.

La correcta definición de un problema y su entorno es un punto de partida que si está bien fundamentado es un gran avance en el planteamiento de su solución. En este capítulo se presenta el problema objeto de la investigación, y se justifica la necesidad que existe de buscar soluciones para el mismo. Así mismo, se plantea una solución y se describe la metodología empleada para llegar a ella.

1.1. PROBLEMÁTICA DEL DESARROLLO DE SOFTWARE.

Dentro de los métodos tradicionales de desarrollo de sistemas de software grandes, generalmente no hay ninguna actividad para identificar las necesidades potenciales, pero no especificadas de los clientes, y mucho menos, necesidades comunes de clientes similares. La actitud predominante es que cada implementación debe realizarse perfectamente adecuada a las necesidades de cada cliente. Más aún, no se considera el costo efectivo de proveer implementaciones que vayan más allá de las necesidades percibidas por el cliente, ya sea para acomodar las posibles necesidades futuras del cliente o para permitir que la implementación se vuelva a usar.

Los procesos de desarrollo actuales se caracterizan por desarrollar un sistema individual para una necesidad específica y una variación del mismo implica el desarrollo de otro sistema que también resultará adecuado sólo a la necesidad que enfrenta.

Lo que sucede en el "mundo real" es que las necesidades del cliente están en constante cambio. El conjunto final de necesidades que se tienen, es diferente de las necesidades percibidas originalmente. Más aún, las implementaciones para un cliente forman las bases para "cortar y pegar" difiriéndolas a las necesidades de otros clientes. Debido a la falta de previsión, el impacto de acomodar esas necesidades no planeadas es, generalmente, bastante severo. El resultado es que los costos de mantenimiento son desproporcionadamente altos con respecto de los costos de desarrollo original. Además, los costos del nuevo desarrollo son extremadamente altos a pesar de la aparente similitud con el software previamente desarrollado.

En muchos casos, los desarrolladores de software, perciben solamente el producto destinado a un cliente final. Generalmente no se reconoce que cada componente dentro de un sistema de software es también un "producto" pero para un cliente diferente. El cliente que tiene que usar un componente individual es quien en el futuro dará mantenimiento y reusará ese componente. El potencial de aplicabilidad de un componente puede ser mucho más amplio que sólo un cliente final. Las necesidades potenciales de ese componente deberían ser entendidas y explotadas.

Para enfrentar apropiadamente estos problemas, el software debe desarrollarse con otro enfoque más orientado al reuso de componentes y los cambio potenciales de los sistemas. Los enfoques de desarrollo de software deben reconocer que los desarrolladores, los que dan mantenimiento y los que reusan, son los clientes reales de los componentes de software, y que el usuario final es solamente el cliente del producto de software completo desarrollado [Ron; 1991].

1.2. FAMILIAS DE PRODUCTOS Y PROGRAMACIÓN GENERATIVA.

El problema de la variabilidad no sólo se presenta en la industria del software sino en muchas otras como la automotriz y la aeroespacial. En estas áreas, los esfuerzos se han concentrado en manejar los cambios continuos que se presentan a fin de ganar ventaja competitiva es decir, ser capaz de producir productos personalizados de manera rápida. Y como en otras áreas, las demandas competitivas están empujando también a los desarrolladores de software a crear con mayor rapidez que antes, productos funcionales, fáciles de usar y confiables y que tengan la capacidad de adaptarse y evolucionar conforme a las necesidades cambiantes de los clientes.

Un sistema de software que tenga las características antes mencionadas requiere de un cuidadoso proceso de ingeniería. Sin embargo, la presión por la rapidez obliga a sacrificar el cuidado en la ingeniería y se pasan de largo aspectos del sistema que son comunes a otros sistemas. Así, cada vez que se requiere un sistema similar o una nueva versión de uno anterior, hay que enfrentar los mismos problemas y se empieza a trabajar desde cero.

Para conciliar este dilema se han explorado ampliamente métodos de producción rápida con ingeniería cuidadosa muchos de los cuales están basados en el concepto de líneas o familias de productos. En este enfoque la ingeniería de software de divide en dos partes: la ingeniería de dominio y la ingeniería de aplicación. La primera define una familia y crea las facilidades de producción de la familia y la segunda usa esas facilidades para crear nuevos miembros de la familia [Weiss; 1999].

El desarrollo de software orientado a familias fue sugerido a principios de 1968. La meta general de esos enfoques es hacer el desarrollo de software más eficiente por medio de especializar un dominio particular de facilidades, herramientas y procesos que se pueden usar para producir software [Weiss; 1999].

Sin embargo, dentro de la industria del software este enfoque, a pesar de no ser algo nuevo y existir trabajos serios desde hace ya bastante tiempo, no es muy aplicado dentro de la práctica ingenieril.

Lucent Technologies es una de las pocas compañías que han aplicado procesos basados en familias y, de acuerdo con ellos, se ha obtenido un decremento en el costo y tiempo de desarrollo para miembros de la familia de entre 60% y 70% con respecto a la

producción de software que no se basa en familias sino en sistemas específicos que enfrentan necesidades específicas [Weiss; 1999].

Si se asume que el costo de producir un nuevo miembro de una familia sin la ingeniería de dominio es constante (C_T), entonces el costo total de producir N miembros de esa familia es N* C_T [Weiss; 1999] (Figura 1.1).

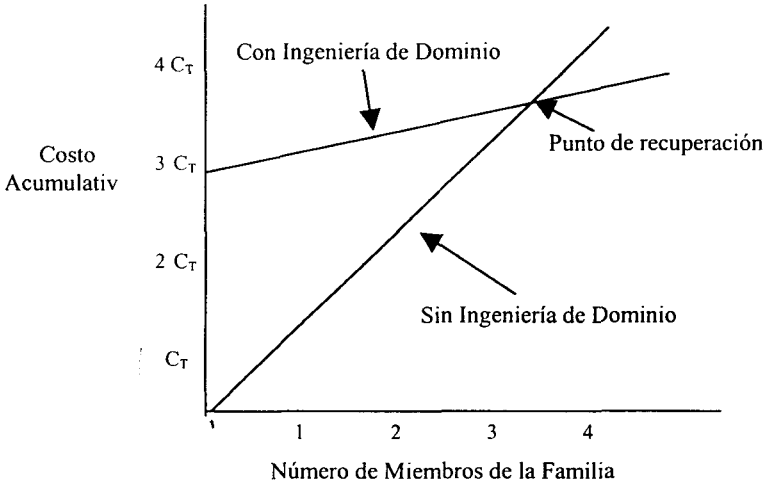


Figura 1.1 Costos de Crear Miembros de una Familia, con Ingeniería de Dominio y Sin Ingeniería de Dominio.

Por otra parte, cuando se usa ingeniería con dominio, el costo inicial es mayor debido a que es necesario realizar un análisis de requerimientos no de un sistema, sino de un conjunto de sistemas similares para determinar sus puntos en común y sus diferencias y establecer los componentes que conformaran a esa familia. El primer miembro de esa familia tiene un costo muy elevado, pero conforme se van creando nuevos miembros el costo se va amortizando debido a que cada nuevo miembro requiere menos análisis de los requerimientos, menos rediseño, recodificación y pruebas lo cual permite la creación más rápida de nuevos miembros.

Un miembro de la familia de sistemas se conforma de varios componentes previamente producidos y agregados a una librería de componentes. Los componentes se diseñan de acuerdo a las necesidades y la variabilidad identificadas en una etapa del desarrollo denominada *análisis del dominio.*

Una vez que se tienen identificados los diferentes componentes que permiten la creación de diferentes miembros de una familia de sistemas el siguiente paso es la selección y ensamblaje de los componentes que permiten la creación de un sistema que enfrente las necesidades actuales de un cliente determinado.

Es aquí donde entran los conceptos de programación generativa que no es más que la selección y ensamblaje automático de componentes en demanda [Czarnecki; 1999]. Más adelante se habla a detalle de los generadores de software.

1.3. JUSTIFICACIÓN.

Los conceptos de familias de sistemas y generación automática de software de que hemos estado hablado, pueden aplicarse a una gran variedad de dominios. Dentro del presente trabajo se aplican los conceptos de líneas de productos y programación generativa al dominio de las fresadoras de alta velocidad dentro de la industria manufacturera .

A continuación se da una introducción al dominio objeto de estudio para justificar la factibilidad de aplicar la programación generativa en la construcción de sistemas de predicción de tiempos de maquinado en fresadoras de alta velocidad, las cuales tienen una alta variabilidad debido a que cada fabricante les imprime características propias, y al mismo tiempo comparten puntos en común, debido a su funcionalidad.

1.3.1. La Industria Manufacturera.

Desde la revolución industrial los procesos de manufactura han evolucionado de ser totalmente manuales, y en muchos casos artesanales, a ser sistemas de producción (procesos que transforman recursos en bienes y servicios útiles) totalmente automatizados [Flores, 1985].

Las máquinas herramientas conceptualizadas como aquellas que "transforman materia prima a través de remoción de material para producir piezas útiles al hombre" (como las esmeriladoras, taladros, tornos, cepillos y fresadoras) también se han transformado [Kief; 1988].

En principio eran máquinas controladas a través de palancas, manivelas, trenes de engranes y levas. Actualmente, dichas máquinas incluyen un control numérico (CN) que las controla automáticamente, mediante información programada y codificada en formato simbólico. Tal información representa las dimensiones de la pieza a maquinar, la herramienta requerida, la velocidad del husillo, la velocidad de avance de la herramienta, etc.

Con el desarrollo computacional se posibilita un nuevo tipo de control: el control numérico por computadora (CNC). Actualmente existe software de Manufactura Asistida por Computadora (CAM) así como de Diseño Asistido por Computadora (CAD) que apoyan ampliamente el campo y que generan los programas de control numérico, mismos que definen la trayectoria de corte de la herramienta en una máquina herramienta.

Paralelo a esto, se dio el desarrollo de los controladores, de las máquinas herramienta y de las herramientas de corte, lo cual ha llevado al incremento en su uso en aplicaciones de maquinado de alto desempeño como es el caso de las fresadoras de alta velocidad.

1.3.2. Problemática Del Dominio De Aplicación (Fresadoras De Alta Velocidad.

Dentro de la industria manufacturera, la industria automotriz es una de las que se ha visto más ampliamente beneficiada por el desarrollo que se ha dado en la tecnología de las máquinas herramienta, especialmente lo que se refiere al uso de fresadoras de alta velocidad, lo cual le ha permitido enfrentar los retos de producción y estimular su crecimiento a nivel mundial.

Las compañías automotrices tanto de Estados Unidos como las japonesas se han introducido a los mercados europeos y de América Latina, donde han hecho grandes inversiones, lo cual a llevado a un crecimiento notable en los últimos años, tanto en su producción global promedio como en las exportaciones como se puede ver en la Tabla 1.1 [Eclac; 1998].

	Producción (Unidades)		Exportaciones (Unidades)	
País/Período	**1980-1991**	**1995-1997**	**1980-1991**	**1995-1997**
México	585,700	783,900	253,100	607,900
Brasil	699,800	1 478,400	137,500	235,500
Argentina	100,900	287,400	2,000	91,500

Tabla 1.1 Producción y Exportación de Autos de pasajeros en Algunos Países de América Latina[1]

México tuvo un crecimiento de aproximadamente 33% en su producción mientras que el crecimiento en Brasil y Argentina fue mayor de 111 y 184% respectivamente. En las Figuras 1.2 y 1.3 se muestra el crecimiento de las importaciones, exportaciones y ventas de unidades automotrices en México y Brasil respectivamente.

Este crecimiento se debe en gran medida a las estrategias competitivas implementadas. La industria automotriz japonesa por ejemplo, basa sus estrategias competitivas en la reducción de sus costos de producción y el incremento en la calidad por la vía de mejores prácticas organizacionales y de administración de sus tecnologías entre otras cosas [Bass; 1998]. Sin embargo, para llevar a cabo estas prácticas con la eficiencia que se requiere, el software de CAD/CAM disponible que sirve de apoyo a las diferentes tareas de manufactura, no enfrenta el desarrollo que se ha dado en los controladores y en las máquinas herramienta mismas.

[1] **Fuente:** CEPAL, Unidad de Inversiones y Estrategias Empresariales

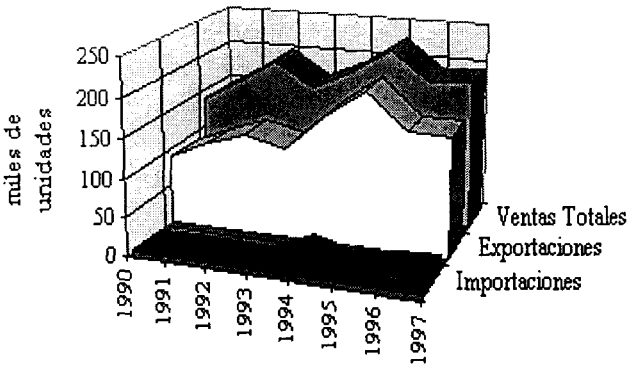


Figura 1.2 Exportaciones, Importaciones y Ventas de vehículos de pasajeros Ford en México, 1990-1997[2]

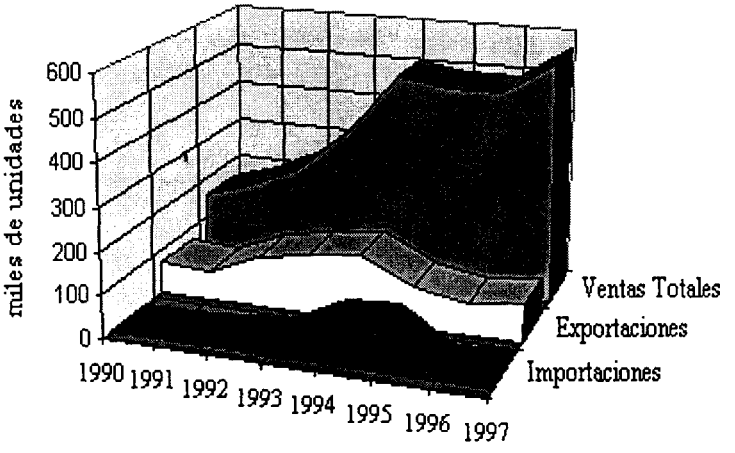


Figura 1.3. Importaciones, Exportaciones y Ventas de Vehículos de Pasajeros FIAT en Brasil, 1990-1997[3]

Entre las funciones que desempeñan los sistemas CAM , está la de calcular el tiempo de maquinado de una pieza, para lo cual toman la distancia total que recorre la herramienta de corte y la dividen sobre la razón de movimiento nominal (definido por el fabricante). Este enfoque sin embargo, no toma en cuenta las limitaciones físicas de las máquinas tales como el tiempo que tarda la máquina en llegar de un estado de reposo al valor nominal de movimiento especificado.

Al empezar a maquinar, una fresadora está en la posición inicial de reposo, denominada "home". A partir de aquí la velocidad empieza a incrementarse siguiendo la trayectoria de corte que viene especificada en un programa que contiene las instrucciones que la definen. La herramienta tarda en llegar del punto de reposo a la velocidad nominal.

[2] **Fuente**: (AMIA)
[3] **Fuente**: (ANFAVEA)

Una vez que ha alcanzado dicha velocidad se estabiliza siempre y cuando la trayectoria definida sea lineal.

Si por el contrario la trayectoria define curvas o cambios de dirección, la herramienta tiene que empezar a decrementar la velocidad desde una distancia que le permita llegar exactamente al punto donde cambia la dirección, cambiar la dirección de la trayectoria y empezar a acelerar hasta alcanzar de nuevo la velocidad nominal.

Este comportamiento se repite para cada cambio de dirección y estos tiempos que tarda la herramienta en alcanzar la velocidad nominal y los tiempos en que está desacelerando para un cambio, no se toman en cuenta en la manera que actualmente se hace el cálculo del tiempo de maquinado.

La diferencia resultante del tiempo de maquinado estimado con respecto a los tiempos reales de maquinado de una pieza es significativa, y se acentúa en condiciones de altas velocidades de avance agravando aún más el problema. De acuerdo a pruebas realizadas [Rodríguez; 1999] se encontraron diferencias de hasta un 6% entre el tiempo de maquinado real y el estimado en fresadoras de alta velocidad.

Entre las máquinas-herramientas en general y las fresadoras de alta velocidad en particular existen diferencias que hacen difícil estandarizar un modelo de predicción de tiempo de maquinado.

Por ejemplo, existen máquinas de dos, tres y hasta 5 ejes. Algunas máquinas son capaces de definir solamente trayectorias lineales y aproximan las curvas por definición de puntos como se explica más adelante en el capítulo IV cuando se habla de los diferentes tipos de interpolación. Por otra parte algunas máquinas definen curvas, parábolas y otros tipos de curvas más complejas. Algunas sólo soportan un tipo de interpolación mientras que otras soportan varios tipos simultáneamente.

Una manera de determinar el tiempo que tarda una fresadora en maquinar una pieza es precisamente maquinando la pieza. Sin embargo este enfoque resulta costoso porque implica materia prima, tiempo y recursos que pueden estarse aprovechando.

Existen trabajos que buscan establecer modelos matemáticos del comportamiento real de la máquinas fresadoras de alta velocidad en cuanto a la predicción del tiempo de maquinado de una pieza dado un programa de control numérico [Rodríguez; 1998]. Este enfoque elimina el costo de tener que maquinar la pieza.

Una estimación más aproximada de los tiempos de maquinado en fresadoras de alta velocidad ayudará para evaluar los beneficios económicos de aplicar herramientas de corte y máquinas herramienta de alto desempeño.

La importancia de una estimación de tiempos más próxima a la realidad, radica en que permitirá una mejor planeación de la producción, y por lo tanto el mejor aprovechamiento de la planta productiva instalada, presupuestos más exactos, entregas de

producto en los tiempos estimados y una mayor satisfacción del cliente lo cual ofrece mayores ventajas competitivas.

Teniendo información confiable respecto al tiempo estimado, no solamente se puede planificar mejor la ocupación de la máquina, evitando cuellos de botella, sino que los plazos de entrega de la producción se pueden predecir de modo más exacto y existe la posibilidad de intercalar pedidos urgentes.

1.4 PROPUESTA DE SOLUCIÓN.

Para enfrentar el problema de la estimación de tiempos de maquinado se presenta una herramienta computacional que permite la generación de aplicaciones de predicción de tiempo de maquinado para cualquier programa de control numérico, a partir de un modelo matemático que describe el funcionamiento de las fresadoras de alta velocidad, con la capacidad de adaptarse a las características de una fresadora específica a partir de un conjunto de datos de calibración de la máquina herramienta.

Para la realización de este trabajo se llevó a cabo un estudio del dominio de aplicación, que en este caso es el de las fresadoras de alta velocidad, y se definió una arquitectura en base a la cual se creó el prototipo de un generador de código de aplicaciones que permiten predecir el tiempo de maquinado de una pieza, teniendo el programa de control numérico que define la trayectoria de maquinado, el modelo matemático que representa el comportamiento de la herramienta de corte de la fresadora durante el proceso de maquinado de la pieza, y los datos de calibración correspondientes.

La arquitectura definida provee la flexibilidad suficiente para implementar la herramienta de predicción en diferentes tipos de fresadora, tomando en consideración que éstas tienen características propias y que esto las hace diferentes entre sí. Tales características se refieren al tipo de interpolación que usan, teniéndose máquinas que solamente soportan un tipo y otras que soportan varios tipos simultáneamente.

Del tipo de interpolación depende el modelo matemático desarrollado para estimar el tiempo de maquinado y, para un mismo interpolador pueden desarrollarse diferentes modelos que permitan comparar los resultados y elegir entre el mejor modelo.

La herramienta permite el uso de diferentes modelos aplicados a diferentes tipos de máquinas fresadoras.

1.5 METODOLOGÍA EMPLEADA.

Para el desarrollo tanto de la herramienta como del generador de sistemas de predicción de tiempo de máquinado en fresadoras de alta velocidad, se siguieron los pasos del proceso de desarrollo basado en arquitecturas que se explica en el capítulo II.

Durante la fase de **Creación del caso del negocio para el sistema** se buscó establecer el mercado objetivo del sistema así como las limitantes que se enfrentaron durante el proceso de desarrollo mismo.

En la segunda fase denominada **Análisis del dominio** se establecieron las características importantes del dominio de aplicación a través de la metodología de análisis de dominio propuesta por [Czarnecki; 1999] y explicada en el capítulo II, en la cual se emplea el análisis de características y la identificación de funcionalidades y componentes del sistema, para luego establecer la dependencia de usos entre las categorías de componentes, lo cual nos permite llegar a la siguiente etapa.

La siguiente etapa dentro de este proceso de desarrollo basado en arquitecturas es la **definición de la arquitectura.** Aquí se busca una arquitectura que permita manejar la variabilidad encontrada en la familia de sistemas de los predictores de tiempo para fresadoras de alta velociadad, a fin de poder automatizar el proceso de generación de nuevos miembros de la familia de sistemas.

Una vez definida, es necesario tener una **representación de la arquitectura.** Para ello se emplea el lenguaje UML el cual permite a través de diferentes símbolos que componen su vocabulario, mostrar las relaciones que existen entre los diferentes componentes encontrados durante la fase de análisis. Estas representaciones proporcionan diferentes puntos de vista de la misma arquitectura definida. En el capítulo III se explica la símbología del lenguaje además de otros aspectos del mismo.

Finalmente se **Implementa el sistema** predictor de tiempos de acuerdo a la arquitectura definida y se automatiza el proceso de creación de nuevos miembros de la familia a través de un generador de software que contempla las limitantes y características de la arquitectura.

1.6. ESTRUCTURA DE LA TESIS.

En el **capítulo uno** se habla de los problemas que se han presentado en el desarrollo de software de aplicación debido a que no se identifican necesidades potenciales ni aspectos comunes entre los sistemas. Se explica el enfoque de familias de productos como un nuevo paradigma que intenta ayudar a solucionar este problema y permitir una mayor productividad en los procesos de desarrollo.

Además, se da una breve introducción al dominio de aplicación que se estudia en el presente trabajo. Se exponen los cambios que ha sufrido la industria manufacturera y se describe la manera en que se realiza la predicción de tiempos de maquinado en fresadoras de alta velocidad enfatizando los puntos débiles de esa metodología y la variabilidad que se puede presentar al tratar de resolverlo aplicando un sólo modelo. Se establece la importancia que tiene en el ámbito de la industria manufacturera (especialmente el área automotriz), una predicción más exacta de los tiempos de maquinado. Y finalmente se propone una herramienta computacional que permita enfrentar las deficiencias que se

presentan en la estimación de tiempos de maquinado y que se ajuste a la variabilidad presentada.

En los capítulos del dos al cuatro se presenta el marco teórico el cual se divide debido a que el trabajo de tesis analiza el área de las máquinas herramientas como el dominio de aplicación, aunque el trabajo en sí está orientado al área del desarrollo de software.

En el **capítulo dos** se habla del estado del arte del área de las arquitecturas de software. Se establece la influencia de su uso en una organización de desarrollo y cómo a su vez es influenciada por el entorno. Se describen diferentes estilos arquitectónicos que actualmente son los más comunes y se proporciona una metodología de desarrollo basada en arquitecturas.

En el **capítulo tres** se describen algunas herramientas usadas en la implementación de diferentes tipos de sistemas tales como las técnicas de programación orientadas a objetos y los compiladores. También se habla de los lenguajes de apoyo para la representación de la arquitectura de un sistema como UML.

En el **capítulo cuatro** se presenta el marco teórico de lo que se establece como el dominio o área de análisis que sirve como referencia de aplicación. Se habla de lo que es el control numérico y el control numérico computarizado, y de los sistemas CAD/CAM como una evolución lógica de la automatización de los procesos de producción. Se habla de los tipos de control y los interpoladores como los elementos que definen las diferencias que existen entre las fresadoras de alta velocidad y se describen los elementos básicos del lenguaje de programación en que se generan los programas de CN y CNC que definen las trayectorias.

En el **capítulo cinco** se describe paso a paso el desarrollo de la metodología usada en el trabajo de tesis. Se establece la arquitectura del dominio de las fresadoras y se describe la aplicación generada en base a tal arquitectura.

En el **capítulo seis** se dan algunas conclusiones de este trabajo de tesis y se proponen áreas que podrían ser exploradas en trabajos futuros.

CAPÍTULO II.

ARQUITECTURAS Y SISTEMAS DE SOFTWARE.

La calidad de un sistema de software depende de varios factores, entre ellos el proceso de desarrollo que se siga. Al igual que en la construcción de un edificio, en la construcción de un sistema de software es importante contar con modelos sobre los que se pueda visualizar el resultado final a fin de poder hacer modificaciones que nos lleven a un producto final que vaya más allá de solamente cumplir con los requerimientos del cliente.

En el presente capítulo se habla de las arquitecturas de software que permiten crear diferentes modelos de un sistema. Se describen algunos de los diferentes estilos arquitectónicos que se han identificado dentro de los sistemas y se habla del proceso de desarrollo de software basado en arquitecturas.

2.1. ARQUITECTURA DE SOFTWARE.

Una arquitectura de software es un enfoque de la creación de un sistema, mucho más completo que el simple hecho de crear una aplicación funcional que enfrente un problema específico dado. Se trata de ir más allá del ciclo de vida del sistema mismo, creando una estructura que puede ser modificada de acuerdo a las necesidades cambiantes del medio ambiente que la influencia y que engloba aspectos inherentes de la organización que no pueden reflejarse en una aplicación, pero que pueden afectar el diseño.

Es necesario entonces, crear una estructura simple y luego extrapolarla de acuerdo al entorno técnico.

Una arquitectura tiene que ver con las estructuras de un sistema de software. Es un punto de vista abstracto que deja de lado detalles de implementación, algoritmos y representación de datos y se concentra en el comportamiento e interacción de componentes de "caja negra". Es el primer paso hacia el diseño de un sistema que tiene un conjunto de propiedades deseadas [Bass; 1998].

En [Bass; 1998] se da la siguiente definición de lo que es una arquitectura de software:

> "...es la estructura o estructuras del sistema, las cuales abarcan componentes de software, propiedades visibles desde afuera de esos componentes y las relaciones entre ellos." (1)

La mayor parte de la literatura concuerda que los elementos básicos de una arquitectura son los componentes y los conectores [Bass; 1998] [Perry; 1992] [Luckham1995], teniendo que un componente es una unidad de software que ejecuta alguna función en tiempo de ejecución y un conector es un mecanismo que media comunicación, coordinación o cooperación entre componentes [Perry; 1992].

También se consideran los controles, los datos y las limitantes. El control se da en la manera como se comunican e interactúan los componentes y la manipulación que se efectúa sobre los datos.

Un punto de control describe cómo el control pasa a través de los componentes y cómo los componentes trabajan juntos temporalmente

El control se comparte, se posiciona y se transfiere entre los componentes. Los datos se comunican a través de todo el sistema, por lo tanto el control y los datos interactúan a fin de lograr la correcta funcionalidad del sistema [Perry; 1992].

Existe la expectativa de que una arquitectura debe ser una plantilla que guíe el desarrollo de un sistema (o familia de sistemas). Como plantilla, una arquitectura además de los componentes, los controles y las interfases, define las limitantes de un sistema [Luckham1995], las cuales establecen las reglas que regulan la manera como esos componentes pueden interactuar unos con otros.

La arquitectura de software de un sistema es el primer producto que habilita las diferentes propiedades que los usuarios del mismo desean, y concilia los probables conflictos o contradicciones que pudieran existir entre dos propiedades deseadas. Un problema puede ser resuelto mediante diferentes estructuras o composiciones de componentes (Figura 2.1) dependiendo del método de diseño que se esté utilizando [Pressman; 1993].

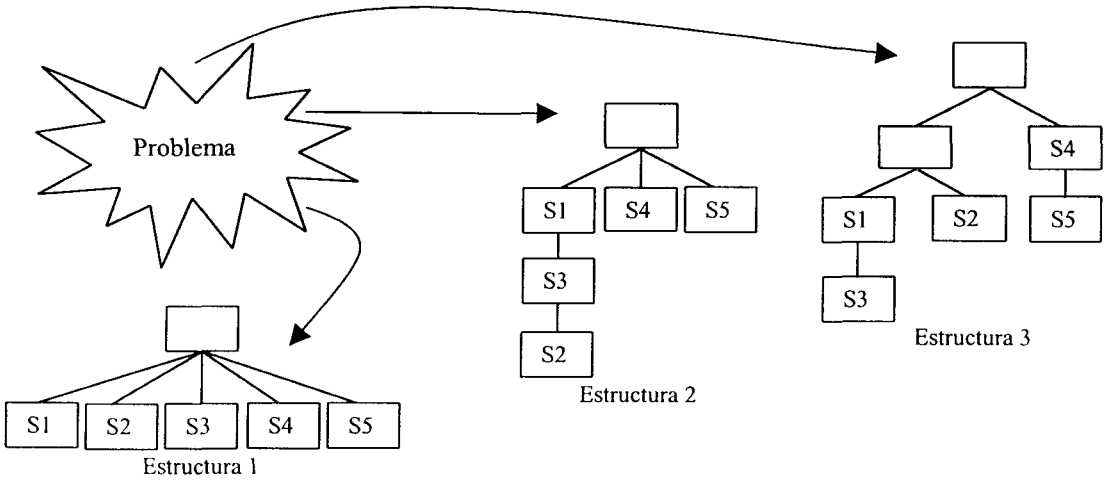


Figura 2.1 Estructuras diferentes para un mismo problema [Pressman; 1993].

La frase "visibles desde fuera" de la definición (1) significa que otros componentes pueden hacer supuestos de un componente tal como sus servicios prestados, características de desempeño, manejo de fallas, uso de recursos compartidos, etc.

Al crear una arquitectura de software se buscan algunos beneficios como [Perry; 1992]:

1) Definir una estructura que sirva como fundamento para satisfacer los requerimientos que los diferentes usuarios tienen del sistema.

2) Disponer de una base técnica que soporte el diseño y sirva como apoyo a las áreas administrativas en la estimación de costos y la administración de procesos.

3) Contar con arquitecturas que habiliten de manera efectiva el reuso de los componentes identificados dentro de un sistema.

4) Tener una arquitectura que sirva como fundamento para el análisis de consistencia y dependencia.

Para lograr estos beneficios se debe tomar en cuenta que una arquitectura influye en factores del ambiente técnico, social y de negocios de una organización y estos a su vez, influyen en la arquitectura (Figura 2.2). Esto se conoce como Ciclo de Arquitectura del Negocio (ABC por sus siglas en inglés) [Barroca; 1999][Bass; 1998].

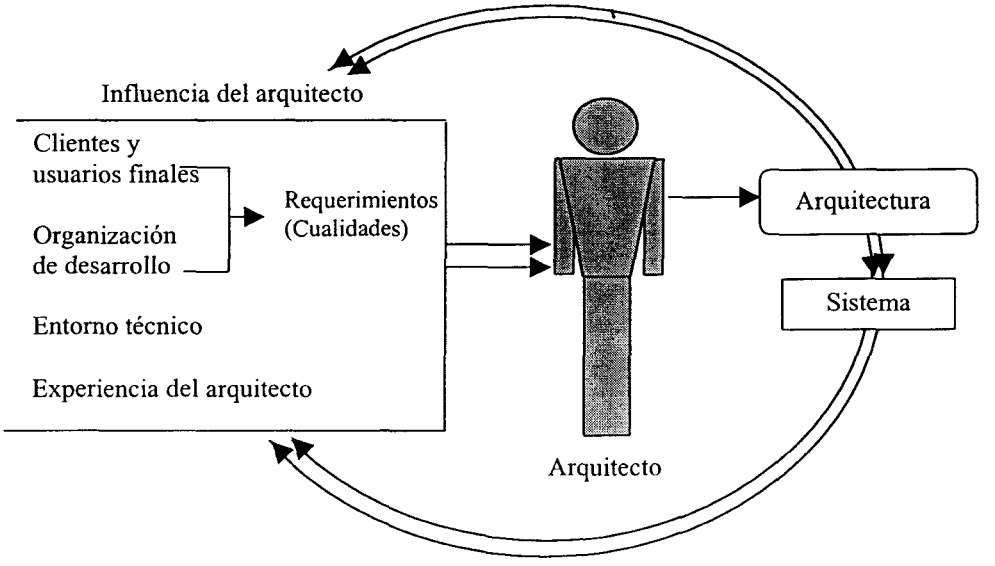


Figura 2.2 Ciclo de arquitectura del negocio.

El ciclo se da de la siguiente manera:

Clientes y usuarios finales: Los usuarios y clientes de un sistema son muchos y con expectativas diferentes sobre lo que será el sistema. Un mismo sistema puede tener diferentes versiones que se adapten a diferentes clientes. Tales expectativas y variabilidad deberán tomarse en cuenta y reflejarse en la arquitectura definida.

Por su parte, la arquitectura puede influir en los requerimientos del cliente porque le da la posibilidad de un sistema mejorado (basado en la arquitectura) de modo más confiable, a tiempo y menos costoso que si tuviera que realizar el sistema desde el principio.

Organización de desarrollo: La influencia que ejerce la organización de desarrollo sobre la arquitectura, depende de los intereses de la misma. La organización puede estar interesada en reuso por lo que la arquitectura llevará esa tendencia. O puede tener interés de crear una infraestructura a largo plazo por lo que verá al sistema como un medio de financiarse y crear la infraestructura deseada.

Por otro lado, una arquitectura prescribe las unidades de software que tienen que integrarse para formar el sistema. Esas unidades corresponden a asignaciones de trabajo las cuales forman las bases para la estructura del proyecto de desarrollo. Los equipos de trabajo se forman alrededor de esas unidades y se asignan presupuestos.

Entorno técnico: Para el desarrollo de la arquitectura, se pueden incluir las prácticas estándares de la industria, o las técnicas de ingeniería de software prevalecientes.

A su vez, la arquitecturas afectan esta variable posibilitando nuevas maneras de enfrentar un problema. Las primeras BD relacionales, generadores de compiladores y sistemas operativos de manejo de tablas afectaron en los 60 y principio de los 70. Las primeras hojas de cálculo y los sistemas de ventanas afectaron en los 80 . CORBA, Java, y el WWW pueden ser ejemplos de los 90.

Experiencia del arquitecto: Un arquitecto siempre se inclinará por aquello en lo que tiene más experiencia o se le facilita más, y a su vez el desarrollo de la arquitectura le proporciona una mayor experiencia al arquitecto.

2.1.1 Estilos arquitectónicos.

Un estilo arquitectónico define una familia de sistemas en términos de los patrones de organización estructural, es una abstracción de un conjunto de arquitecturas que tienen características formales comunes. Más específicamente, determina el vocabulario de componentes y conectores que pueden ser usados en instancias de ese estilo, junto con un conjunto de limitantes sobre cómo pueden ser combinados [Garlan; 1994] [Perry; 1992].

Un estilo arquitectónico es una descripción de tipos de componentes y un patrón de su control en tiempo de ejecución y/o transferencia de datos (por ejemplo el modelo cliente-servidor).

Un estilo arquitectónico define un conjunto de características claves y reglas para combinar esas características de modo que la integridad arquitectónica se conserva [Pressman; 1993]. Un estilo arquitectónico de software se determina por lo siguiente:

- Un conjunto de tipos de componentes (repositorios de datos, un proceso, un procedimiento) que desempeñan alguna función en tiempo de ejecución.
- Un diseño topológico de esos componentes indicando sus interrelaciones en tiempo de ejecución.
- Un conjunto de restricciones (por ej. a un repositorio de datos no se le permite cambiar los valores almacenados en él).
- Un conjunto de conectores (llamadas a subrutinas, a procedimientos remotos, flujos de datos, sockets) que median comunicación, coordinación o cooperación entre componentes.

A continuación se detallan algunos estilos arquitectónicos.

Arquitectura centrada en los datos

Son sistemas con acceso muy frecuente a los datos almacenados. Básicamente consiste de un repositorio de datos central al cual tienen acceso diferentes clientes ya sea para modificar los datos o sólo para consultarlos (Figura 2.3).

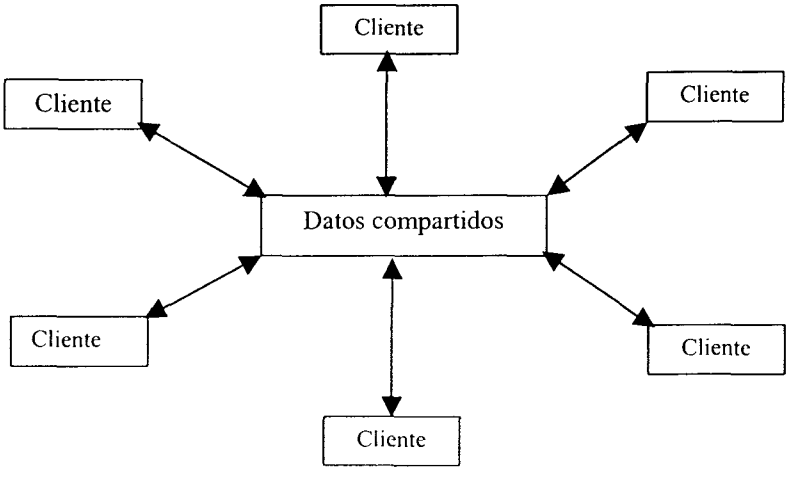


Figura 2.3 Arquitectura centrada en los datos.

Se distinguen dos tipos de repositorios: pasivo o central y activo o de pizarrón. El repositorio pasivo es un conjunto de datos (archivo) que son accesados por los diferentes clientes. En el repositorio activo o de pizarrón además de permitir el acceso, se notifica a los clientes que lo accesan cuándo cambian los datos.

Este estilo ofrece una solución a la integrabilidad (ej. Datawarehouses) a través del mecanismo de pizarrón.

Ventajas:

- Los clientes son relativamente independientes unos de otros.
- El almacén de datos es independiente de los clientes.
- Es escalable y pueden agregarse con facilidad nuevos clientes debido a que son independientes unos de otros.
- La funcionalidad de un clientes puede modificarse sin afectar a los demás.

Note que cuando los clientes son independientes tenemos un estilo cliente-servidor en el cual el servidor contiene el repositorio de datos y se encarga de controlar el acceso que cada cliente tiene a tales datos.

Arquitecturas de flujo de datos

Su propósito es el reuso y lograr sistemas altamente modificables. Se visualiza al sistema como una serie de transformaciones en piezas sucesivas de datos de entrada.

En este estilo existen dos categorías: batch secuencial y pipe-and-filter. En el primero los componentes son programas independientes y uno se ejecuta completamente antes de que inicie la ejecución del siguiente (Figura 2.4). Los datos se transmiten como un todo en los diferentes componentes.

El estilo pipe-and-filter involucra una transformación incremental de los datos a través de cada componente (estilo que utilizan en sistemas operativos UNIX).

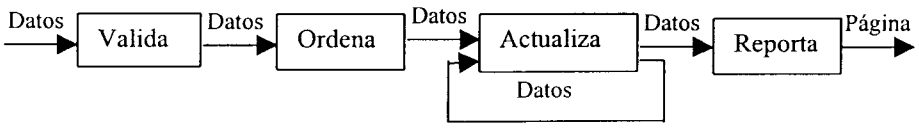


Figura 2.4 Arquitectura Batch Secuencial.

Cada componente lee un flujo de datos en su entrada, les aplica una transformación local y produce un flujo de datos en su salida, entregando una instancia completa de los resultados en un orden estándar. La salida empieza antes de que la entrada de datos se consuma completamente.

Los componentes se denominan filtros (filter) y, puesto que los conectores de este estilo sirven como conductos de los flujos transmitiendo las salidas desde un componente a otro, se denominan tuberías (pipes).

Entre sus características se destacan las siguientes:

a) Los filtros deben ser entidades independientes es decir, no deben compartir su estado con otros filtros.

b) Los filtros desconocen la identidad de los filtros del flujo superior e inferior. Tienen claro qué es lo que esperan recibir y lo que deben entregar pero no saben quién es el filtro que les entrega o que recibe sus flujos de datos.

c) La correctitud de las salidas de un filtro no debe depender del orden en que los filtros ejecutan el procesamiento.

Ventajas de pipe-and-filter:

- No hay interacciones de componentes complejos que administrar.
- Una función del sistema se compone a partir de funciones simples.
- Simplifica el mantenimiento del sistema y posibilita el reuso puesto que los filtros pueden tratarse como cajas negras (sólo se conocen la entrada y la salida).
- La estructura puede componerse jerárquicamente (Pueden agruparse filtros conectados por pipes y presentarse al mundo exterior como un filtro).
- Puesto que un filtro puede procesar su entrada de manera aislada del resto del sistema, un sistema pipe-and-filter fácilmente se puede hacer en paralelo o distribuido proveyendo oportunidades de mejorar el desempeño del sistema sin modificarlo.

Desventajas:

- Es difícil crear aplicaciones interactivas debido a la manera en que el problema es dividido (existe una mentalidad de agrupamiento o batch).

- Ordenar los filtros puede ser difícil. No hay manera en que los filtros cooperen interactivamente para resolver un problema.

- El desempeño en tales sistemas con frecuencia es pobre, por razones que surgen de la manera aislada de su funcionalidad, lo que lo hace tan modificable.

- Típicamente forzan a una representación de datos muy simple (tal como flujos ASCII). Si la entrada necesita transformase en tokens hay un costo operativo de parsing/unparsing.

- Si un filtro no puede producir su salida hasta que toda la entrada se ha recibido, requerirá un buffer de tamaño ilimitado.

- Cada filtro opera como un proceso aparte o una llamada a procedimiento aparte incurriendo en algún costo operativo cada vez que es llamado.

Arquitecturas de máquina virtual

Su objetivo es lograr sistemas portables es decir, sistemas que requieren poco esfuerzo para transferirse de un hardware y/o entorno de sistemas de software a otro [Pressman; 1993]. Simulan alguna funcionalidad que no es parte del hardware y/o software en el cual se implementan (Figura 2.5). Algunos ejemplos son: intérpretes, sistemas basados en reglas, shells sintácticos y procesadores de lenguajes de comandos.

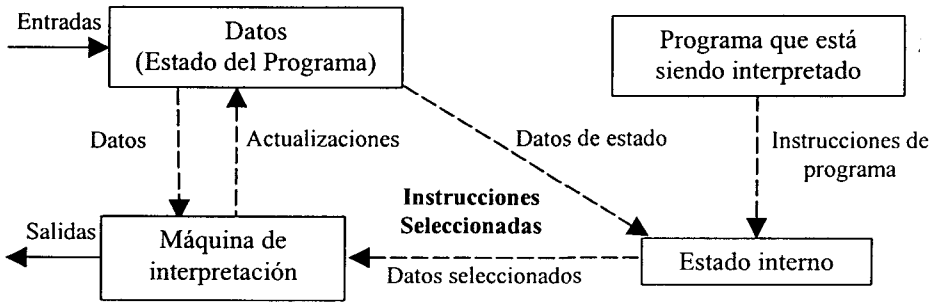


Figura 2.5 Arquitectura de Máquina Virtual.

La máquina de interpretación selecciona una instrucción del programa que está siendo interpretado, actualiza su estado interno (valores de los registros o el comando actual que está siendo ejecutado), y basado en la instrucción, potencialmente actualiza los datos del programa.

Ejecutar un programa usando un intérprete agrega flexibilidad a través de la capacidad de interrumpir y filtrar el programa e introducir modificaciones en tiempo de

ejecución, pero hay un costo de desempeño debido a la computación adicional involucrada en la ejecución.

Arquitecturas de llamada y retorno

Su objetivo es lograr sistemas modificables y escalables. Dentro de este estilo podemos encontrar la siguiente clasificación:

(a) **Arquitectura de programa principal y subrutinas**: Descompone un programa en pequeñas piezas de manera jerárquica (Figura 2.6). El control pasa de padres a hijos.

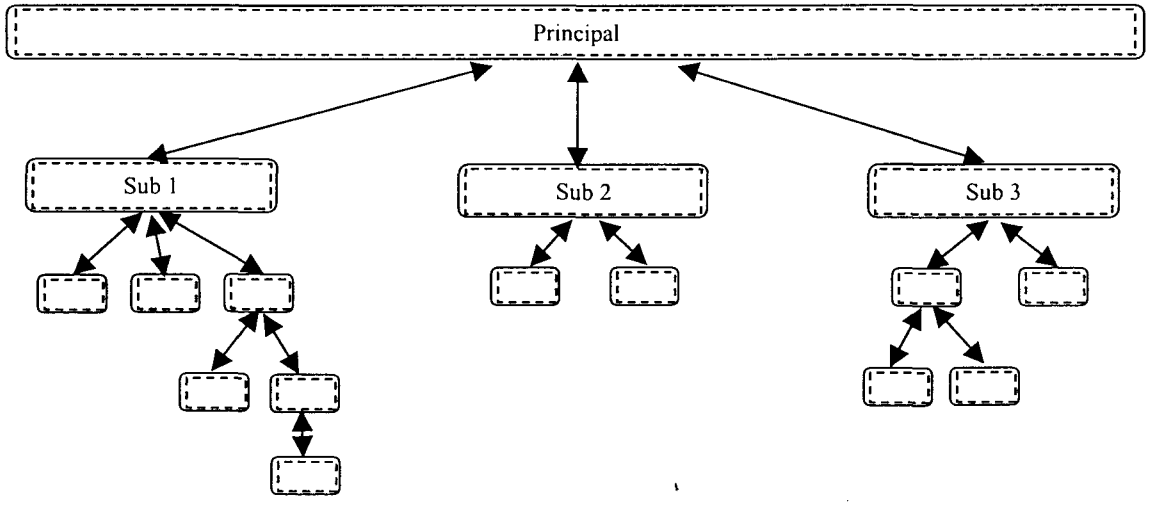


Figura 2.6 Arquitectura de programa principal y subrutinas.

(b) **Sistemas de llamada a procedimiento remoto**: Es un estilo como el anterior pero descompone el sistema en partes que se alojan en computadoras conectadas en red. Mejora el desempeño distribuyendo la computación y tomando ventaja de múltiples procesadores.

(c) **Sistemas orientados a objetos o de tipos de datos abstractos**: Encapsula los datos y su forma de acceso y manipulación escondiendo esto del exterior (Figura 2.7). El acceso a objetos se hace a través de operaciones llamadas *métodos,* las cuales son formas restringidas de llamadas a procedimiento. Esta encapsulación promueve el reuso y la modificación.

Cada objeto es responsable de preservar la integridad de su representación, misma que está escondida a los otros objetos. Esto permite cambiar la implementación sin afectar a los clientes del objeto.

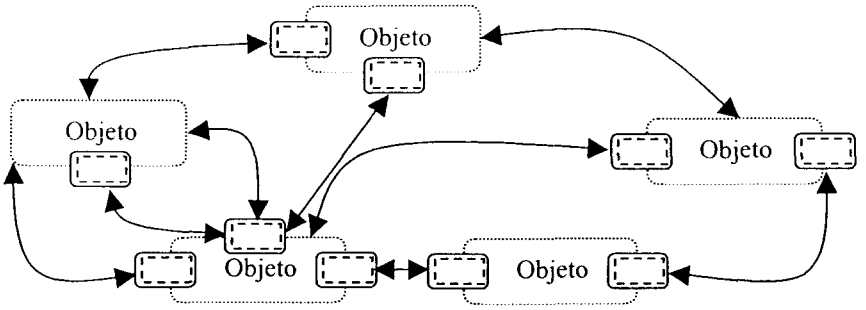


Figura 2.7 Estilo Orientado a Objetos.

Entre las desventajas que se presentan en este estilo es que un objeto debe conocer la identidad de los objetos con los que interactúa, de modo que cuando la identidad de un objeto cambia, es necesario modificar todos los demás objetos que lo invocan explícitamente.

Arquitectura de Invocación Implícita, Basada en Eventos.

Es un sistema que funciona diferente a aquellos en que se invoca una rutina dentro de un componente específico. En este estilo, los componentes anuncian uno o mas eventos. Los demás componentes del sistema pueden mostrar interés en un evento asociando un procedimiento con el evento. Cuando un evento se anuncia, el sistema mismo invoca todos los procedimientos que han sido registrados para tal evento. Por lo tanto, un anuncio de un evento "implícitamente" causa la invocación de procedimientos en otros módulos [Garlan; 1994].

Los componentes en este estilo son módulos cuyas interfases proveen tanto una colección de procedimientos como un conjunto de eventos. Los procedimientos pueden ser llamados de una manera convencional pero además, un componente puede registrar algunos de sus procedimientos con eventos del sistema de tal modo que esos procedimientos sean invocados cuando se anuncien esos eventos al tiempo de ejecución.

Los conectores incluyen llamadas tradicionales a procedimientos así como vínculos entre anuncios de eventos y llamadas de procedimiento.

Una característica de este estilo es que los anuncios de eventos desconocen cuales componentes se afectarán. Por lo tanto, los componentes no pueden asumir el orden del procesamiento o incluso qué procesamiento ocurrirá como resultado de sus eventos.

Este estilo provee un fuerte soporte para el reuso ya que se puede introducir cualquier componente simplemente registrándolo para los eventos de ese sistema. Además, la

invocación implícita favorece la evolución del sistema. Los componentes pueden ser sustituidos por otros sin afectar las interfases de otros componentes en el sistema.

Entre sus desventajas es que los componentes renuncian al control sobre la computación efectuada por el sistema. Cuando un componente anuncia un evento, no tiene idea qué otros componentes le responderán, ni el orden en que serán invocados y mucho menos cuándo ya terminaron.

Otro problema es que algunas veces se pasan datos con el evento pero en otras ocasiones los sistemas basados en eventos tienen que depender de un repositorio compartido para la interacción.

Arquitectura de Sistemas por Capas.

Los componentes se asignan a una capa de manera jerárquica para controlar la interacción entre componentes (Figura 2.8). Cada nivel sólo se comunica con las capas inmediatas superior e inferior.

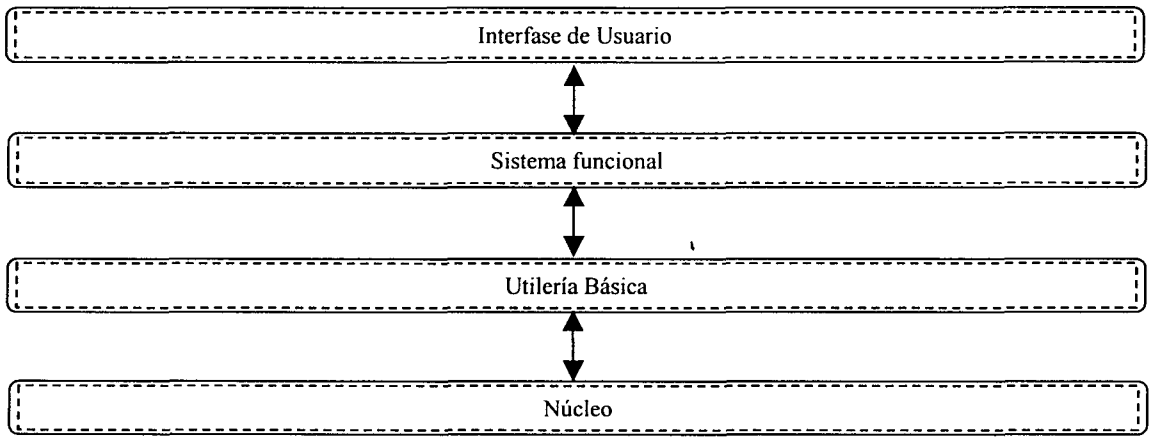


Figura 2.8 Estilo por Capas.

Los conectores están definidos por los protocolos que determinan cómo interactúan las capas .

Entre las limitantes topológicas se incluye que las interacciones sólo se dan hacia y desde las capas adyacentes.

Ventajas:

a) Soporta diseño basado en niveles de abstracción incremental lo que permite dividir un problema complejo en una secuencia de pasos incrementales.

b) Soporta mejoras. Los cambios en la función de una capa afecta cuando mucho a otras dos capas.

c) Soporta reuso. Se pueden tener diferentes implementaciones de la misma capa las cuales pueden ser usadas de manera intercambiable siempre y cuando provean de la misma interfase a las capas adyacentes.

Desventajas:

a) No todos los sistemas son fácilmente estructurados en forma de capas.

b) Las consideraciones de ejecución pueden requerir mayor acoplamiento entre funciones de alto nivel y sus implementaciones de bajo nivel.

c) Puede ser muy difícil encontrar el nivel correcto de abstracción.

Arquitecturas de componentes independientes.

Consiste de varios componentes u objetos independientes que se comunican a través de mensajes. El mensaje puede ser transmitido a un participante específico o se pasa entre participantes no nombrados, como en el caso de sistemas de eventos que usan el paradigma publicar / suscribir.

En los sistemas de eventos el control es parte del modelo. Los componentes individuales anuncian los datos que quieren compartir (publican) con su ambiente –un conjunto de componentes no nombrados. Esos componentes pueden registrar un interés en esta clase de datos (suscriben). Cuando los datos aparecen, los componentes se invocan y reciben los datos.

Este paradigma desacopla la implementación de los componentes de saber los nombres y ubicación de los otros componentes. Esto facilita la integración de componentes.

Arquitecturas Heterogéneas.

Todos los estilos mencionados anteriormente se refieren a arquitecturas "puras". Sin embargo, en la realidad la mayoría de los sistemas se forman por alguna combinación de estilos. Así por ejemplo, podemos tener un sistema cuya arquitectura por capas contiene en alguna capa un estilo de arquitectura de llamada y retorno o bien, otra arquitectura donde un componente podría accesar a un repositorio por una parte y por otra interactuar a través de invocación implícita con otros componentes del sistema.

2.2. ARQUITECTURAS DE SOFTWARE DE DOMINIO ESPECÍFICO (DSSA).

Se toman las arquitecturas de software de dominio específico en otro apartado debido a que éstas tienen que ver ampliamente con líneas de productos y con generadores de software.

El término DSSA va mucho más allá de una simple arquitectura como las que hemos estado analizando hasta este momento. Se aplica no a tratar de resolver un problema, sino a un dominio de problemas dados; abstrae una arquitectura que representa a esos problemas en su conjunto [Bass; 1998].

Esa arquitectura que engloba al dominio puede reusarse a través de una serie de múltiples sistemas similares. Así mismo los componentes definidos con la arquitectura y que se encuentran en la librería de componentes pueden reusarse a fin de crear sistemas que enfrenten las necesidades de la organización.

El reuso a través de DSSA provee de beneficios tales como reducción de costos de construcción y de tiempo para sacar un producto al mercado [Bass; 1998].

Esta serie de sistemas similares nos lleva a pensar en ellos como en una familia de productos que se aplica a un dominio de problemas específicos.

2.2.1. Familias y Líneas de Productos.

Una familia de productos es un conjunto de ítems que tienen aspectos comunes y variabilidad predecible y una línea de productos es una familia de productos diseñados para tomar ventaja de los aspectos comunes y la variabilidad predecible [Weiss; 1999].

En muchos casos cada miembro de la familia es un producto completo, tal como una familia de automóviles en la cual cada automóvil es un miembro de la familia y es un producto completo en sí mismo. En otros casos cada miembro es parte de un producto y se pretende que quepa en una gran familia, tal como las llantas de un automóvil.

Una definición más completa la encontramos en [Bass; 1998] quien dice que una línea de productos

> "... es una colección de sistemas que comparten un conjunto administrado de características construidas a partir de un conjunto de bienes básicos de software. Esos bienes incluyen una arquitectura base y un conjunto de componentes comunes y quizá ajustables a diferentes necesidades".

Y [Barroca; 1999] expone que una arquitectura de líneas de productos establece una estructura de software que sirve como base para todos los productos, e identifica y crea un

repositorio de componentes que se combinan precisamente para crear tales productos o sistemas de acuerdo a la funcionalidad que de ellos se espera obtener.

La producción de software orientada a familias se basa en algunos supuestos [Weiss; 1999]:

- La mayoría de los desarrollos consisten de crear variaciones sobre los sistemas de software existentes. Usualmente cada sistema tiene más cosas en común que diferencias con respecto a otras variaciones. Es posible evitar volver a desarrollar los aspectos comunes del software.

- Es posible predecir los cambios que probablemente serán necesarios para un sistema dentro de su ciclo de vida. Por lo tanto podemos predecir la familia de miembros que serán necesarios.

- El software y sus desarrolladores pueden organizarse de modo que un cambio de algún tipo predecible puede hacerse independientemente de cambios de otros tipos de modo que hacer esos cambios requiera modificar sólo algunos cuantos módulos del sistema. Se puede tomar ventaja de la predictibilidad.

El proceso de desarrollo de software basado en familias se divide en dos partes: la ingeniería de dominio y la ingeniería de aplicación (Figura 2.9) [Weiss; 1999].

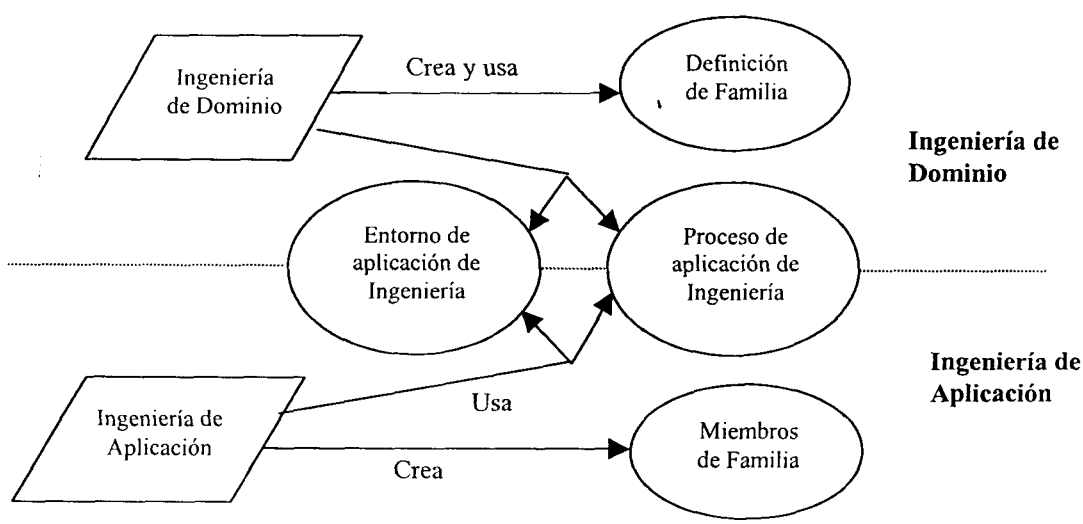


Figura 2.9. Proceso de Desarrollo de Software Basado en Familias.

La ingeniería de dominio define una familia y crea las facilidades de producción para la creación de sus miembros. La ingeniería de aplicación usa las facilidades de producción creadas por la ingeniería de dominio para crear nuevos miembros de la familia.

El resultado de los procesos de desarrollos basados en familias es hacer posible la creación de aplicaciones de software de gran calidad y mucho más rápido que con los procesos actuales.

En resumen, se identifica un conjunto de programas que constituyen la familia, a través de definir un conjunto de aspectos comunes y de variabilidades que determinar qué es y qué no es la familia.

A continuación se presentan algunos de los beneficios que pueden obtenerse al construir sistemas como líneas de productos basadas en componentes [Bass; 1998]:

- Producción más eficiente de nuevos miembros de la familia.
- Capacidad de soportar una gran variabilidad en los miembros de la familia.
- Capacidad de tomar ventaja más rápidamente de nuevos productos y nuevas tecnologías.
- Ventaja competitiva en dominios de aplicación
- Reducción significativa del tiempo que tarda un producto en salir al mercado porque el inventario de componentes está listo para ser usado.
- Mayor productividad de los empleados con énfasis no en el código sino en el reuso y la integración.
- Permite a los desarrolladores especializarse en áreas de aplicación de la organización.
- Sistemas más flexibles y evolutivos debido a que un componente puede ser intercambiado por otro más eficiente.

Para crear una familia de productos primero se identifican colecciones de programas que pueden ser considerados familias. A continuación se diseña un proceso para producir miembros de la familia concurrentemente con crear un diseño común para miembros de la familia. La idea es facilitar la producción de cualquier miembro de la familia a través del seguimiento del proceso, el cual incluye pasos para aplicar el diseño común.

Para cada familia se debe crear una manera de modelar los miembros de la familia ya que ningún tipo de modelo es apropiado para todas las familias. Para cada familia se usa un tipo de modelo que está de acuerdo con las características de esa familia. El modelo puede ser expresado en forma de un lenguaje de modelación de la aplicación (AML) y puede incluir:

- Lenguajes de especificación textual.
- Representaciones gráficas de árboles de decisión.
- Tablas.
- Diagramas de flujo de datos.

Puesto que las familia evolucionan conforme cambian las necesidades de los clientes, conforme surgen nuevas ideas para productos y conforme emergen nuevas tecnologías, la ingeniería de dominio debe ser un proceso de interacción continua.

2.2.2. Relación de las DSSA y los Sistemas Generadores de Software.

La idea principal que subyace en la definición de una DSSA es la creación de una línea de productos. Es decir, varios sistemas a partir de la arquitectura definida para el dominio y la combinación de componentes de software relacionados con la arquitectura.

Un componente puede ser usado en varios productos individuales lo cual significa reuso tanto de código como de diseño, con el beneficio adicional de ahorro en tiempo [Bass; 1998].

La DSSA no sólo provee una estructura para componentes de software reusables, sino que también organiza la lógica de diseño y la adaptabilidad de las estructuras [Batory; 1995].

La premisa es que muchos de los problemas del dominio en estudio están bien entendidos. Un análisis de dominio puede ser usado para identificar componentes y limitantes inherentes al dominio. Un producto del análisis de dominio, llamado una "arquitectura de referencia" es un bosquejo para generadores de sistemas de software para el dominio [Batory; 1995].

Los generadores de software son sistemas de ensamblaje de sistemas de alto desempeño que se adecuan a las necesidades específicas de la organización a la que sirven es decir, son sistemas personalizados para grandes familias de aplicaciones [Batory; 1997].

Para poder crear un generador de aplicaciones, es indispensable contar con un conjunto de componentes bien definidos que puedan combinarse de varias maneras para producir una aplicación de acuerdo a cierta arquitectura definida [Bass; 1998].

El programador establece lo que desea en términos abstractos y el generador produce el sistema o componente deseado. Esto significa *Programación Generativa* [Czarnecki; 1999].

De acuerdo a [Czarnecki; 1999] el análisis de dominio para la creación de una familia de sistemas, involucra el análisis de características el cual permite descubrir los puntos de diferencia y de igualdad dentro del dominio.

El resultado del análisis de características puede ser documentado usando diagramas de características (Figura 2.10)

La raíz del diagrama representa el concepto pricipal que en este caso es un carro. Los nodos restantes son las características que puede tener ese elemento principal.

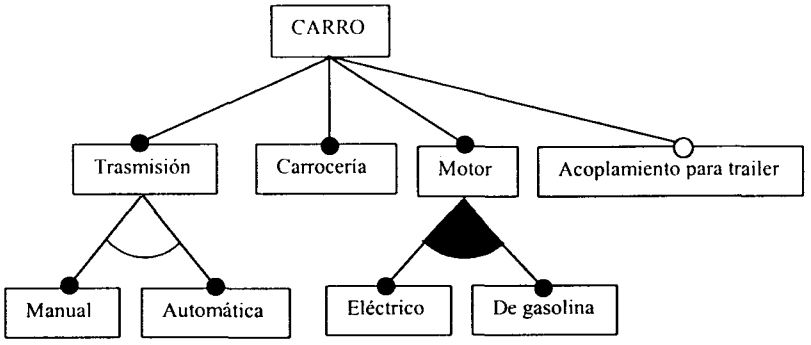


Figura 2.10 Diagrama de Características de un Carro.

Este tipo de diagramas contienen 4 tipos de características:

- Características principales: Se representan por líneas terminadas con un círculo relleno. Son las características que tienen que ir siempre en el sistema.
- Características opcionales: Se representan con líneas terminadas con un círculo vacío.
- Características alternativas: Se representan por líneas unidas en un extremo por un arco.
- Características OR: Se representan por líneas unidas en un extremo por un arco relleno.

Este tipo de diagrama no expresa las limitantes que se tienen en cuanto a las combinaciones de los elementos, por lo tanto será necesario poner las limitantes por separado.

Después del diagrama de características, hay que continuar con la definición de la arquitectura para la línea de productos.

Generadores GenVoca.

El modelo GenVoca está adaptado para generación de sistemas de software. Depende de un estilo particular de sistemas de software, organizados jerárquicamente en términos de

conjuntos estandarizados de capas parametrizadas y de interfases compatibles y reusables, llamadas componentes. Se pueden definir grandes familias de sistemas de software usando componentes y reglas para su composición [Batory; 1995].

El modelo GenVoca depende de la descomposición jerárquica de los sistemas del dominio en capas que importan y exportan interfases estandarizadas, y es independiente del dominio en cuanto a la definición de familias escalables de sistemas jerárquicos, como composiciones de componentes reusables.

El uso de interfases estandarizadas es clave para la escalabilidad, reusabilidad y componibilidad de los componentes.

La característica que distingue a GenVoca son los reinos (o librerías) de componentes simétricos de interfase compatible que pueden, con relativa facilidad, intercambiarse unos por otros, además de las ecuaciones tipo.

Un componente es la implementación de un conjunto de clases, sus objetos y funciones que trabajan cooperativamente para implementar un sistema. A continuación se muestran los reinos S y T con tres componentes y W con 4 componentes.

$$S=\{a, b, c\}$$

$$T=\{d[S], e[S], f[S]\} \qquad (1)$$

$$W=\{n[W], m[W], p, q[T,S]\}$$

Un componente tiene un parámetro (reino) para cada interfase que importa. Es decir que cada componente de T por ejemplo, exporta la interfase de T e importa la interfase de S.

Si un componente exporta la misma interfase que importa se dice que es *simétrico* y pueden hacerse composiciones en cualquier manera arbitraria. Por ejemplo en el reino W de (1), los componentes n y m son simétricos, mientras que p y q no lo son. Por lo tanto las composiciones n[m[p]], m[n[p]], n[n[p]] y m[m[p]] son válidas.

Una ecuación tipo es una composición de componentes que tiene un nombre y que representa un sistema, por ejemplo:

$$Sistema1 = d[b]$$

$$Sistema2 = f[a]$$

Ambos sistemas son ecuaciones de tipo T porque su componente exterior es de tipo T. Por lo tanto, ambos sistemas son implementaciones intercambiables de la interfase de T.

El diseño de una arquitectura GenVoca requiere de los siguientes pasos [Czarnecki; 1999]:

- Identificar las principales funcionalidades en el diagrama de características a partir del Análisis de Dominio.
- Enumerar las categorías de componentes y los componentes por categoría.
- Identificar las dependencias de "usos" entre las categorías de los componentes
- Ordenar las categorías en un arquitectura en capas.

Paradigma de la fábrica de software.

En vez de moverse de los requerimientos a un sistema de software para estimación de tiempos de fresado de alta velocidad (un proceso llamado *diseño sin reuso*), que es el tema del presente trabajo de tesis, se reconoce que los sistemas tales pertenecen a una familia o dominio de productos de software similares. Analizando el dominio es posible definir librerías de componentes primitivos y una fábrica para ensamblar esos componentes en sistemas objetivo. El bosquejo para las librerías y la fábrica se denomina *arquitectura de referencias*. Usar los requerimientos de diseño y la fábrica para producir sistemas objetivo se denomina "diseño con reuso" (Figura 2.11) [Batory; 1995].

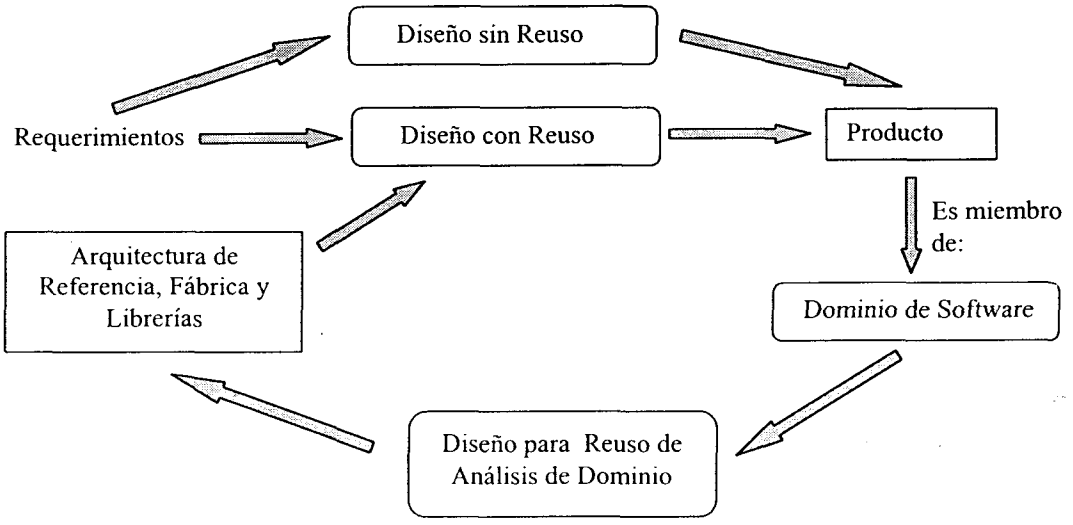


Figura 2.11 Un paradigma de Fábrica de Software.

2.3. Pasos del Proceso de Desarrollo Basado en Arquitectura.

Un proceso de desarrollo de software son los pasos que se siguen en la creación de un sistema de software y se conoce como *Ciclo de Vida*. Existen varios modelos del ciclo de vida. En la figura 2.12 se muestra el modelo en espiral [Pressman; 1993] en el cual se pueden ver los pasos básicos en la construcción de un sistema.

El modelo define cuatro actividades principales, representadas por los cuatro cuadrantes de la figura:

Planificación: Esta actividad involucra la determinación de objetivos, alternativas y restricciones del sistema.

Análisis de riesgo: En esta actividad se analizan las alternativas y se identifican y resuelven los riesgos inherentes al ciclo de vida del sistema.

Ingeniería: Esta actividad consiste en el desarrollo del producto en versiones sucesivas cada vez más completas.

Evaluación del cliente: Esta es una actividad de valoración de los resultados por parte del cliente.

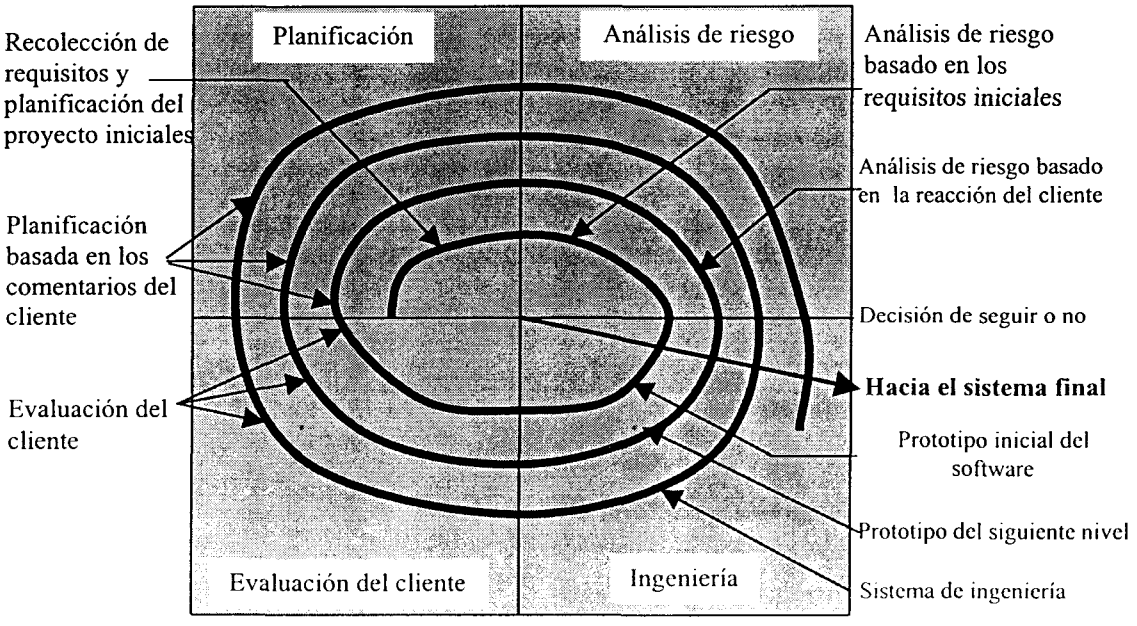


Figura 2.12 Modelo de ciclo de vida en espiral.

El modelo inicia en el centro siguiendo la línea de espiral hacia el exterior. Durante la primera vuelta se definen los objetivos, las alternativas y las restricciones, y se analizan e identifican los riesgos. Si el análisis de riesgo indica que hay una incertidumbre en los requisitos, se puede usar la creación de prototipos en el cuadrante de ingeniería para dar asistencia tanto al encargado del desarrollo como al cliente.

El cliente evalúa el trabajo de ingeniería y sugiere modificaciones que llevan a la siguiente fase de planificación y de análisis de riesgo. Esto lleva a la construcción de versiones sucesivas de software cada vez más completas.

Dentro de este modelo en espiral, podemos insertar a la arquitectura como una parte importante del proceso de diseño.

Los pasos en un proceso basado en arquitecturas serían los siguientes[Bass; 1998]:

Crear el caso del negocio para el sistema. En esta parte del proceso se evalúan aspectos acerca de la necesidad que tiene el mercado del sistema. Toma en cuenta factores como el costo del sistema, el mercado objetivo, el tiempo estimado para mercadear, interfase del sistema con otros sistemas, limitaciones del sistema.

Es importante para los arquitectos conocer y entender las clases de limitantes de un proyecto tan pronto como sea posible.

Entender los requerimientos. Hay técnicas que ayudan para esto. Por ejemplo el análisis orientado a objetos usa escenarios o "casos de uso". Los sistemas de seguridad crítica usan enfoques más rigurosos como máquinas de estado finito o lenguajes de especificación formal.

Otra técnica es el *Modelo de dominio.* El producto del análisis de dominio identifica las características comunes y las instancias diferentes de sistemas similares.

Cuando el sistema involucra un generador de software la modelación tiene objetivos muy claros desde el inicio del proceso de modelado del sistema [Batory; 1995]:

1) Identificar los componentes primitivos reusables del software del dominio.

2) Explicar cómo los componentes se unen para formar sistemas y subsistemas escalables.

3) Explicar las variaciones del software del dominio como combinaciones diferentes de componentes.

4) Delinear las características de un generador de sistemas del dominio que sea plausible.

Este último objetivo refleja el hecho obvio de que cualquier arquitectura de referencia es una teoría que postula cómo generar sistemas de software del dominio objetivo. Hasta que la teoría es validada a través de experimentación extensiva e implementación, ésta es sospechosa.

Y por último podemos mencionar la creación de prototipos que permite al usuario final visualizar el proyecto y definir mejor sus requerimientos.

Crear o seleccionar la arquitectura. En cualquier proceso de desarrollo habrá múltiples estilos arquitectónicos que pueden representar al sistema. De entre estos estilos se debe decidir por uno tomando en cuenta que las características encontradas en el análisis efectuado tengan concordancia con las del diseño en sí.

Aún no hemos llegado al estado en que se tiene un conjunto estándar de elementos de diseño. Cada sistema es en esencia una nueva arquitectura, un nuevo estilo arquitectónico [Perry; 1992].

Sin embargo dos cosas son importantes al momento de decidir la arquitectura de software que se va a utilizar: la evolución y la creación de sistemas que verdaderamente enfrenten las necesidades de los usuarios. Los sistemas evolucionan y se adaptan a nuevos usos. La evolución lleva a fragilidad de los sistemas debido a la erosión arquitectónica y al movimiento a la deriva de la arquitectura.

La erosión arquitectónica se da debido a las violaciones a la arquitectura. El rumbo a la deriva es debido a la insensibilidad respecto a la arquitectura. Un cambio que no toma en cuenta la arquitectura y se realiza de manera incongruente a las limitaciones impuestas por ésta, deteriora la estructura [Perry; 1992].

Representar y comunicar la arquitectura. Todas las personas que tienen que ver con el sistema deben entender de manera clara y sin ambigüedades la arquitectura presentada. Los desarrolladores deben entender las asignaciones de trabajo que la arquitectura requiere, los probadores deben entender la estructura de la tarea que se les impone, los administradores deben entender las implicaciones de agenda de actividades que sugiere, etc. Las arquitecturas pueden ser representadas o especificadas usando un lenguaje de descripción de arquitectura (ADL) que son la nueva tendencia tecnológica.

Analizar o evaluar la arquitectura. Principalmente hay que evaluar la mantenibilidad y la capacidad de soportar cambios. Los ADLs proveen de capacidades analíticas muy valiosas pero tienden a concentrarse en las propiedades de tiempo de ejecución del sistema.

Implementar el sistema basado en la arquitectura. La arquitectura define un sistema de una manera abstracta y permite modificaciones más controladas del mismo. Una implementación basada en la arquitectura tomará ventaja de ésta en el sentido de que permitirá un mantenimiento más fácil y una confiabilidad mayor del sistema.

Asegurarse que la implementación está de acuerdo a la arquitectura. Sobre todo cuando se presentan cambios es importante mantener consistencia entre la implementación y la arquitectura.

Todas estas actividades tienen una relación de retroalimentación unas con otras.

Como puede observarse , existe una relación entre el ciclo de vida en espiral y el proceso basado en arquitecturas en cuanto a sus actividades. Haciendo una analogía, podemos decir que la creación del caso del negocio corresponde al cuadrante de planificación, entender los requerimientos cae dentro del cuadrante de ingeniería y el resto de las actividades del proceso basado en arquitecturas tiene aspectos que bien pueden caer dentro de cualquiera de los cuadrantes del modelo en espiral. Sin embargo, mientras que el proceso de desarrollo en espiral se enfoca al sistema computacional en sí, el enfoque arquitectónico toma en cuenta el entorno de aplicación del sistema.

Resumen

Una arquitectura de software es una o varias estructuras de un sistema cuyos elementos básicos son los componentes y los conectores, teniendo que un componente es una unidad de software que ejecuta alguna función en tiempo de ejecución y un conector es un mecanismo que media comunicación, coordinación o cooperación entre componentes.

Una arquitectura de software pretende servir como base para satisfacer los diferentes requerimientos de un sistema, además de soportar el diseño y apoyar otras áreas como la administrativa en lo referente a estimación de costos y administración de procesos. Una arquitectura influye en factores del ambiente técnico, social y de negocios de una organización y estos a su vez, influyen en la arquitectura. A esto se le conoce como el Ciclo de Arquitectura del Negocio (ABC).

Un estilo arquitectónico define una familia de sistemas y determina el vocabulario de componentes y conectores que pueden ser usados en instancias de ese estilo, junto con un conjunto de limitantes sobre cómo pueden ser combinados para formar un sistema.

Entre los estilos arquitectónicos se pueden mencionar los siguientes: arquitectura centrada en los datos, arquitectura de flujo de datos, de máquina virtual, de llamada y retorno la cual se subdivide en arquitectura de llamada principal y subrutinas, sistemas de llamada a procedimientos remotos y sistemas orientados a objetos. También están los estilos arquitectónicos de invocación implícita basada en eventos, por capas, de componentes independientes y arquitecturas de software de dominio específico (DSSA)

Una DSSA está relacionada con líneas de productos y con generadores de software. Se centra en un dominio de problemas dados y abstrae una arquitectura que representa a esos problemas en su conjunto. Esto nos lleva al reuso tanto de la arquitectura como de los

componentes a lo largo del desarrollo de diferentes sistemas que caen dentro del dominio de problemas.

Una vez que se obtiene una "arquitectura de referencia" a partir de un análisis de dominio, es posible usarla en un generador de aplicaciones tal como el modelo GenVoca el cual depende de la descomposición jerárquica de los sistemas del dominio en capas que importan y exportan interfases estandarizadas.

Un proceso de desarrollo basado en arquitecturas de software incluye actividades como crear el caso del negocio para el sistema, entender los requerimientos, crear o seleccionar la arquitectura que se va a usar, representarla, comunicarla y evaluarla en un proceso cíclico hasta llegar a la implementación del sistema mismo. Este enfoque de desarrollo cae dentro del modelo de ciclo de vida en espiral.

Sin embargo, un sistema o aún un generador requiere de algunas técnicas de implementación de un sistema como son el uso de compiladores o de lenguajes de programación orientada a objetos e inclusive de métodos de representación de la arquitectura misma.

En el siguiente capítulo se presentan algunas de las técnicas que se usaron en la implementación del sistema de estimación de tiempos de maquinado en fresadoras de alta velocidad, tales como los compiladores, técnicas de programación orientada a objetos y el lenguaje de representación gráfica usado en la arquitectura del sistema, el UML.

CAPÍTULO III.

TÉCNICAS DE IMPLANTACIÓN DE UN SISTEMA DE SOFTWARE.

En el desarrollo de software existen muchas técnicas y herramientas que pueden usarse para apoyarlo e implementarlo. En este capítulo se presenta una breve explicación de aquellas técnicas que fueron usadas en el desarrollo del sistema para predecir tiempos de maquinado en fresadoras de alta velocidad, mismo que es tema del presente trabajo.

En este capítulo se habla de compiladores, técnicas de representación de arquitecturas como el lenguaje UML y de programación orientada a objetos.

3.1. COMPILADORES [AHO;1990].

Un compilador es un programa que lee un programa escrito en un lenguaje llamado "lenguaje fuente", y lo traduce a otro programa equivalente llamado "lenguaje objeto". Si existen errores en el programa fuente el compilador lo informa al usuario (Figura 3.1).

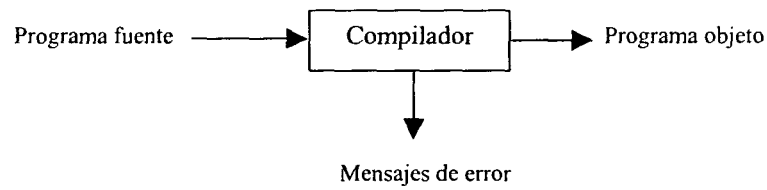


Figura 3.1 Proceso de compilación.

El lenguaje objeto puede ser otro lenguaje de programación o el lenguaje máquina de una computadora, o simplemente un código que se emplea posteriormente con otro lenguaje o en alguna aplicación.

Un proceso de compilación consta de dos partes principales: análisis y síntesis. La primera divide al programa fuente en sus componentes y crea una representación intermedia del programa fuente. La síntesis construye el programa objeto a partir de la representación intermedia del programa fuente.

Tradicionalmente se concibe un compilador como un programa que traduce un programa fuente al lenguaje ensamblador o de máquina de algún tipo de computadora. Sin embargo, hay aplicaciones donde se usa la tecnología de los compiladores con bastante frecuencia y que al parecer no están relacionadas con el concepto que se tiene, por ejemplo, programas que usan cadenas de caracteres como códigos para formatear texto.

Tal es el caso que nos ocupa en el presente trabajo, donde se toma la salida producida por un sistema CAD/CAM (el programa de CN) y se compila para eliminar las instrucciones que no involucran tiempo de maquinado.

3.1.1 Fases de un Compilador.

Un compilador está dividido en varias fases, cada una de las cuales aplica una transformación al programa fuente (Figura 3.2).

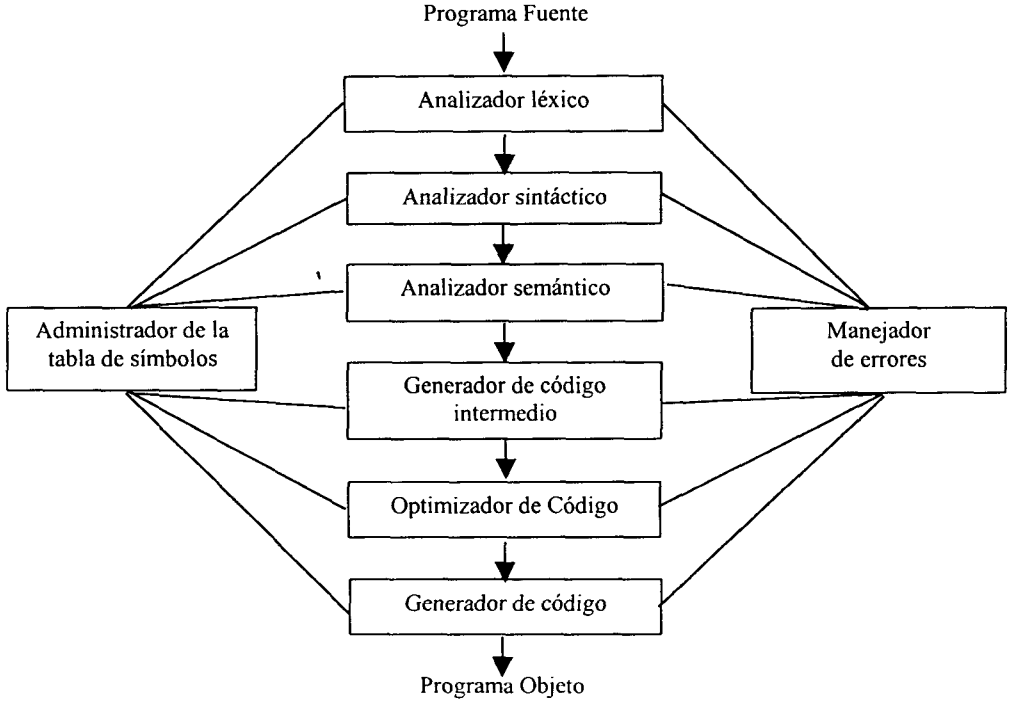


Figura 3.2 Fases de un compilador.

En la práctica estas fases pueden agruparse en una etapa inicial y una etapa final. La etapa inicial comprende aquellas fases que dependen principalmente del lenguaje fuente y que son en gran parte independientes de la máquina en que se ejecuta el programa. Normalmente se incluye el análisis léxico y sintáctico, la creación de la tabla de símbolos, el análisis semántico y la generación de código intermedio. Incluye además, el manejo de errores correspondiente a cada una de esas fases.

La etapa final incluye aquellas partes del compilador que dependen de la máquina objeto y, en general esas partes no dependen del lenguaje fuente, sino sólo del lenguaje intermedio. En esta etapa se encuentran aspectos de la fase de optimización de código,

además de la generación de código, junto con el manejo de errores y las operaciones con la tabla de símbolos.

Análisis léxico: Durante esta fase se lee de izquierda a derecha la cadena de caracteres que constituye el programa fuente y se agrupa en componentes léxicos llamados **tokens.** Un token es una secuencia de caracteres que tiene un significado para el programa.

Por ejemplo la proposición

$$x = y + z * 60$$

se agruparía en los componentes léxicos siguientes:

1)	Identificador	x
2)	Símbolo de asignación	=
3)	Identificador	y
4)	Signo de suma	+
5)	Identificador	z
6)	Signo de multiplicación	*
7)	Número	60

Los espacios en blanco que separan los caracteres de estos componentes léxicos, normalmente se eliminan.

Análisis sintáctico: En esta fase se agrupan en frases gramaticales los componentes léxicos generados en la fase anterior. Tales frases gramaticales generalmente se representan mediante un árbol de análisis sintáctico (Figura 3.3)

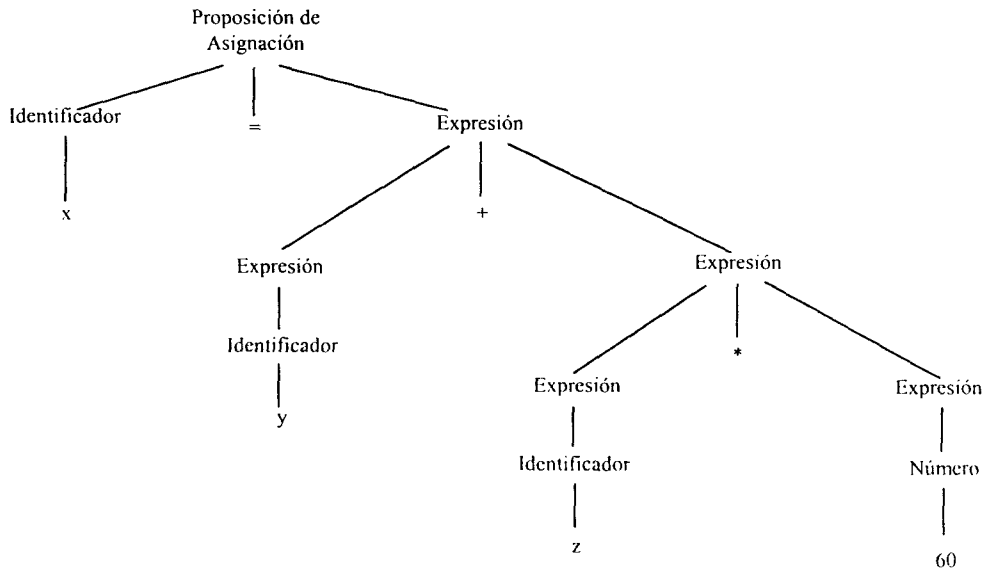


Figura 3.3 Árbol de análisis sintáctico.

La estructura jerárquica de un programa normalmente se expresa utilizando reglas recursivas. Por ejemplo, se pueden dar las siguientes reglas como parte de la definición de expresiones:

1. Cualquier identificador es una expresión.
2. Cualquier número es una expresión.
3. Si expresión1 y expresión2 son expresiones, entonces también lo son:
   $expresión_1 + expresión_2$
   $expresión_1 + expresión_2$
   $(expresión_1)$

Las reglas (1) y (2) son básicas es decir, no recursivas, en tanto que la regla (3) define expresiones en función de operadores aplicados a otras expresiones es decir, es recursiva.

Análisis semántico: En esta fase se ejecutan algunas revisiones para asegurarse que los componentes de un programa se agrupan de un modo significativo.

Tabla se símbolos: Es una estructura de datos que contiene un registro por cada identificador del programa fuente, con un campo para cada atributo del identificador.

El analizador léxico detecta un identificador en el programa fuente y lo introduce a la tabla de símbolos. Las fases restantes introducen información sobre los identificadores en la tabla de símbolos y después la utilizan de varia formas.

Errores: En cada fase se pueden encontrar errores. Estos errores deben tratarse de alguna manera para evitar que el proceso de compilación se termine y no sea posible detectar más errores en el programa.

Código intermedio: Algunos compiladores generan una representación intermedia explícita del programa fuente. Esta representación debe ser fácil de producir y fácil de traducir al programa objeto.

Optimización: Trata de mejorar el código intermedio de modo que resulte un código de máquina más rápido de ejecutar.

Generación de código: Se refiere al código objeto que por lo general consiste de código máquina relocalizable o código ensamblador.

Dentro del sistema de predicción de tiempos de maquinado en fresadoras de alta velocidad, la tecnología de los compiladores se usa para estandarizar la salida de los programas APT que definen los movimientos que deberán realizar los diferentes ejes de una máquina herramienta durante el proceso de maquinado.

Una vez estandarizada esta salida, se efectúa el procesamiento de las instrucciones de movimiento para efectuar el cálculo del tiempo que tomará obtener la pieza terminada. Para el procesamiento de las instrucciones y el cálculo se definió una arquitectura que se

describe en el capítulo V y para la cual se utilizaron herramientas de representación de modelos como el lenguaje modelación unificado (UML por sus siglas en inglés).

3.2. UML.

UML surge como un lenguaje estándar de modelado de sistemas, apoyado por empresas como Hewlett-Packard, i-Logix, IBM, ICON Computing, Intellicorp, MCI Systemhouse, Microsoft, ObjecTime, Oracle, Platinum Technology, Ptech, Rational Software, Reich Technologies, Sterling Software, Softeam, Taskon y Unisys [González; 1998].

Es un lenguaje estándar para representar sistemas de software, aunque no está limitado a ello ya que es lo suficientemente expresivo para modelar sistemas que no están relacionados con el software [Booch; 1999].

La modelación dentro del un proceso de desarrollo de software es tan importante como los planos y las especificaciones en la industria de la construcción por diversas razones entre las cuales [González; 1998] menciona las siguientes:

a) Los modelos representan la forma de ir plasmando las aproximaciones e ideas necesarias para resolver los requerimientos del cliente;

b) Son esenciales para la comunicación entre los distintos equipos de trabajo y entre las distintas disciplinas que participan en un proyecto;

c) Es mucho más económico hacer correcciones sobre un modelo "en papel" que hacerlas durante la construcción misma.

d) La documentación brinda una mayor facilidad para dar mantenimiento a una aplicación.

Se construyen modelos para comunicar la estructura y el comportamiento deseados en el sistema, para visualizar y controlar su arquitectura, para tener un mejor entendimiento y exponer las oportunidades de simplificación y reuso y para una mejor administración de los riesgos [Booch; 1999].

A continuación se enumeran algunos de los objetivos principales de UML [González; 1998]:

1.- Permite la interoperabilidad entre distintas herramientas ya que es posible el intercambio de diagramas y de formas de representación. De hecho esta fue una de las principales razones de su creación.

2.- Proporciona un lenguaje visual de modelado, y permite aprovechar componentes, plantillas y patrones a partir de soluciones exitosas que ya han sido probadas.

3.- Es independiente de los lenguajes, de las bases de datos y de la marca del software y del equipo de cómputo. Es solamente un medio para especificar, visualizar y documentar los artefactos del desarrollo e implantación de sistemas.

Cabe hacer hincapié en que UML no es un método de análisis y diseño sino solamente un lenguaje independiente del método empleado en el proceso de desarrollo, que contempla la estandarización a nivel de símbolos y de notación. Dado que no existe una secuencia rígida de actividades que sea aplicable a todos los tipos de desarrollo, UML se puede aplicar al desarrollo de cualquier sistema.

Elementos Principales de UML

UML es un lenguaje provee un vocabulario y reglas para combinar las palabras de ese vocabulario con el único objetivo de facilitar la comunicación. El vocabulario y las reglas se centran en la representación física y conceptual de un sistema.

El vocabulario de UML se compone de tres bloques de construcción que son:

1. Elementos que pueden ser estructurales, de comportamiento, de agrupamiento y de notación.
2. Relaciones.
3. Diagramas.

Elementos estructurales.

Son las partes estáticas de un modelo UML que representan componentes físicos o conceptuales (Tabla 3.1).

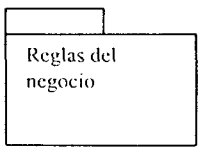
Elementos de comportamiento.

Son las partes dinámicas de los modelos UML. Representan su comportamiento a través del tiempo y del espacio. Básicamente son dos los elementos de comportamiento: Las interacciones y las máquinas de estado (Tabla 3.2)

Elementos de agrupamiento .

Son partes organizacionales de los modelos de UML. Básicamente hay un elemento de agrupamiento llamado *paquete.*

Paquete: Es un mecanismo de propósito general para organizar elemento en grupos. Los elementos estructurales, de comportamiento y aún otros elementos de agrupamiento, pueden ponerse dentro de un paquete.



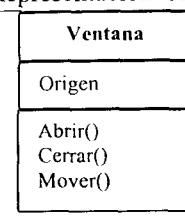
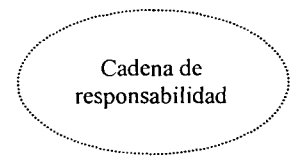
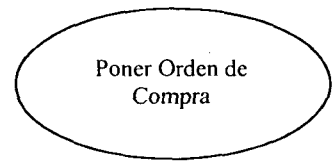
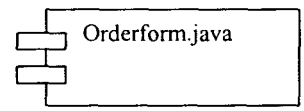
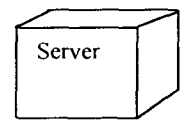
Elemento	Descripción	Representación Gráfica
Clases	Describen un conjunto de objetos que comparten los mismos atributos, operaciones, relaciones y semántica.	
Interfases	Son conjuntos de operaciones que especifican un servicio de una clase o componente. Describen el comportamiento externo visible de ese elemento.	
Colaboraciones	Son sociedades de roles y otros elementos que trabajan juntos para proveer algún comportamiento cooperativo que es mayor que la suma de sus elementos.	
Casos de uso	Son descripciones de un conjunto de secuencias de acciones que un sistema ejecuta para obtener un resultado observable de valor a un actor particular del sistema.	
Componente	Es un parte física y sustituible de un sistema que se adecúa y provee la realización de un conjunto de interfases	
Nodo	Es un elemento físico que existe en tiempo de ejecución y representa un recurso computacional, generalmente con al menos algo de memoria y a menudo, capacidad de procesamiento.	

Tabla 3.1 Elementos Estructurales de UML

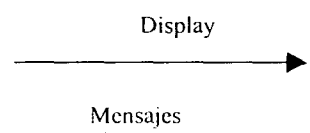
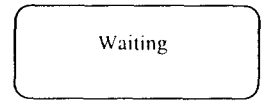
Elemento	Descripción	Representación Gráfica
Interacciones	Es el comportamiento que comprende el conjunto de mensajes intercambiados entre un conjunto de objetos dentro de un contexto particular con un propósito específico. También se involucran secuencias de acción y ligas.	
Máquinas de estado	Es el comportamiento que especifica la secuencia de estados que toma un objeto o una interacción como respuesta a eventos durante su ciclo de vida, junto con su respuesta a esos ciclos de vida.	

Tabla 3.2 Elementos De Comportamiento de UML

Elementos notacionales.

Son partes explicatorias de los modelos UML. Son comentarios que se aplican para describir, iluminar y remarcar algo sobre algún elemento en el modelo

Nota: Es un símbolo simple para renderizar limitantes y comentarios ligados a un elemento o colección de elementos.

Relaciones.

Existen cuatro tipos de relaciones que se explican en la Tabla 3.3:

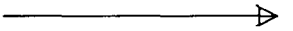
Relación	Descripción	Representación Gráfica
Dependencia	Es una relación semántica entre dos cosas en las cuales un cambio en una (el elemento independiente) puede afectar la semántica de la otra (el elemento dependiente).	
Asociación	Es una relación estructural que describe un conjunto de ligas, siendo una liga una conexión entre objetos. La agregación es una clase especial de asociación que representa un relación estructural entre un todo y sus partes.	
Generalización	Es una relación de especialización / generalización en la cual los objetos o los elementos especializados (los hijos) son sustituibles por objetos de los elementos generalizados (el padre). De este modo el hijo comparte la estructura y el comportamiento del padre.	
Realización	Es una relación semántica entre clasificadores, de los cuales un clasificador especifica un contrato que otro clasificador garantiza que se llevará a efecto. Las realizaciones se dan entre interfases y las clases o componentes que las realizan, y entre los casos de uso y las colaboraciones que las realizan.	

Tabla 3.3. Relaciones de UML

Diagramas de UML.

Son representaciones gráficas de un conjunto de elementos conectados . Los diagramas se usan para visualizar un sistema desde diferentes perspectivas. En teoría un diagrama puede contener cualquier cantidad de elementos y relaciones sin embargo, en la práctica solo unos cuantas de esas combinaciones son perspectivas útiles desde el punto de vista arquitectónico. UML incluye 9 de esos diagramas:

Un diagrama de clases muestra un conjunto de clases, interfases y colaboraciones y sus relaciones. Son la representación estática de un sistema.

Un diagrama de objetos muestra un conjunto de objetos y sus relaciones. Representan un momento estático de instancias de las cosas encontradas en los diagramas de clases. Modelan el comportamiento de un sistema.

Un diagrama de casos de uso muestra un conjunto de casos de uso y actores (un caso especial de clases) y sus relaciones.

Los diagramas de secuencia y las colaboraciones son dos clases de diagramas de interacción. Los diagramas de interacción muestran un conjunto de objetos y sus relaciones incluyendo mensajes que pueden ser enviados entre ellos. Son el aspecto dinámico de un sistema. El diagrama de secuencia es un diagrama de interacción que enfatiza el orden en el tiempo de los mensajes en tanto que el diagrama de colaboración es un diagrama de interacción que enfatiza la organización estructural de los objetos que envían y reciben mensajes.

Un diagrama de estado muestra una máquina de estado que consiste de estados, transiciones, eventos y actividades. Modela el comportamiento de una interfase, clase o colaboración y enfatiza el comportamiento ordenado por eventos lo cual es especialmente útil en el modelado de sistemas reactivos.

Un diagrama de actividad es una clase especial de diagrama de estados que muestra el flujo de actividad a actividad dentro de un sistema

Un diagrama de componentes muestra la organización y dependencias entre un conjunto de componentes. Está relacionado con los diagramas de clases en el sentido de que un componente típicamente mapea a una o más clases, interfases o colaboraciones.

Un diagrama de despliegue muestra la configuración de nodos de procesamiento en tiempo de ejecución. Se relaciona con los diagramas de componentes en el sentido que un nodo típicamente encierra uno o más componentes.

Una vez definido el modelo arquitectónico tanto del sistema como de la familia de productos de software, la implementación se realizó empleando la tecnología de programación orientada a objetos y específicamente el compilador de Borland C++ Builder IV.

3.3. TECNOLOGÍA Y PROGRAMACIÓN ORIENTADA A OBJETOS (TOO, POO).

La programación orientada a objetos se toma como un paradigma de programación, una nueva manera de hacer las cosas, donde se toma el concepto de que todo es un objeto. Cada objeto es una entidad independiente y encierra en sí misma su propio conocimiento, e interactúa con los demás objetos a través de mensajes [Greiff; 1994].

El mismo [Greiff; 1994] define a un lenguaje de programación orientado a objetos como aquel que soporta objetos donde un objeto es "algo que contiene un conjunto de operaciones y un estado que 'recuerda' el efecto de las operaciones".

Un objeto se define a partir de una clase que contiene los elementos de datos y métodos (procesos o funciones) que proveen la funcionalidad de los objetos.

Un objeto es una instancia de las clases y puede llamar un método de otro objeto a través de mensajes. Los métodos definen el comportamiento de los objetos.

Calderón [Calderón; 1994] menciona Smalltalk como el lenguaje de programación orientado a objetos, a partir del cual surgen los demás pero que no es propiamente OO puro; Eiffel y Actor son otros ejemplos. También menciona que otros lenguajes como LISP, C y Pascal se vieron motivados a agregar soporte para objetos.

Actualmente existe una gran cantidad de herramientas OO que apoyan en el desarrollo de proyectos de software exitosos. En general, la TOO se ha reconocido como un gran apoyo debido a una serie de razones entre las que sobresalen [Ibargüengoitia;1999]:

- Proporciona un lenguaje común con el dominio del problema a resolver, lo que facilita la comunicación entre los usuarios del sistema y los desarrolladores. Esto es debido a que en la fase de análisis el lenguaje es absolutamente en términos del dominio del problema. En esos términos, el desarrollador modela lo que en fases subsecuentes del desarrollo se convertirá en un sistema computacional.

- La estructura general del sistema es más cercana a la del dominio de la aplicación. Esto es debido a que se construye con base a objetos y clases (que son abstracciones de las entidades existentes en el dominio del problema), lo que tiene varias ventajas:

  · Se identifican clases y objetos generales del dominio de la aplicación que pueden especializarse a través de la herencia. Esto permite crear abstracciones más generales y no particulares a un problema.

    La herencia es una de las características de los lenguajes de POO, la cual permite definir nuevas clases de objetos a partir de los ya existentes. Los objetos nuevos heredan la funcionalidad de los ya existentes y pueden ser mejorados, ampliados o especializados [Katrib; 1994].

    Sin embargo no todos los lenguajes de POO implementan la herencia de la misma manera. Una diferencia importante es en cuanto al nivel de acceso que una clase tiene sobre los miembros de la clase de que está heredando.

  · Las clases se agrupan en conjuntos relacionados.

- Las clases encapsulan (ocultan la información de los detalles de implementación) las propiedades y comportamientos generales, lo que facilita reaccionar a cambios pues éstos están delimitados a la clase que lo amerite, lo que es transparente al resto de la aplicación.

- Se fomenta la reutilización (volver a utilizar) de componentes del sistema (modelos de análisis, diseño, código y demás), lo que facilita hacer nuevos desarrollos de forma más rápida.

- Se apega a un modelo de desarrollo de sistemas llamado incremental-iterativo, en el que se plantea inicialmente un núcleo básico de funcionalidades del sistema e iterativamente se va convergiendo en la solución completa, aumentando funcionalidades. Esto facilita el mantenimiento (que en realidad no existe) y la adaptación del sistema a nuevas necesidades, creando nuevas versiones (va evolucionando).

- Al construir sistemas OO, no sólo se busca resolver el problema sino además, en general se busca que el sistema cuente con otras características de los sistemas modernos como son: una interfaz amistosa a través de ventanas, botones, etc.; que corra en un ambiente cliente-servidor; que cuente con el apoyo de manejadores de bases de datos eficientes y rápidos. Los ambientes de desarrollo orientados a objetos (Delphi, Java. C++ y otros) cuentan con bibliotecas preparadas para construir sistemas con todas estas características, que no son exclusivas de su competencia, de forma que se pueden incorporar fácilmente.

En el presente trabajo se tomó ventaja de las características de la TOO para soportar el desarrollo del sistema y definir y crear sus componentes.

Resumen

Un compilador es un programa que traduce un programa escrito en un lenguaje llamado "lenguaje fuente", a otro programa equivalente llamado "lenguaje objeto" que generalmente es el lenguaje máquina. Sin embargo esta tecnología se utiliza también en otro tipo de aplicaciones que requieren de una transformación del código fuente a otro tipo de codificación que puede ser simplemente texto formateado o códigos especiales.

Un compilador consta de varias fases tales como el análisis léxico, el análisis sintáctico, el análisis semántico, el generador de código intermedio, el optimizador de código y el generador de código, todas las cuales realizan una transformación del programa fuente de manera incremental hacia el programa objeto.

Sin embargo, en el desarrollo de un sistema intervienen además de los procesos de compilación, los de modelación del sistema mismo. La modelación arquitectónica ayuda a visualizar el producto final ya que establece la estructura y presenta el comportamiento de

un sistema. Además, los modelos sirven de guía en la construcción del sistema y documentan las decisiones tomadas.

UML surgió como un lenguaje estándar de modelado de sistemas para representar sistemas de software, aunque no está limitado a ello ya que es lo suficientemente expresivo para modelar sistemas que no están relacionados con el software

Se construyen modelos para comunicar la estructura y el comportamiento deseados en el sistema, para visualizar y controlar su arquitectura, para tener un mejor entendimiento y exponer las oportunidades de simplificación y reuso y para una mejor administración de los riesgos [Booch; 1999].

El vocabulario de UML se compone de tres bloques de construcción que son:

1. Elementos que pueden ser estructurales, de comportamiento, de agrupamiento y de notación.
2. Relaciones.
3. Diagramas.

Una vez obtenida la arquitectura y representada en un modelo con un lenguaje de modelación como UML, es necesaria la implementación de los sistemas de software construidos a partir de tal arquitectura.

En este sentido se ha reconocido a la POO, como un gran apoyo debido a que proporciona un lenguaje común con el ambiente del problema a resolver e identifica clases y objetos generales del dominio de la aplicación que pueden especializarse a través de la herencia.

Un lenguaje de programación orientado a objetos es aquel que soporta objetos, donde un objeto es "algo que contiene un conjunto de operaciones y un estado que 'recuerda' el efecto de las operaciones"

Además, la POO fomenta la reutilización de componentes del sistema (modelos de análisis, diseño, código y demás), lo que facilita hacer nuevos desarrollos de forma más rápida.

En el siguiente capítulo se analiza el dominio de aplicación sobre el que se trabajó y que es el de las máquinas fresadoras de alta velocidad. Se describen algunas de sus características comunes así como también de la variabilidad que se puede encontrar de una a otra. Se habla de los tipos de control y de los interpoladores, así como de los sistemas CAD/CAM que apoyan a los procesos de producción dentro de la industria manufacturera.

CAPÍTULO IV.

MAQUINAS-HERRAMIENTA.

En este capítulo se presenta el área de dominio de aplicación del presente trabajo de investigación: las máquinas herramienta y sus controles. Se establecen las definiciones de control numérico y control numérico computarizado. Se describen los tipos de control y de interpolación y se habla de la relación entre los sistemas CAD/CAM y los controles. Además, se describen algunos de los aspectos más importantes del lenguaje de programación APT.

4.1. PROGRAMA DE CONTROL NUMÉRICO (CN) Y CONTROL NUMÉRICO COMPUTARIZADO (CNC).

Dentro de la industria manufacturera se utilizan máquinas de producción denominadas máquinas-herramientas, tales como tornos, fresadoras, esmeriladoras, etc., que maquinan piezas a través de movimientos repetitivos de los diferentes ejes que las componen. Dichos ejes pueden ser lineales o rotatorios.

Los procesos de producción usando máquinas-herramientas han evolucionado hasta que, en nuestros días, los movimientos de sus ejes se controlan de manera automática previniendo que la producción tenga defectos provocados por la intervención humana. A esta situación ha contribuido en gran medida la introducción de la microcomputadora como sistema de control y se ha visto reforzada por el advenimiento de los sistemas de software de programación de partes asistidos por computadora, que cada vez son menos costosos.

Bajo este contexto, cada pieza que se va a maquinar requiere de un programa concreto que es el que contiene la información de los desplazamientos de los ejes y las funciones de interrupción tales como aquellas que se requieren para el cambio de herramientas. Las máquinas herramienta utilizan números para definir desplazamientos relativos entre la herramienta y la pieza, avances, velocidad de giro del husillo, tipo de herramienta que se emplea o las instrucciones auxiliares para el cambio de herramienta o de pieza o para el empleo del refrigerante. Se conoce como programa de control numérico (CN) al agrupamiento de estos valores de acuerdo a la secuencia de maquinado elegida y con una estructura definida por la sintaxis del lenguaje de control empleado.

Existe también el concepto de control numérico computarizado ó CNC que no es más que el control numérico que incorpora uno o varios microprocesadores para la ejecución de las funciones de control. CN y CNC se emplean como sinónimos.

Podemos decir entonces que un control numérico es una computadora especial para controlar una máquina, un robot u otro sistema de mecanizado. Para funcionar requiere de un software de control que es el que determina las capacidades y funciones de la máquina herramienta, su velocidad, precisión, etc. [Kief; 1988].

Además, en el proceso de producción automático, el control numérico asume otras tareas y funciones tales como el diagnóstico y supervisión de las máquinas, enlace del CNC con una computadora central para el intercambio de datos, entrada manual de datos desde el teclado del CNC por medio de interfaces gráficas, etc. En la figura 4.1 se muestra la manera en que a partir del dibujo de la pieza a maquinar se genera el programa de CN que mueve los ejes de la máquina para producir la pieza.

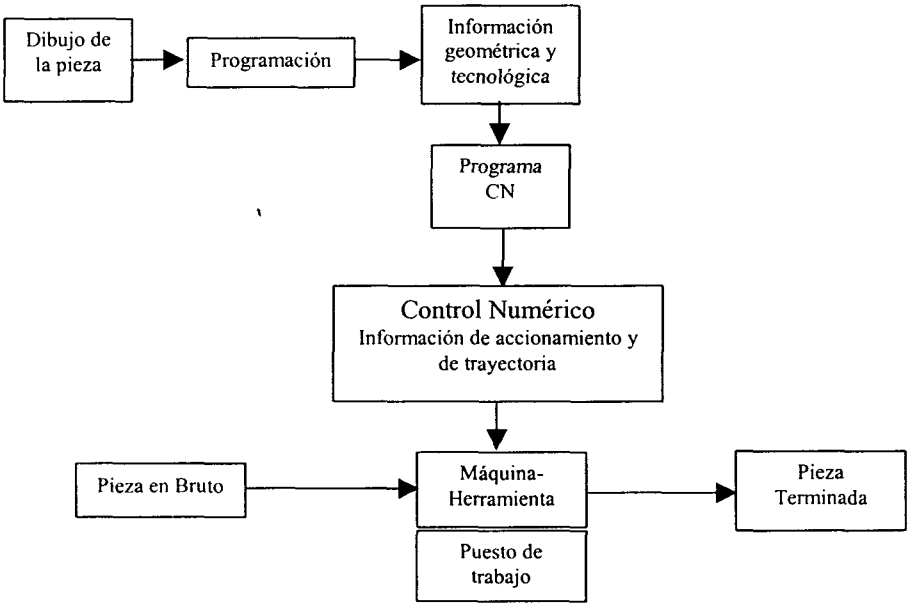


Figura 4.1.- Proceso de Programación del CN

Por una parte se tiene un diseño de la pieza que se va a maquinar, mismo que se dibuja usando algún sistema de programación de partes asistido por computadora. Estas mismas herramientas computacionales generan un código de programa de la parte dibujada. Este programa contiene la información geométrica necesaria para definir la pieza que, junto con la información tecnológica (tipo de herramienta, enfriador, etc.) permite la creación del programa de CN completo.

Este programa de control numérico se alimenta a la máquina herramienta para su ejecución. La máquina herramienta recibe además del programa de control numérico, una pieza en bruto sobre la cual se hacen los cortes de acuerdo a las trayectorias definidas en el programa CN para producir la pieza terminada.

4.2. TIPOS DE CONTROL.

De acuerdo a los componentes de control utilizados, se habla de controles mecánicos, eléctricos, neumáticos o hidráulicos cuya finalidad es colocar los diferentes ejes de la máquina en posiciones exactamente definidas. Aún hoy se utilizan levas y accionadores de levas para lograr esto. Así, es posible distinguir entre cuatro tipos de control [Kief; 1988] :

1.- Control de puntos.

2.- Control de segmentos.

3.- Control de trayectorias 2D.

4.- Control de trayectorias 3D.

En los dos primeros, todos los ejes se desplazan simultáneamente a gran velocidad hasta que cada uno de ellos ha alcanzado su posición. El maquinado de una pieza empieza en el momento en que los ejes están en su posición. Los controles de trayectoria pueden usarse en cualquier tarea de desplazamiento en la que se tengan que utilizar dos o más ejes con una relación exacta entre sí.

4.3. INTERPOLADORES.

Butler [Butler; 1968] define una **interpolación** como "la operación matemática que determina un valor no tabular $f(x)$ a partir de una tabla dada. Esta tabla, consiste de varios valores de una variable independiente x y los valores correspondientes de $f(x)$... "

Por su parte Wilson [Wilson; 1963] define un **interpolador** como "un dispositivo para encontrar la trayectoria para ser seguida por una herramienta de corte, cursor o elemento de una máquina-herramienta, cuando se le introduce una descripción matemática de ella. Provee el enlace entre los puntos programados y líneas o superficies lisas"

Los controles de trayectoria mencionados anteriormente requieren de un interpolador. En los CNC esto se refiere a un dispositivo electrónico especial que es el encargado de calcular todos los valores intermedios entre las posiciones inicial y final de un bloque sobre una curva matemáticamente definible, y accionar todos los ejes simultáneamente, cada uno a la velocidad adecuada para que lleguen al mismo tiempo a su posición final programada, haciendo que la herramienta recorra la mencionada curva [Kief; 1988].

4.3.1 Tipos de Interpolación.

Entre otros, existen los siguientes tipos de interpolaciones:

Interpolación lineal: Es la más comúnmente usada. La herramienta se desplaza en línea recta entre los puntos inicial y final (Figura 4.2). Las curvas se pueden aproximar por

trazos poligonales (Figura 4.3). Cuanto más cerca se encuentren los distintos puntos de apoyo, tanto más exacta será la aproximación al perfil dado. Sin embargo, con el número de puntos se eleva la cantidad de datos a procesar por unidad de tiempo, para un avance determinado.

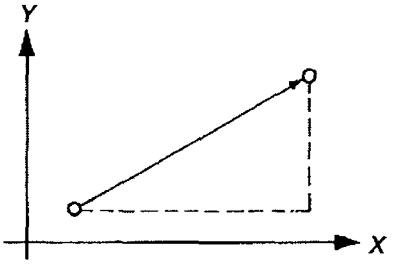


Figura 4.2.- Interpolación Lineal

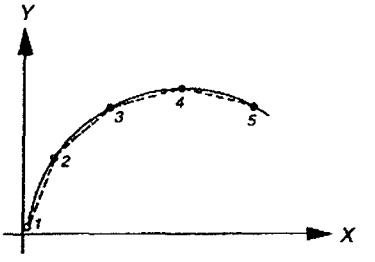


Figura 4.3.- Aproximación a una curva mediante un trazado poligonal.

Al momento de la puesta en marcha de esta interpolación existen dos subclasificaciones de la trayectoria en:

a) **Movimiento rápido** que es un desplazamiento de la herramienta de un punto a otro pero sin efectuar ningún corte en la pieza y sin garantizar la trayectoria específica de dicha herramienta.

b) **Movimiento de maquinado** en el que la herramienta se desplaza al mismo tiempo que se va efectuando un corte en la pieza y se controla la trayectoria específica de dicha herramienta.

Interpolación circular: La interpolación circular se programa de forma distinta según el control: en cuartos de círculo, como círculo completo, por parámetros IJK, con ayuda de la indicación del centro del círculo o por programación del punto final y el radio del círculo (Figura 4.4).

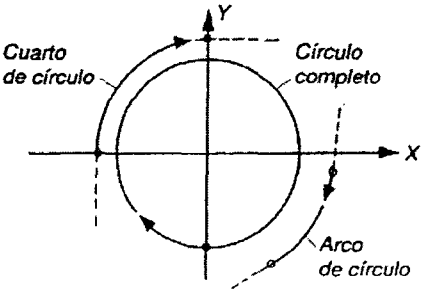


Figura 4.4.- Interpolación Circular [Kief; 1988]

Interpolación parabólica: Usa una ecuación de segundo grado para producir círculos, elipses, parábolas e hipérbolas. Ofrece ventajas especiales sólo en máquinas de

entre tres y cinco ejes, ya que necesita menos bloques que la interpolación lineal para desplazamientos simultáneos de varios ejes (Figura 4.5).

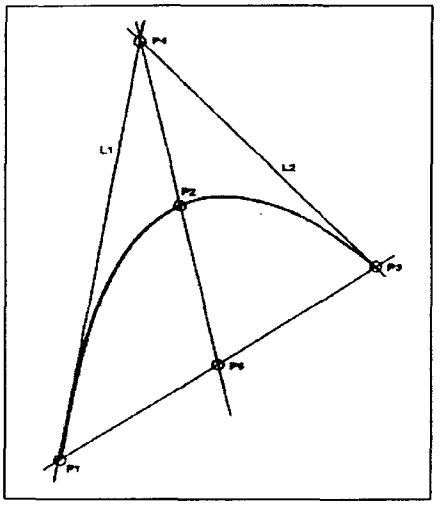


Figura 4.5.- Interpolación Parabólica [Kief; 1988]

Las interpolaciones circular y parabólica reducen el número de datos a introducir y facilitan la programación de las trayectorias incrementando su precisión.

Interpolación por splines: Se entiende por ello la separación de curvas matemáticas de orden superior donde las transiciones son tangenciales. Mediante este tipo de interpolación se pueden presentar curvas de formas complejas con un número de bloques más reducido que con la aproximación por trazos poligonales e interpolación lineal. Este tipo de interpolación se usa sobre piezas que requieren de una forma libre y que no se ajustan a ninguna ecuación analítica

4.4 SISTEMAS CAD/CAM.

Actualmente existe una amplia variedad de sistemas CAD (Diseño Asistido por Computadora) para los más amplios campos de aplicación. Uno de dichos campos se refiere al área de la fabricación mecánica. Los sistemas CAD/CAM (Manufactura asistida por computadora) apoyan en el diseño, dibujo y fabricación de diferentes piezas. Dado que prácticamente todos los proveedores de sistemas de programación para CN con asistencia gráfica hablan ya de CAD/CAM es importante establecer la diferencia entre ambos.

El **CAD** abarca todos los procesos para la elaboración asistida por computadora de los datos de las piezas como planos, listas de piezas y modelos gráficos de las mismas [Kief; 1988]. Los sistemas CAD almacenan dibujos en representaciones de tres dimensiones. Usando los ejes X, Y y Z permiten la representación de puntos, líneas y curvas. A su vez, una pieza se define por su geometría la cual incluye puntos, líneas,

círculos, planos, cilindros y otras superficies permitiendo de manera natural el uso de los sistemas CAD para su representación [Chang; 1998].

Una de las ventajas que presentan los sistemas CAD es su interactividad y la facilidad que proveen en la creación de nuevos diseños. Además, proveen la posibilidad de simular el comportamiento del sistema antes de la construcción del prototipo, modificando si es necesario sus parámetros, y permiten la generación de planos con todo tipo de vistas, detalles y secciones.

El **CAM** se integró al CAD para la manipulación de las máquinas herramienta que intervienen en la fabricación y de los dispositivos especiales o robots. Históricamente, los CAD comenzaron como una ingeniería tecnológica computarizada mientras que los CAM eran una tecnología semiautomática para el control de la máquina de forma numérica [Mompím; 1988].

Básicamente el CAM es la utilización de computadoras para tareas técnicas y de gestión técnica en la fabricación y montaje de piezas. Tales tareas incluyen la elaboración de planos de mecanizado, planos de herramientas y la ejecución de los programas de CN. CAM es la parte que se encarga de la elaboración del producto ya que concentra la información que se relaciona con la producción misma, incluidas las listas de herramientas y las secuencias de trabajo [Kief; 1988].

4.4.1. Relación entre el CAD y la programación de CN.

Existen dos formas en que el CAD se relaciona con la programación de CN [Chang; 1998]:

1.- El diseñador toma el programa de CN generado directamente en el sistema CAD (programación automatizada de partes).

2.- La programación del CN se hace aparte del CAD pero usando los datos que este genera aplicados a un sistema de programación específico (programación manual de partes).

El primer caso es una situación muy ventajosa dado que puede obtenerse todo el programa CN mediante algoritmos geométricos de los datos en el sistema CAD. Los datos tecnológicos como avance, velocidad de giro del husillo o profundidad de corte, están limitados en la mayor parte de los casos a algunos parámetros al inicio del programa. El programa CN completo consta principalmente de instrucciones geométricas. La utilización de datos del CAD para la programación CN presupone bloques de datos correctos absolutos.

4.5 LENGUAJE APT.

Uno de los más populares lenguajes de programación de partes en los Estados Unidos es el APT (Automatically Programmed Tool). Algunas de sus características se enumeran a continuación [Chang; 1998]:

- Define superficies indefinidas tridimensionales y puntos para representar piezas que serán maquinadas.
- Las superficies se definen en un sistema de coordenadas XYZ elegido por el programador de la pieza.
- En programación, la herramienta ejecuta todos los movimientos mientras que la pieza es estacionaria.
- Utiliza la interpolación lineal para aproximar curvas.

4.5.1 Convenciones del Lenguaje APT [IBM; 1991].

Un programa APT consiste de una serie de líneas de datos. Cada línea representa una sentencia a ejecutarse, pero una sentencia puede componerse de más de una línea.

Una línea de datos puede contener caracteres especiales, símbolos de variables o identificadores, etiquetas, números o palabras de vocabulario.

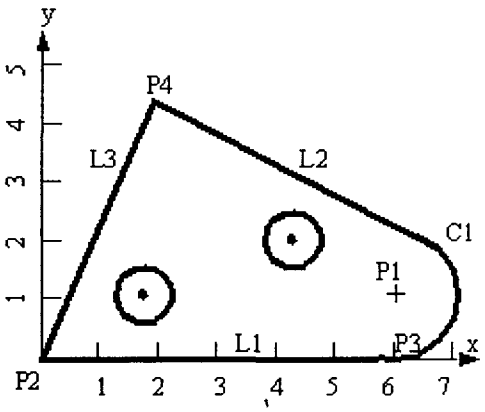
Las palabras de vocabulario son palabras de dos a seis caracteres que tienen un significado especial para el lenguaje. A continuación se clasificas las palabras de vocabulario de acuerdo a su función:

- Palabras que especifican el tipo geométrico de una entidad definida: POINT, LINE, CIRCLE, PLANE, CYLINDR.
- Palabras que especifican movimiento: GOTO, GOLFT, GORGT, GOFWD, GOBACK, RAPID.
- Palabras que especifican funciones relacionadas con el dispositivo que se está controlando: FEDRAT, SPINDL, COOLNT.

El lenguaje APT no tiene restricciones en cuanto al orden en que se especifican las sentencias. La regla principal es que no se puede hacer referencia a un símbolo que no ha sido definido.

La Figura 4.6. presenta un ejemplo de un programa APT. El programa produce una curva como la que se muestra y que esta definida por los puntos P1, P2, P3 y P4. En el programa primero se establecen esos cuatro puntos para ser usados posteriormente en la

definición de las líneas L1 y L3 y el plano PL1. La curva C1 se define con el comando CIRCLE usando como parámetro P1 . La línea L2 se define desde P4 como una tangente a C1. Posteriormente se inician los movimientos con la instrucción GO y las líneas, planos y círculos antes definidos, hasta dibujar el contorno de la curva objetivo.

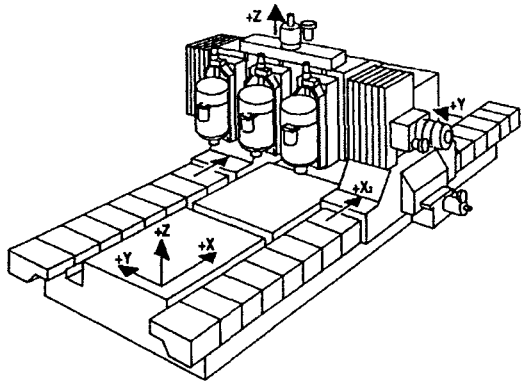



```
P0=POINT/0,-1.0,0
P1=POINT/6.0,1.125,0
P2=POINT/0,0,0
P3=POINT/6.0,0,0
P4=POINT/1.75,4.5,0
L1=LINE/P2,P3
C1=CIRCLE/CENTER,P1,RADIUS,1.125
L2=LINE/P4,LEFT,TANTO,C1
L3=LINE/P2,P4
PL1=PLANE/P2,P3,P4
FROM/P0
GO/TO,L1,TO,PL1,PAST,L3
GORGT/L1,TANTO,C1
GOFWD/C1,PAST,L2
GOFWDL2,PAST,L3
GOLFT/L3,PAST,L1
GOTO/P0
```

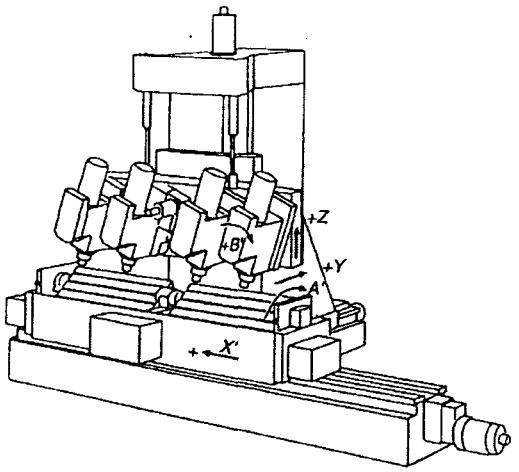
Figura 4.6.- Programa APT

4.6 MÁQUINAS FRESADORAS DE ALTA VELOCIDAD.

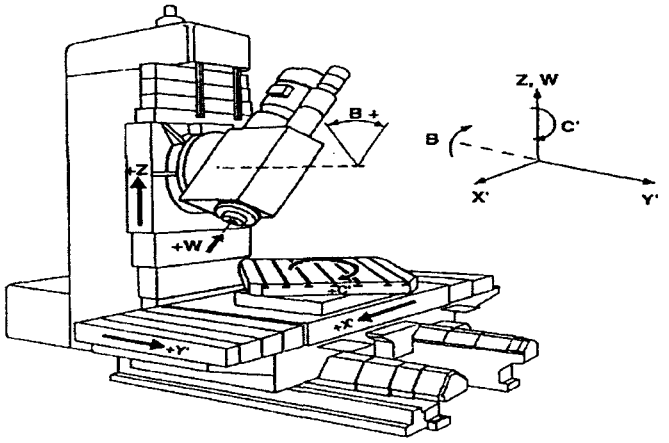
Una fresadora es un tipo específico de máquina herramienta que puede tener de 3 a 5 ejes (Figura 4.7). Los movimientos de sus ejes son rotatorios y de traslación. Usa herramientas de corte que a través de trayectorias bien definidas en un programa de control numérico generan una pieza o producto.



(a) Fresadora de pórtico de tres ejes con tres unidades de fresado.



(b) Fresadora cinco ejes con cuatro husillos paralelos.



(a) Fresadora de cinco ejes

Figura 4.7.- Diferentes tipos de fresadoras[Kief; 1988]

A mayor número de ejes en una fresadora es posible la producción de piezas cada vez más complicadas geométricamente.

La evolución de la industria manufacturera ha llevado a la existencia de fresadoras de alta velocidad. Sin embargo no hay una estandarización en cuanto al significado del término "alta velocidad". Algunos fabricantes de fresadoras destacan las velocidades de giro del husillo, otros las velocidades de avance que dichas máquinas pueden alcanzar. Pero

es sólo la combinación de ambos factores lo que permite un maquinado eficiente de una pieza a una alta velocidad [Kief; 1988]

Las velocidades de giro del husillo de las fresadoras de alta velocidad alcanzan hasta 20,000 rev/min y se trasladan a velocidades de hasta 40 m/min. [Rodríguez; 1998].

Este tipo de máquinas se utilizan en la construcción aeronáutica de componentes integrales de aluminio [Kief; 1988] así como en la manufactura de partes automotrices [Rodríguez; 1998].

Resumen

Las máquinas-herramienta producen piezas a través de movimientos repetitivos de los diferentes ejes que las componen, y que en la actualidad son controlados de manera automática usando la computadora como sistema de control. Las fresadoras de alta velocidad son un ejemplo de máquinas herramientas, empleadas principalmente en la construcción aeronáutica de componentes integrales de aluminio y en la manufactura de partes automotrices.

Para maquinar una pieza, una máquina herramienta necesita un programa de control numérico que no es más que el agrupamiento de números que definen los desplazamientos relativos entre la herramienta y la pieza de acuerdo a una secuencia de maquinado elegida y la sintaxis del lenguaje de control usado. Al empleo de la computadora para funciones de control se le conoce como control numérico computarizado.

La función fundamental de un control es el desplazamiento de los diferentes ejes de una máquina herramienta hasta un punto dado. Para efectuar dicho desplazamiento se requiere de un interpolador el cual en un CNC es un programa de software que se encarga de calcular todos los valores intermedios entre las posiciones inicial y final de un bloque sobre una curva matemáticamente definible, y acciona todos los ejes simultáneamente, cada uno a la velocidad adecuada para que lleguen al mismo tiempo a su posición final programada, haciendo que la herramienta recorra la mencionada curva.

Existen varios tipos de interpolación entre las cuales se mencionan la lineal, circular, parabólica y splines.

Para apoyar a los procesos de diseño y programación de piezas de maquinado se han creado sistemas CAD/CAM mismos que proveen facilidad y flexibilidad en tareas tales como elaboración de diferentes vistas, planos de mecanizado, listas de piezas y modelos gráficos de las mismas, planos de herramientas y la generación de los programas de CN.

Uno de los lenguajes más usados en la programación de partes es el APT.

La teoría sobre las máquinas herramientas forma parte del dominio de aplicación en estudio. En el siguiente capítulo nos introduciremos más a fondo en el dominio, específicamente en la problemática que se enfrenta en el presente trabajo y en base al análisis efectuado se define la arquitectura para la familia de sistemas de predicción de tiempo.

CAPÍTULO V.

PROCESO DE DESARROLLO PARA EL SISTEMA DE PREDICCIÓN DE TIEMPO DE MAQUINADO (PreTie).

Para la definición del modelo arquitectónico y la implementación de la herramienta computacional, propuesta como solución al problema de estimación de tiempos de maquinado en fresadoras de alta velocidad, se utiliza el proceso de desarrollo basado en arquitecturas explicado en el capítulo II. En este capítulo se detallan los pasos de ese proceso aplicados a la herramienta de predicción de tiempos.

5.1. CREACIÓN DEL CASO DEL NEGOCIO PARA EL SISTEMA.

El mercado objetivo del sistema creado son las industrias que emplean fresadoras de alta velocidad en sus procesos de manufactura. Entre estas se cuenta principalmente la industria automotriz y la aeronáutica.

También es importante considerar que este tipo de aplicación puede extenderse a otros dominios además del de las fresadoras e incluir otro tipo de máquinas herramienta.

Algunas de las limitantes que se enfrentaron en el desarrollo del presente trabajo fue la existencia de un sólo modelo matemático mismo que se creó a la par que el desarrollo del sistema de predicción. El modelo matemático obtenido representa el comportamiento de una fresadora de tres ejes que soporta interpolación lineal y es el único con que se cuenta actualmente.

El diseño del sistema sin embargo, tomó en consideración las posibles variaciones que pudieran darse en el dominio de aplicación, tales como el número de ejes de una fresadora que pueden ser de 3 a 5, el tipo de interpolación que puede ser no sólo lineal sino también circular, parabólica, splines o helicoidal, y el hecho de que para cada tipo de interpolación pueden definirse más de un modelo matemático y que cada eje tiene sus propios parámetros de calibración.

5.2. ELEMENTOS DEL DOMINIO Y ANÁLISIS DE REQUERIMIENTOS.

Dentro de la industria manufacturera, las fresadoras son máquinas que presentan grandes diferencias entre sí, especialmente en lo que se refiere a la cinemática y la forma de accionamiento. Otras diferencias tienen que ver con el número de ejes, los distintos almacenes y cambiadores de herramientas, interruptores de fin de carrera por software o la conexión de dispositivos de verificación de herramientas. Por lo anterior es fácil deducir

CAPÍTULO V.

PROCESO DE DESARROLLO PARA EL SISTEMA DE PREDICCIÓN DE TIEMPO DE MAQUINADO (PreTie).

Para la definición del modelo arquitectónico y la implementación de la herramienta computacional, propuesta como solución al problema de estimación de tiempos de maquinado en fresadoras de alta velocidad, se utiliza el proceso de desarrollo basado en arquitecturas explicado en el capítulo II. En este capítulo se detallan los pasos de ese proceso aplicados a la herramienta de predicción de tiempos.

5.1. CREACIÓN DEL CASO DEL NEGOCIO PARA EL SISTEMA.

El mercado objetivo del sistema creado son las industrias que emplean fresadoras de alta velocidad en sus procesos de manufactura. Entre estas se cuenta principalmente la industria automotriz y la aeronáutica.

También es importante considerar que este tipo de aplicación puede extenderse a otros dominios además del de las fresadoras e incluir otro tipo de máquinas herramienta.

Algunas de las limitantes que se enfrentaron en el desarrollo del presente trabajo fue la existencia de un sólo modelo matemático mismo que se creó a la par que el desarrollo del sistema de predicción. El modelo matemático obtenido representa el comportamiento de una fresadora de tres ejes que soporta interpolación lineal y es el único con que se cuenta actualmente.

El diseño del sistema sin embargo, tomó en consideración las posibles variaciones que pudieran darse en el dominio de aplicación, tales como el número de ejes de una fresadora que pueden ser de 3 a 5, el tipo de interpolación que puede ser no sólo lineal sino también circular, parabólica, splines o helicoidal, y el hecho de que para cada tipo de interpolación pueden definirse más de un modelo matemático y que cada eje tiene sus propios parámetros de calibración.

5.2. ELEMENTOS DEL DOMINIO Y ANÁLISIS DE REQUERIMIENTOS.

Dentro de la industria manufacturera, las fresadoras son máquinas que presentan grandes diferencias entre sí, especialmente en lo que se refiere a la cinemática y la forma de accionamiento. Otras diferencias tienen que ver con el número de ejes, los distintos almacenes y cambiadores de herramientas, interruptores de fin de carrera por software o la conexión de dispositivos de verificación de herramientas. Por lo anterior es fácil deducir

CAPÍTULO V.

PROCESO DE DESARROLLO PARA EL SISTEMA DE PREDICCIÓN DE TIEMPO DE MAQUINADO (PreTie).

Para la definición del modelo arquitectónico y la implementación de la herramienta computacional, propuesta como solución al problema de estimación de tiempos de maquinado en fresadoras de alta velocidad, se utiliza el proceso de desarrollo basado en arquitecturas explicado en el capítulo II. En este capítulo se detallan los pasos de ese proceso aplicados a la herramienta de predicción de tiempos.

5.1. CREACIÓN DEL CASO DEL NEGOCIO PARA EL SISTEMA.

El mercado objetivo del sistema creado son las industrias que emplean fresadoras de alta velocidad en sus procesos de manufactura. Entre estas se cuenta principalmente la industria automotriz y la aeronáutica.

También es importante considerar que este tipo de aplicación puede extenderse a otros dominios además del de las fresadoras e incluir otro tipo de máquinas herramienta.

Algunas de las limitantes que se enfrentaron en el desarrollo del presente trabajo fue la existencia de un sólo modelo matemático mismo que se creó a la par que el desarrollo del sistema de predicción. El modelo matemático obtenido representa el comportamiento de una fresadora de tres ejes que soporta interpolación lineal y es el único con que se cuenta actualmente.

El diseño del sistema sin embargo, tomó en consideración las posibles variaciones que pudieran darse en el dominio de aplicación, tales como el número de ejes de una fresadora que pueden ser de 3 a 5, el tipo de interpolación que puede ser no sólo lineal sino también circular, parabólica, splines o helicoidal, y el hecho de que para cada tipo de interpolación pueden definirse más de un modelo matemático y que cada eje tiene sus propios parámetros de calibración.

5.2. ELEMENTOS DEL DOMINIO Y ANÁLISIS DE REQUERIMIENTOS.

Dentro de la industria manufacturera, las fresadoras son máquinas que presentan grandes diferencias entre sí, especialmente en lo que se refiere a la cinemática y la forma de accionamiento. Otras diferencias tienen que ver con el número de ejes, los distintos almacenes y cambiadores de herramientas, interruptores de fin de carrera por software o la conexión de dispositivos de verificación de herramientas. Por lo anterior es fácil deducir

que se requiere que cada tipo de máquina tenga un software de control especial que vaya de acuerdo con las características que ésta presenta.

Sucede lo mismo con el sistema de predicción de tiempo de maquinado. Puesto que cada tipo de interpolación tiene un manejo matemático diferente y una misma fresadora puede soportar varios tipos de interpolación simultáneamente, el sistema PreTie necesita contar con los modelos matemáticos necesarios para enfrentar la predicción de tiempo de manera correcta. Más aún, un mismo modelo matemático puede definirse de diferentes maneras usando parámetros distintos, con lo cual la variabilidad dentro del sistema se hace aún mayor.

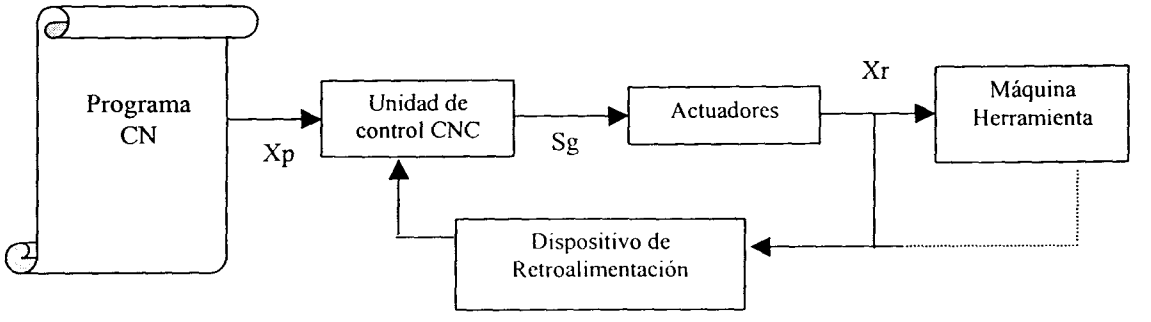
5.2.1. Elementos Del Dominio de Aplicación.

a) Proceso de Manufactura

En el control de una maquina herramienta mediante CNC, existen 5 componentes básicos (Figura 5.1):

1. **El programa de control numérico** el cual proporciona los detalles de los movimientos que realizará la máquina herramienta.

2. **La unidad de control CNC** la cual interpreta las ordenes del programa y las convierte en señales de control para los actuadores de la máquina herramienta. Este es el punto donde se compara la señal de referencia del programa con la posición real obtenida.



Xp	Orden de Posición	Sg	Señal de movimiento
Xr	Posición Real	.....	Lazo cerrado independiente
——	Lazo cerrado dependiente		

Figura 5.1 Componentes Básicos del control de una MH.

3. **Actuadores**. Son los dispositivos que ejecutan las señales de control y las convierten en acciones mecánicas para mover los mecanismos de la máquina herramienta.

4. **Elementos de retroalimentación**. Son los instrumentos de medición que proporcionan a la unidad de control la posición real de la máquina. El dispositivo de retroalimentación puede sensar la salida del actuador (lazo cerrado directo) o el desplazamiento lineal de los ejes de la máquina (lazo cerrado indirecto)

Finalmente, existe un elemento de retroalimentación en forma de instrumentos de medición, que permite comparar la posición actual de las herramientas con la posición teórica en que debería encontrarse y realizar los ajustes necesarios.

Otro elemento importante son los sistemas de CAD/CAM que facilitan la programación de las máquinas herramientas, puesto que producen de manera automática programas que contienen instrucciones de movimiento (archivos de posición) [Kief; 1988] que indican la trayectoria que deberá seguir la herramienta de corte durante el maquinado de partes.

El código generado es alimentado a la unidad de CNC la cual interpreta cada una de las instrucciones y manipula de una manera exacta, precisa y rápida, los actuadores pertinentes y se generan los movimientos en las diferentes partes de las máquinas.

b) Estimación del Tiempo de Maquinado

Como ya se ha mencionado, los sistemas CAD/CAM actuales calculan el tiempo que una máquina herramienta tarda en maquinar una pieza sin tomar en consideración aspectos físicos del proceso mismo de maquinado. Para un programa de CN se considera únicamente la distancia total que recorre la herramienta de corte de un punto a otro, la velocidad programada de avance, y los datos suministrados por el fabricante de la máquina-herramienta sobre la velocidad máxima alcanzada por ella [Rodríguez; 1999].

También se ha dicho que la diferencia entre el tiempo calculado de esta manera y el tiempo real de maquinado de una pieza, puede ser significativo y que esta diferencia se acentúa en condiciones de altas velocidades de avance.

Una forma más aproximada de calcular el tiempo de maquinado considera la disminución de la velocidad de la herramienta de corte al acercarse a un ángulo de esquina, además del tipo de interpolación que se emplea para la definición de la geometría de la pieza.

Para cada tipo de interpolación es posible establecer uno o más modelos matemáticos diferentes que definen el comportamiento de la máquina herramienta durante el proceso de maquinado de una pieza. Cada modelo involucra un conjunto diferente de parámetros.

Para efectos del presente trabajo se desarrolló en el área de manufactura del Instituto Tecnológico y de Estudios Superiores de Monterrey (ITESM) un modelo que define el comportamiento de una fresadora de alta velocidad que soporta una interpolación lineal.

c) Modelo Matemático Para una Interpolación Lineal

El modelo matemático toma en consideración los siguiente factores:

Velocidad de avance programado ($V_{f\text{-}prog}$): Es la velocidad nominal de la máquina, indicada en el programa de control numérico.

Geometría de la ruta que sigue la herramienta: Esto determina el número de cambios en la ruta que sigue la herramienta de corte y la longitud de cada segmento.

Aceleración de la máquina herramienta (a_m): Esto determina la capacidad de acelerar o desacelerar que tiene la máquina para alcanzar la velocidad de avance nominal ($V_{f\text{-}prog}$).

Ángulos y avance de esquina: La figura 5.2 muestra los ángulos de esquina (θ) para un conjunto de interpolaciones lineales. En cada intersección existe una velocidad de avance ($V_{f\text{-}esq}$) diferente que está determinada por el ángulo dado. Conforme la herramienta de corte se acerca a intersección dentro de su trayectoria debe modificar su velocidad de avance a fin de poder cambiar su dirección. Una vez que la dirección a cambiado se inicia de nuevo el proceso de aceleración hasta que la herramienta alcanza la velocidad nominal dentro del siguiente segmento de la trayectoria.

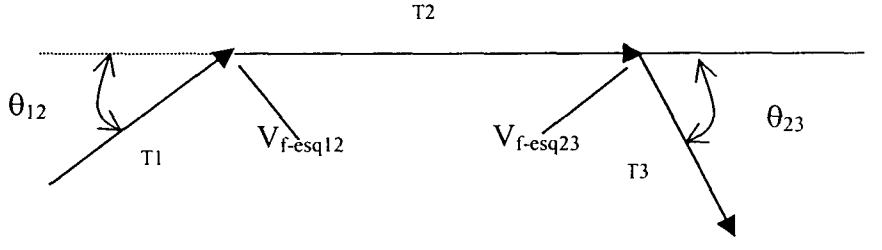


Fig. 5.2 Ángulos y avances de esquina.

La estimación del tiempo de maquinado se basa en la razón de aceleración, la razón de avance programado, los avances de esquina y la longitud de los segmentos de la trayectoria[4]. Además, se asume que un cambio en la trayectoria de la herramienta, con interpolaciones lineales, está acompañado de una pequeña trayectoria circular (Figura 5.3) cuyo radio (r_{esq}) está controlado por la tolerancia de la esquina (e) de acuerdo a la siguiente ecuación:

[4] Ver Anexo I.

$$r_{esq} = e\left\{\frac{\cos(\theta_{esq}/2)}{1-\cos(\theta_{esq}/2)}\right\} \qquad (1)$$

La máxima velocidad de avance que se logra con el radio r_{esq} y se calcula entonces con la siguiente ecuación:

$$V_{f-esq} = \sqrt{a_m r_{esq}} \qquad (2)$$

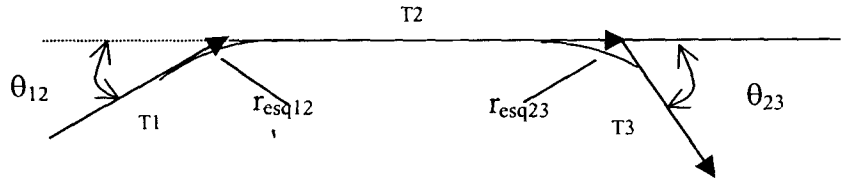


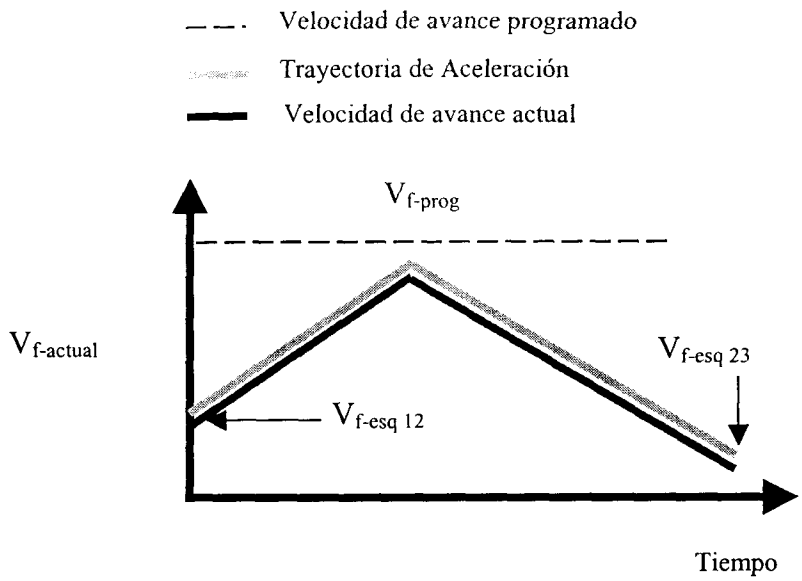
Fig. 5.3 Definición de radio de esquina

Existen tres casos que se toman en cuenta dentro del modelo y se clasifican como A, B y C:

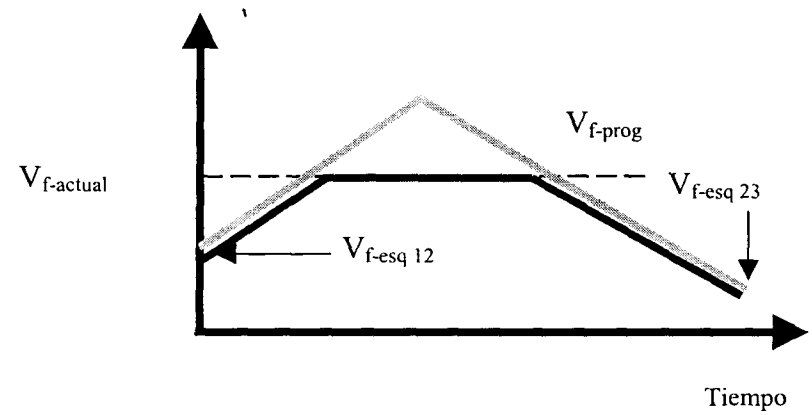
Caso tipo A: Este caso se presenta cuando la velocidad programada de avance ($V_{f\text{-}prog}$) es mayor que la velocidad máxima teórica de la máquina ($V_{f\text{-}actual}$), la cual depende del ángulo de la pieza. (Fig. 5.4 (a))

Caso tipo B: Cuando la herramienta de corte no alcanza la velocidad máxima de la máquina (Fig. 5.4 (b)) y,

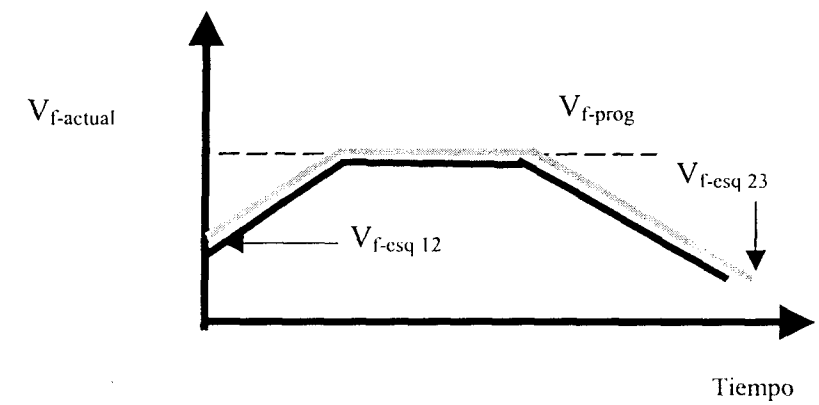
Caso tipo C: Cuando la herramienta de corte alcanza la velocidad máxima de la máquina (Fig. 5.4 (c))



(a) Velocidad programada mayor que velocidad actual



(b) No se alcanza la velocidad máxima de la máquina



(c) Se alcanza la velocidad máxima de la máquina.

Figura 5.4 Casos analizados en el modelo de interpolación lineal [Rodríguez; 1999].

d) Datos de calibración de la MH

En máquinas fresadoras son útiles hasta 5 ejes simultáneos: XYZ para determinar la aproximación al punto final en el espacio y otros dos (A y B) de desplazamientos adicionales de giro para la determinación de la posición en el espacio del eje de fresado.

Cada modelo matemático involucra un conjunto de parámetros específicos que le permiten efectuar los cálculos involucrados en la estimación del tiempo de maquinado. Un tipo de modelo tiene un conjunto constante de parámetros es decir, los parámetros están relacionados directamente con el modelo de una manera conceptual.

Estos parámetros también están relacionados con los ejes de la fresadora. Así tenemos que por ejemplo, la velocidad del eje X es un parámetro del modelo matemático que se esta usando, para la interpolación lineal.

En una máquina fresadora el movimiento que describe cada uno de sus ejes puede ser representado por un modelo matemático diferente y por lo tanto, cada eje puede estar involucrado con un conjunto de parámetros diferentes que dependen del modelo matemático que se esté usando.

A estos parámetros se les conoce como *parámetros o datos de calibración* y son obtenidos del usuario como entrada del sistema y alimentados al modelo matemático para su ejecución.

e) Archivo estandarizado de instrucciones de movimiento.

Para la estimación del tiempo de maquinado se utiliza el programa de CN generado por el sistema CAD/CAM bajo los siguientes supuestos:

- El programa de CN describe la trayectoria que sigue la herramienta de corte al definir la geometría de la pieza.

- Existe un modelo matemático que describe el comportamiento de la fresadora en cuanto a la velocidad real de la herramienta de corte en los diferentes puntos de la trayectoria.

- Se dispone del conjunto de parámetros para la fresadora particular que son utilizados dentro del modelo matemático para la realización del cálculo.

Tomando en consideración lo anterior, es posible predecir el tiempo de maquinado a partir del programa de CN sin necesidad de maquinar la pieza, lo cual involucraría un alto costo.

Para ello, es necesario filtrar la información contenida en el archivo generado por el sistema CAD/CAM (programa CN) de modo que contenga solamente aquellas

instrucciones que definan movimientos dentro de la trayectoria de corte, así como los datos necesarios para realizar el cálculo tales como las coordenadas a que llegan los ejes en una instrucción de movimiento o la velocidad que alcanza la máquina en un movimiento de corte de la pieza.

Las instrucciones del programa CN se filtran mediante un preprocesador para obtener un archivo de formato estándar que, en conjunto con el modelo matemático y los parámetros ingresados por el usuario, efectúa el cálculo correspondiente. La definición del preprocesador es similar a la de un compilador, la diferencia entre ellos radica en que un preprocesador no produce código ejecutable.

f) Potencial para incluir nuevos modelos matemáticos

A pesar de que actualmente sólo se cuenta con un modelo matemático que representa el comportamiento de una máquina fresadora con tres ejes que soporta interpolación lineal, existe la posibilidad de agregar nuevos modelos a la librería de componentes del sistema, mismos que pueden representar otros tipos de interpolación y máquinas de más de tres ejes así como otras versiones de la estimación de tiempo en cada una de las interpolaciones.

Al considerar otros tipos de interpolación se debe tomar en cuenta que los movimientos de corte no se definen en un eje sino en un plano delimitado por dos ejes .

Así, tenemos que para una interpolación lineal el movimiento de los diferentes ejes de la máquina se da de acuerdo a las coordenadas y puede ser en cualquiera de los ejes o bien, en una combinación de estos.

En cambio, en las interpolaciones circular, parabólica, splines y helicoidal, los movimientos se dan por planos puesto que por ejemplo, un círculo necesariamente necesita de 2 ejes para describirse.

5.2.2. Análisis de Características del Dominio.

El análisis de características del Dominio de las fresadoras para la generación de sistemas de PreTie se describe de acuerdo a los siguientes puntos (Figura 5.5) [Czarnecki; 1999]:

- Todo sistema de predicción de tiempos de maquinado (PreTie) debe tener un proceso de estandarización de los datos que se incluyen en las instrucciones de movimiento generadas por el sistema CAD/CAM. Además debe incluir un componente básico que define los procesos que ejecuta la máquina fresadora en la estimación del tiempo.

- Una máquina fresadora puede soportar uno o más tipos de interpolación diferentes entre los cuales pueden estar la interpolación lineal (para la cual se cuenta con el modelo matemático), circular, parabólica, splines y helicoidal.

- Cualquier máquina fresadora soporta 3 ejes como mínimo y 5 como máximo. Cualquier tipo de interpolación que se vaya a usar en el sistema PreTie debe tomar en cuenta esto y tener una configuración mínima de 3 ejes.

- Para cada eje de la fresadora se aplica un modelo matemático que represente su comportamiento. El modelo puede ser el mismo para todos los ejes o bien a cada eje puede aplicársele un modelo diferente.

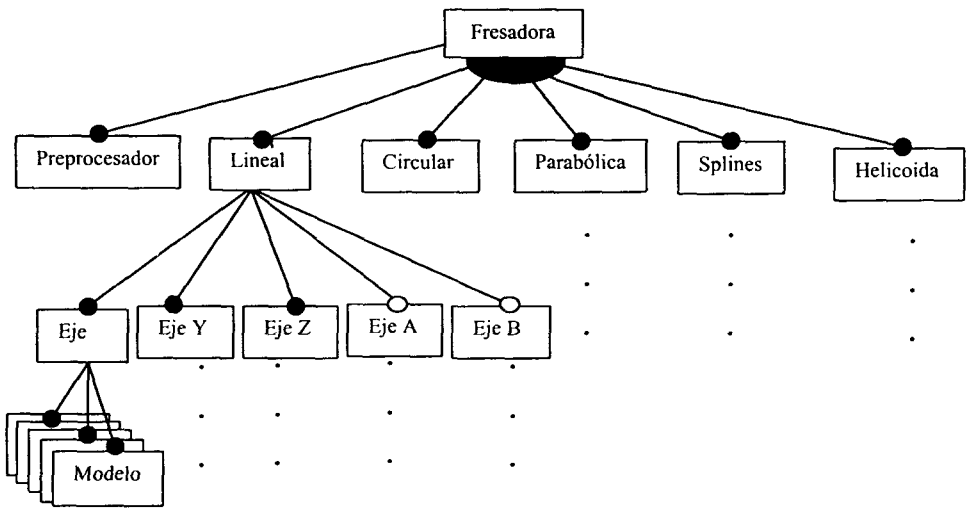


Figura 5.5 Diagrama de Características del Dominio de las Fresadora Para Generación de Sistemas de PreTie con Diferentes Configuraciones.

Como puede observarse, esta definición describe una gran cantidad de PreTies diferentes. Así, podemos tener una máquina de 4 ejes que soporta las interpolación circular y lineal y que para cada eje usa un modelo matemático diferente, por ejemplo el 1, 2, 3 y 4. O bien, podemos tener otra máquina con la misma configuración pero que usa solamente un modelo para todos sus ejes.

Esto nos lleva a pensar en los sistema de PreTie como en una familia de productos a la cual se le puede aplicar la programación generativa para automatizar el proceso de creación de nuevos miembros de la familia, a partir de definiciones abstractas de las características de la máquina fresadora que será el cliente del sistema.

Continuando con el análisis del domino, las principales funcionalidades y componentes identificados dentro del diagrama de características son:

Preprocesador	Máquina Fresadora	Interpolación	Ejes	Modelos
Preprocesador	Máquina Base	Lineal	Eje X	Modelo1
		Circular	Eje Y	Modelo2
		Parabólica	Eje Z	Modelo 3....
		Esplines	Eje A	
		Helicoidal	Eje B	

El preprocesador, como ya se menciono anteriormente, es el encargado de estandarizar el archivo APT generado por los sistemas CAD/CAM, en un formato que permita obtener solamente la información relativa a los movimientos que se efectúan para maquinar la pieza.

La Figura 5.6. muestra la dependencia de usos entre las categorías de componentes de acuerdo al análisis de dominio propuesto:

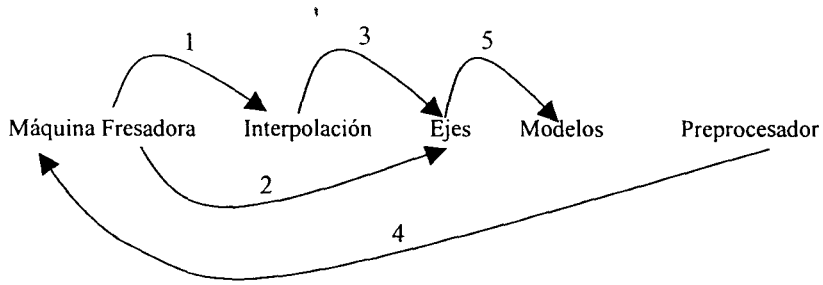


Figura 5.6 Dependencia de Usos Entre las Categorías de Componentes.

Una máquina fresadora usa Interpolación (1), Ejes(2) y Preprocesador(4), Interpolación usa Ejes (3) y Ejes usa Modelos (5).

La máquina base es un elemento fijo dentro de cualquier sistema de PreTie. Este esquema deriva fácilmente en una arquitectura por capas como se muestra en la Figura 5.7.

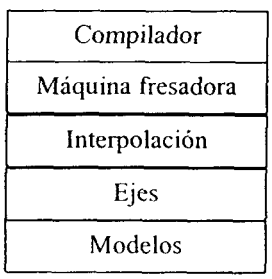


Figura 5.7 Arquitectura por Capas de un Sistema de PreTie.

En la siguiente sección se analiza cómo esta arquitectura se adapta bien a las necesidades del sistema y más aún, a las necesidades de la familia de productos de PreTies.

5.3. DISCUSIÓN DE LA ARQUITECTURA.

En esta etapa del proceso de desarrollo basado en arquitecturas se analizaron los diferentes estilos arquitectónicos mencionados dentro del marco teórico. Se tomaron en cuenta sus características, sus ventajas y desventajas, y la manera como los componentes y requerimientos del sistema podían beneficiarse de ellas.

Básicamente, los elementos y requerimientos identificados dentro del sistema estimador de tiempos fueron:

- Fresadoras de alta velocidad con características diferentes entre sí.
- Interpoladores diferentes.
- Un modelo matemático para cada eje/plano de la máquina fresadora y para cada interpolador soportado.
- Programas de CNC en formato del lenguaje APT
- El Preprocesador que permite la estandarización de los datos generados en el CNC.
- Datos de calibración.
- Una interfase de captura de los datos de calibración.
- Un estimador para cada tipo de fresadora.
- Un generador de estimadores.

Partiendo de lo más general a lo más particular, se analizaron los estilos arquitectónicos como a continuación se detalla:

a) Puesto que lo que se pretende es crear un generador de sistemas estimadores de tiempos de maquinado que se apliquen al dominio de fresadoras de alta velocidad, conviene utilizar las arquitecturas de software de dominio específico (DSSA) ya que, como se explicó en el capítulo II, éstas tienen como precepto el uso de generadores para crear sistemas intercambiado componentes tomados de una librería. En este caso particular cada modelo matemático, tomado como un componente, es factible de ser intercambiable lo mismo que los diferentes tipos de interpolación soportados por la fresadora. Dado que los datos de calibración están íntimamente relacionados con el modelo, también son susceptibles de intercambiarse al cambiar el modelo, sin que esto afecte la funcionalidad del sistema generado y sobre todo, sin que se afecte su estructura.

En la siguiente sección se analiza cómo esta arquitectura se adapta bien a las necesidades del sistema y más aún, a las necesidades de la familia de productos de PreTies.

5.3. DISCUSIÓN DE LA ARQUITECTURA.

En esta etapa del proceso de desarrollo basado en arquitecturas se analizaron los diferentes estilos arquitectónicos mencionados dentro del marco teórico. Se tomaron en cuenta sus características, sus ventajas y desventajas, y la manera como los componentes y requerimientos del sistema podían beneficiarse de ellas.

Básicamente, los elementos y requerimientos identificados dentro del sistema estimador de tiempos fueron:

- Fresadoras de alta velocidad con características diferentes entre sí.
- Interpoladores diferentes.
- Un modelo matemático para cada eje/plano de la máquina fresadora y para cada interpolador soportado.
- Programas de CNC en formato del lenguaje APT
- El Preprocesador que permite la estandarización de los datos generados en el CNC.
- Datos de calibración.
- Una interfase de captura de los datos de calibración.
- Un estimador para cada tipo de fresadora.
- Un generador de estimadores.

Partiendo de lo más general a lo más particular, se analizaron los estilos arquitectónicos como a continuación se detalla:

a) Puesto que lo que se pretende es crear un generador de sistemas estimadores de tiempos de maquinado que se apliquen al dominio de fresadoras de alta velocidad, conviene utilizar las arquitecturas de software de dominio específico (DSSA) ya que, como se explicó en el capítulo II, éstas tienen como precepto el uso de generadores para crear sistemas intercambiado componentes tomados de una librería. En este caso particular cada modelo matemático, tomado como un componente, es factible de ser intercambiable lo mismo que los diferentes tipos de interpolación soportados por la fresadora. Dado que los datos de calibración están íntimamente relacionados con el modelo, también son susceptibles de intercambiarse al cambiar el modelo, sin que esto afecte la funcionalidad del sistema generado y sobre todo, sin que se afecte su estructura.

Con esto estamos implementando una arquitectura de línea de productos para el dominio de las fresadoras de alta velocidad

b) Como vimos en el punto 5.2.2. el análisis del dominio nos llevó de una manera natural a definir una arquitectura en capas que además nos permite aplicar un generador de software para la selección, ensamblaje y automatización en la creación de nuevos miembros de la familia de productos.

Sin embargo, el modelo de arquitectura que ahí se muestra tiene algunos aspecto que hay que considerar a la luz del proceso que se sigue en la estimación del tiempo de maquinado, el cual se da de la siguiente manera:

- Se define una estructura del sistema en la cual se contempla el número de ejes de que está compuesta una fresadora y los tipos de interpolación que soporta. Suponga que se tiene una máquina fresadora de 3 ejes que soporta interpolación parabólica. De acuerdo con la arquitectura antes mencionada la estructura quedaría como se muestra en la Figura 5.8.

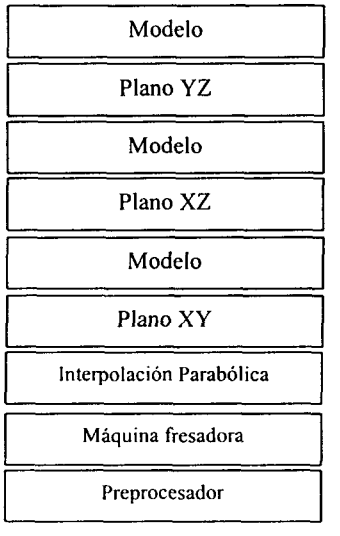


Figura 5.8. Estructura de un PreTie que soporta 3 ejes e interpolación parabólica.

- Se toma un registro del archivo estandarizado de instrucciones de movimiento que genera el compilador.

- Se analiza el tipo de interpolación a que se refiere la instrucción (lineal, circular, parabólica, etc.).

- Una vez definido el tipo de interpolación se establece en qué ejes o planos se da el movimiento.

- El proceso de cálculo de tiempo de maquinado se ejecuta para cada eje que se mueve. Este proceso se manda llamar dentro de la estructura. Esto significa que el programa ejecuta todos los procesos de cálculo que encuentra dentro de la estructura definida (Figura 5.8).

Como puede observarse de la Figura 5.8, la búsqueda de los proceso de cálculo adecuados para esa instrucción de movimiento, se da en todos y cada uno de los ejes o planos involucrados en la estructura descartando aquellos que no corresponden a los ejes en movimiento ni al tipo de interpolación. Si existen más interpolaciones esta búsqueda se hace mucho más larga.

Lo ideal es que si ya se conoce el tipo de interpolación y el eje que se mueve, el proceso se dirija directamente a la capa que corresponde a esa interpolación y ya dentro se dirija al eje correspondiente. Una pequeña modificación a la estructura ayuda a hacer el sistema más claro y consistente en ese sentido.

Un eje o plano dado, relacionado con un tipo de interpolación específico no tiene por que relacionarse con otro tipo de interpolación durante el proceso de cálculo de una instrucción de movimiento. Entonces se hace evidente la necesidad de representar esta relación modelo/eje/interpolación en una estructura aparte.

La figura 5.9 muestra la arquitectura modificada de modo que un eje o plano sólo tiene relación con la interpolación que lo maneja.

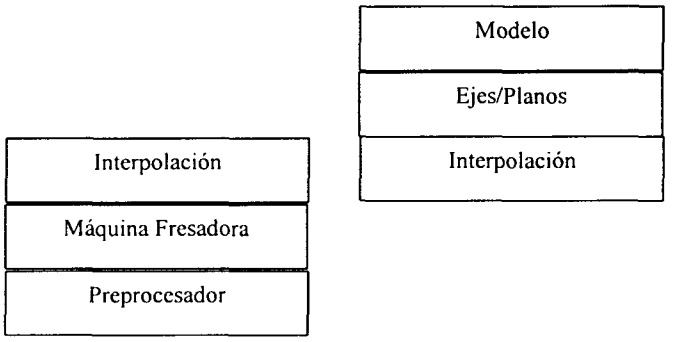


Figura 5.9 Arquitectura Modificada Para el Sistema de PreTie.

Esta nueva definición de la arquitectura implica menos búsquedas de los procesos de cálculo correspondientes y nos muestra de manera más clara, que un tipo de interpolación se relaciona a un conjunto de ejes o planos, cada uno de los cuales se procesa por medio de un modelo específico. Un modelo matemático para una interpolación lineal necesariamente resultará muy diferente de un modelo de interpolación circular por ejemplo, y en este sentido es que cabe la diferenciación conceptual de un plano o eje para un tipo de interpolación con respecto a otro plano o eje para otro tipo de interpolación.

La Figura 5.10 muestra la estructura de un sistema PreTie basado en la arquitectura definida y que se compone de 3 ejes con capacidad de procesamiento para interpolación lineal y circular.

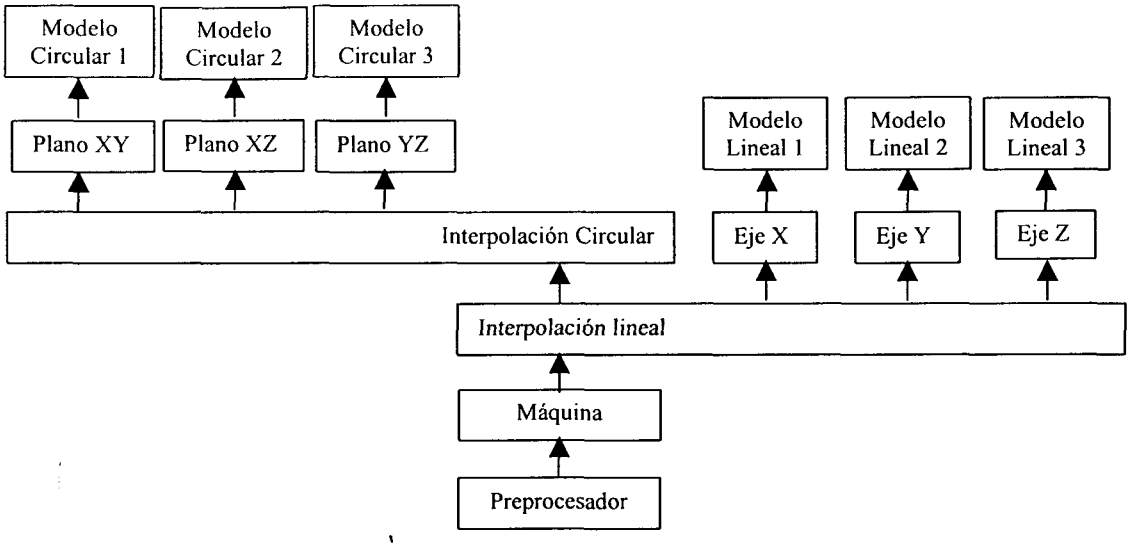


Figura 5.10 PreTie de una Fresadora con 3 ejes e Interpolaciones circular y lineal.

Como puede observarse se tiene una estructura base en la cual están incluidas todas las interpolaciones que soporta la fresadora. Además, cada tipo de interpolación incluye subestructuras para cada eje o plano.

Bajo esta estructura es relativamente fácil agregar tipos de interpolación dado que sólo implica agregar una nueva capa a la estructura principal (la que corresponde a la nueva interpolación) con sus respectivos ejes.

5.4. IMPLEMENTACIÓN DE LA ARQUITECTURA.

En los puntos anteriores de la metodología que se siguió, ya se hacen algunas representaciones de lo que es la arquitectura propuesta. En este apartado se le da formalidad a esa representación a través del uso de UML, con diagramas que reflejan diferentes vistas del sistema.

5.4.1. Diagrama De Clases.

Para soportar las variaciones existentes en el sistema se hace uso de la característica de herencia soportada por los lenguajes orientados a objetos como C++ misma que en los diagramas de clases de UML se representa como una generalización / especialización.

La Figura 5.10 muestra la estructura de un sistema PreTie basado en la arquitectura definida y que se compone de 3 ejes con capacidad de procesamiento para interpolación lineal y circular.

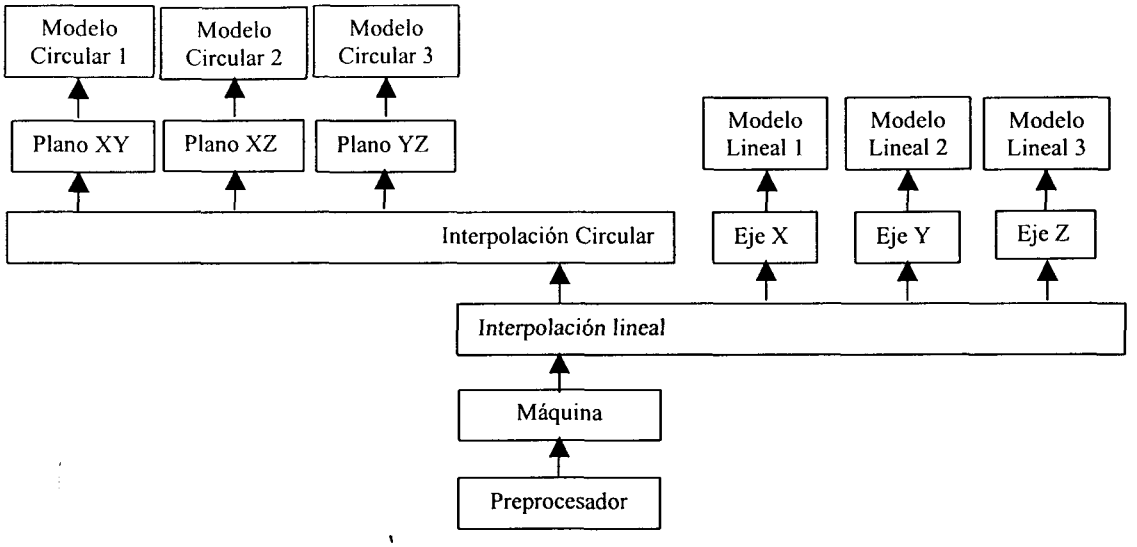


Figura 5.10 PreTie de una Fresadora con 3 ejes e Interpolaciones circular y lineal.

Como puede observarse se tiene una estructura base en la cual están incluidas todas las interpolaciones que soporta la fresadora. Además, cada tipo de interpolación incluye subestructuras para cada eje o plano.

Bajo esta estructura es relativamente fácil agregar tipos de interpolación dado que sólo implica agregar una nueva capa a la estructura principal (la que corresponde a la nueva interpolación) con sus respectivos ejes.

5.4. IMPLEMENTACIÓN DE LA ARQUITECTURA.

En los puntos anteriores de la metodología que se siguió, ya se hacen algunas representaciones de lo que es la arquitectura propuesta. En este apartado se le da formalidad a esa representación a través del uso de UML, con diagramas que reflejan diferentes vistas del sistema.

5.4.1. Diagrama De Clases.

Para soportar las variaciones existentes en el sistema se hace uso de la característica de herencia soportada por los lenguajes orientados a objetos como C++ misma que en los diagramas de clases de UML se representa como una generalización / especialización.

La herencia es un mecanismo mediante el cual se puede derivar una clase de otra. La clase derivada o descendiente hereda los datos miembros y métodos de la clase ascendiente o base. La derivaciónde una clase refina a la clase ascendiente añadiendo nuevos atributos y operaciones. Además, la clase derivada puede sobreponer los métodos heredados cuando sus operaciones no sean adecuadas para la clase derivada [Arnush; 1996].

El diagrama de clases muestra el conjunto de clases del sistema y sus relaciones. Es un aspecto estático de los bloques de construcción que compone al sistema y nos muestra algunos detalles de cada bloque (Figura 5.11).

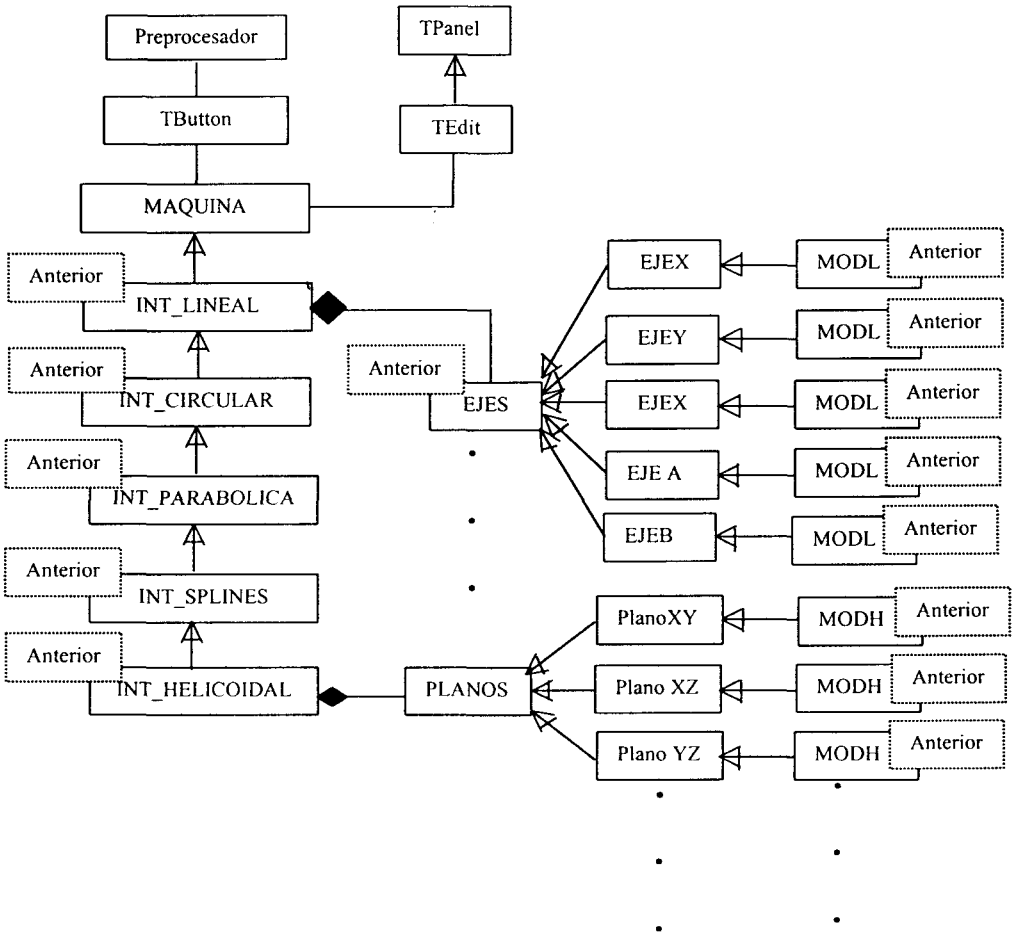


Figura 5.11 Diagrama de Clases de Sistema de PreTie

Este diagrama es una representación general de lo que sería el sistema de PreTies. En la Figura 5.12 se muestra el mismo Diagrama de clases pero para una máquina fresadora de 3 ejes que soporta interpolación lineal, mismo que fue implementado.

En este diagrama se muestran algunos detalles de las clases tales como su nombre, los datos miembro y algunos de sus métodos.

En la parte del presente trabajo que corresponde a los detalles de la implementación, se explicará detenidamente de qué manera fueron empleadas estas clases.

MAQUINA es la clase base y se relaciona con el compilador a través de un objeto Tbutton, a fin de obtener el archivo estandarizado de instrucciones de movimiento, y con las ventanas de texto donde se capturan los parámetros de la máquina.

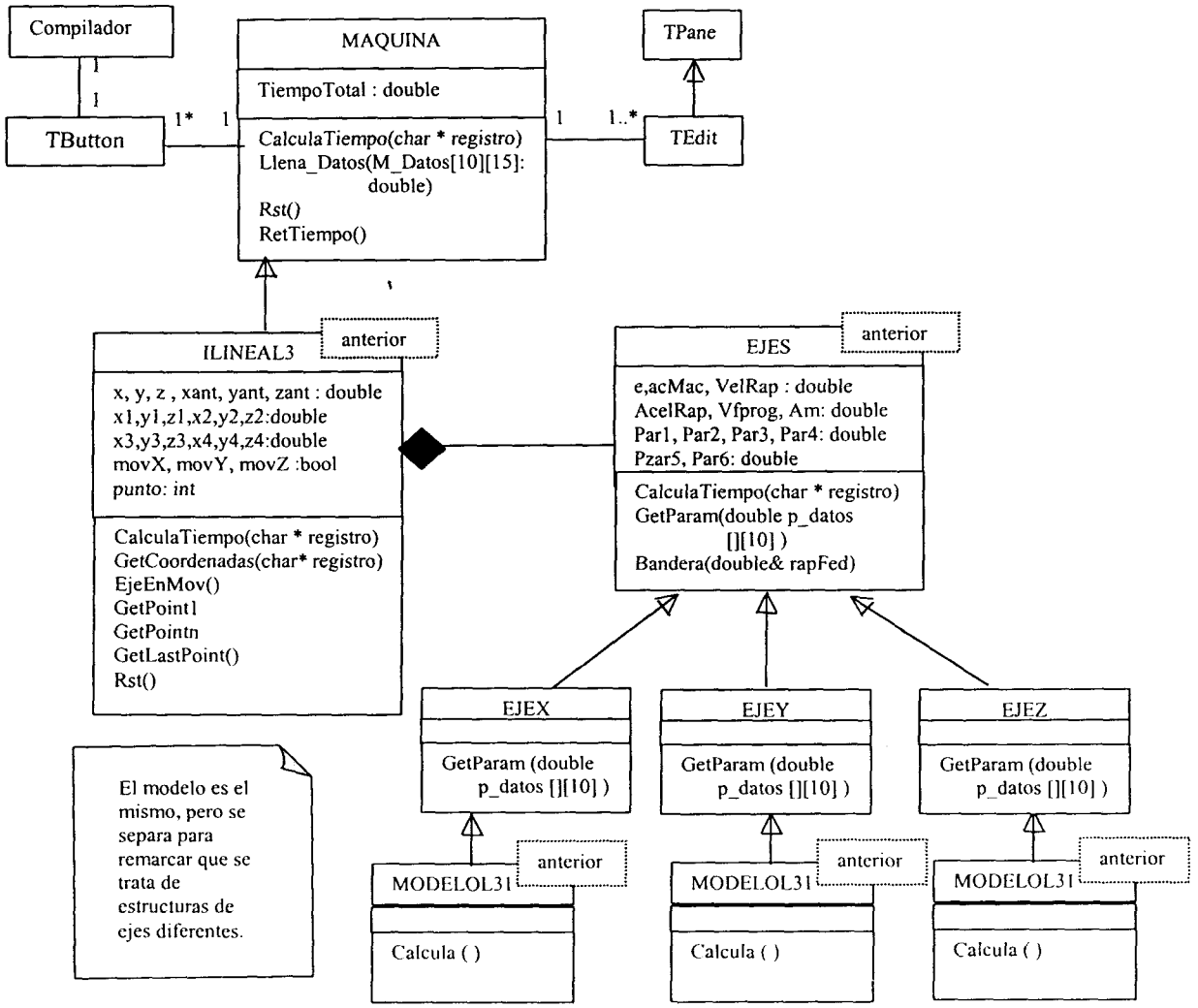


Figura 5.12. Diagrama de Clases del PreTic Implementado.

Esta clase contiene solamente un dato miembro: TiempoTotal, que es donde se va almacenado el tiempo de cada instrucción de movimiento procesada. Esta variable se distribuye a lo largo de la estructura por medio de la herencia.

ILINEAL y EJES son clases parametrizadas que toman como parámetro la clase anterior para heredar de ahí. Esta parametrización es la que le da la flexibilidad al sistema de modo que puedan intercambiarse componentes en cada una de las capas de la arquitectura, sin que esto afecte a las demás capas.

Hay que tomar en cuenta que para que esto suceda, los componentes deben tener ciertas características que les permitan insertarse o quitarse sin ningún problema en los puntos donde sea necesario para el correcto funcionamiento del sistema.

La clase EJES además, está representada como una composición en la cual ILINEAL3 representa el "todo" y EJES representa las "partes". Esta composición también indica que si quitamos la capa de interpolación lineal, la clase EJES desaparece.

Para cada eje de la fresadora se aplica un modelo matemático. Cabe recordar que sólo se cuenta con un modelo matemático, por lo cual el diagrama de clases establece el mismo para todos los ejes. Sin embargo, la representación de cada modelo se hace por separado para indicar que corresponde un modelo a cada eje.

5.4.2. Diagrama De Objetos.

En esta vista del sistema podemos apreciar las instancias de los objetos contenidos en el diagrama de clases presentado en el punto anterior (Figura 5.13).

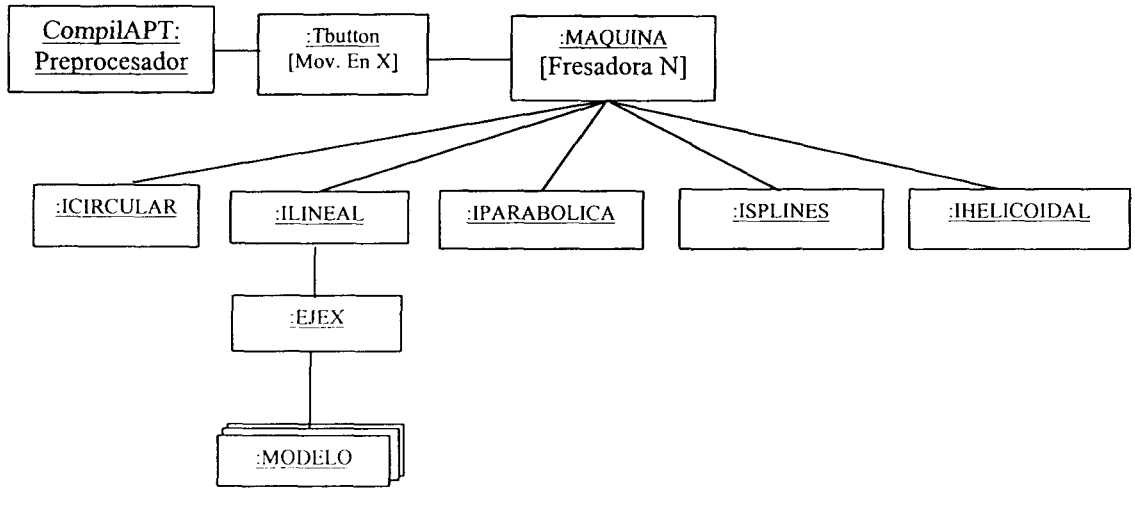


Figura 5.13 Diagrama de Objetos.

Un diagrama de objetos representa las relaciones de un conjunto de objetos en un momento dado en el tiempo [Booch; 1999].

Este diagrama nos muestra las relaciones entre objetos que se dan cuando al leer un registro del archivo estandarizado de instrucciones de movimiento de la fresadora, éste define un movimiento del eje X para una interpolación lineal.

La relación se da desde la instancia de la clase MAQUINA, hacia las instancias de ILINEAL, EJES y MODELOS.

Si el eje en movimiento fuera el Z la relación en ese instante se daría hacía el eje Z con su respectivo modelo matemático

5.4.3. Diagrama De Colaboración.

Esta vista del sistema presenta un aspecto dinámico del mismo ya que muestra las interacciones que se dan entre los objetos: los mensajes que se trasmiten entre sí y la secuencia en que se dan dichos mensajes. Esta colaboración entre objetos es lo que define la secuencia de ejecución de los diferentes procesos del sistema (Figura 5.14)

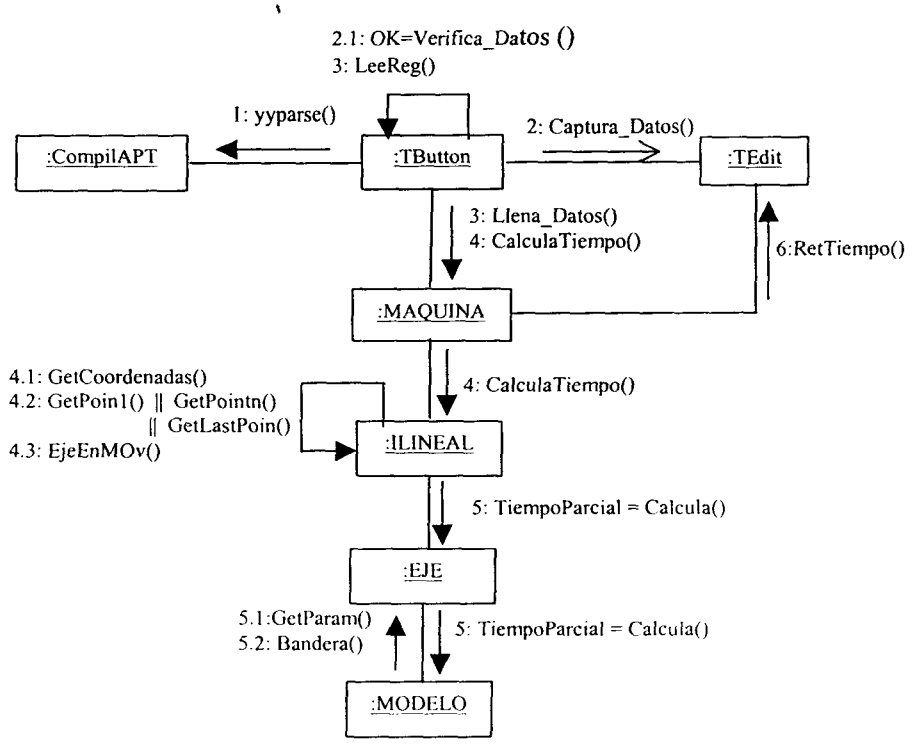


Figura 5.14 Diagrama de Colaboración

Existe un objeto Tbutton a través del cual se manda llamar al preprocesador. Este mensaje va acompañado del nombre del archivo APT que se va a filtrar. Posteriormente se envía un mensaje de Llena_Datos() al objeto MAQUINA a fin de realizar la captura de los datos que corresponden a los parámetros para cada uno de los ejes de la fresadora.

El orden de estos dos procesos puede intercambiarse ya que de cualquier manera, en el paso siguiente, donde otro objeto Tbutton ejecuta un proceso de verificación de datos, si uno de los procesos no se ha ejecutado no puede continuarse con el siguiente proceso.

Una vez que los datos están correctos, Tbutton lee un registro del archivo estandarizado de instrucciones de movimientos y le pide a MAQUINA que efectúe el cálculo del tiempo para ese registro.

El mensaje CalculaTiempo(registro) pasa a través de la estructura definida verificando en cada una de las capas si ésta contiene el proceso que corresponde con el tipo de interpolación que viene especificado dentro del registro leído, hasta que la encuentra. En caso de ser una instrucción que no contiene ningún tipo de interpolación o que esa interpolación no esta soportada por la fresadora, simplemente le agrega un 0 al tiempo total.

Para el PreTie implementado, la única capa es la de la interpolación lineal y es la que se muestra en el diagrama de la Figura 5.14. Entonces, cuando el movimiento es lineal, ILINEAL3 ejecuta los procesos para obtener las coordenadas del movimiento y en base a éstas, el nuevo punto donde se van a ubicar los ejes de la fresadora y cuáles ejes se van a mover para llegar a ese punto.

Una vez hecho esto, manda llamar al proceso Calcula() del modelo matemático que se usa para el eje en movimiento. Este mensaje se transfiere a través de EJE hacia modelo gracias a que modelo hereda de eje (Figura 5.12). El modelo le envía un mensaje GetParam() al eje para obtener los datos de calibración que corresponden a ese eje y en base a esos datos, a través del mensaje Bandera() obtiene los valores de los parámetros que va a utilizar para sus cálculos.

Calcula() regresa el tiempo que tarda el movimiento para ese eje.

Una vez procesados todos los registros del archivo estandarizado de instrucciones de movimiento, el resultado se envía a un objeto Tedit para mostrarse en pantalla.

5.5. IMPLEMENTACIÓN DEL SISTEMA PreTie.

Como ya lo vimos anteriormente, los componentes del sistema son el preprocesador, el archivo estandarizado de instrucciones de movimiento, el o los modelos matemáticos, los interpoladores y el controlador de la secuencia de los procesos involucrados en la predicción del tiempo de fresado. Para efectos de la implementación se agregó un nuevo componente: un archivo de configuración.

A continuación se analizará la manera como se implentaron cada una de las partes que componen el sistema completo.

El orden de estos dos procesos puede intercambiarse ya que de cualquier manera, en el paso siguiente, donde otro objeto Tbutton ejecuta un proceso de verificación de datos, si uno de los procesos no se ha ejecutado no puede continuarse con el siguiente proceso.

Una vez que los datos están correctos, Tbutton lee un registro del archivo estandarizado de instrucciones de movimientos y le pide a MAQUINA que efectúe el cálculo del tiempo para ese registro.

El mensaje CalculaTiempo(registro) pasa a través de la estructura definida verificando en cada una de las capas si ésta contiene el proceso que corresponde con el tipo de interpolación que viene especificado dentro del registro leído, hasta que la encuentra. En caso de ser una instrucción que no contiene ningún tipo de interpolación o que esa interpolación no esta soportada por la fresadora, simplemente le agrega un 0 al tiempo total.

Para el PreTie implementado, la única capa es la de la interpolación lineal y es la que se muestra en el diagrama de la Figura 5.14. Entonces, cuando el movimiento es lineal, ILINEAL3 ejecuta los procesos para obtener las coordenadas del movimiento y en base a éstas, el nuevo punto donde se van a ubicar los ejes de la fresadora y cuáles ejes se van a mover para llegar a ese punto.

Una vez hecho esto, manda llamar al proceso Calcula() del modelo matemático que se usa para el eje en movimiento. Este mensaje se transfiere a través de EJE hacia modelo gracias a que modelo hereda de eje (Figura 5.12). El modelo le envía un mensaje GetParam() al eje para obtener los datos de calibración que corresponden a ese eje y en base a esos datos, a través del mensaje Bandera() obtiene los valores de los parámetros que va a utilizar para sus cálculos.

Calcula() regresa el tiempo que tarda el movimiento para ese eje.

Una vez procesados todos los registros del archivo estandarizado de instrucciones de movimiento, el resultado se envía a un objeto Tedit para mostrarse en pantalla.

5.5. IMPLEMENTACIÓN DEL SISTEMA PreTie.

Como ya lo vimos anteriormente, los componentes del sistema son el preprocesador, el archivo estandarizado de instrucciones de movimiento, el o los modelos matemáticos, los interpoladores y el controlador de la secuencia de los procesos involucrados en la predicción del tiempo de fresado. Para efectos de la implementación se agregó un nuevo componente: un archivo de configuración.

A continuación se analizará la manera como se implentaron cada una de las partes que componen el sistema completo.

5.5.1 El Preprocesador.

El compilador es el encargado de la estandarización de los datos generados por el CAD/CAM en un archivo con formato APT (La Figura 5.15)

Se tienen dos elementos: el sistema CAD/CAM y el Preprocesador. El primero define la geometría de una pieza que se va a maquinar en un programa de control numérico. Las instrucciones del programa generado se definen de acuerdo a la sintaxis del lenguaje APT.

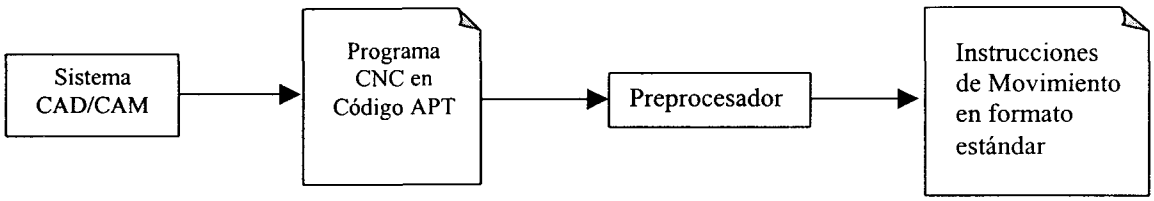


Figura 5.15 Secuencia de Flujos de Datos del Compilador

El preprocesador lee las instrucciones del programa de control numérico y toma solamente aquellas que definen movimientos de los diferentes ejes de la fresadora, eliminando todas las demás. Las instrucciones seleccionadas (las de movimiento) se guardan en un archivo con un formato estandarizado.[5]

El preprocesador sólo procesa la salida del CAD/CAM hasta que se ha generado por completo. Así mismo, el preprocesador entrega una salida completa hacia la siguiente etapa del cálculo de tiempo de maquinado.

El compilador se implementó usando los lenguajes Bisson y Yacc como generadores. En el anexo II se muestra el listado de los archivos fuente tanto para el analizador léxico como para el analizador sintáctico.

Las palabras clave del código APT[6] que se tomaron en consideración fueron:

- Palabras que especifican el tipo geométrico de una entidad definida: CIRCLE
- Palabras que especifican movimiento: GOTO, RAPID
- Palabras que especifican funciones relacionadas con el dispositivo que se está controlando: FEDRAT, SPINDL
- Palabras que definición de unidades: IPM, MMPM, RPM

El archivo de salida se denominó "Parametro.txt" e incluye los datos de las coordenadas hacia donde se desplazan los ejes de la fresadora para realizar los cortes a la pieza que se está maquinando.

[5] .Ver Anexo III.
[6] Capítulo IV 4.5 Apdo. 4.5

5.5.2 El Sistema de PreTie.

5.5.2.1 La pantalla.

El sistema consta de una sola pantalla (Figura 5.16). En la parte superior se encuentran todos aquellos componentes que permiten la manipulación de los datos que el sistema recibe como entrada, tal como los parámetros de calibración.

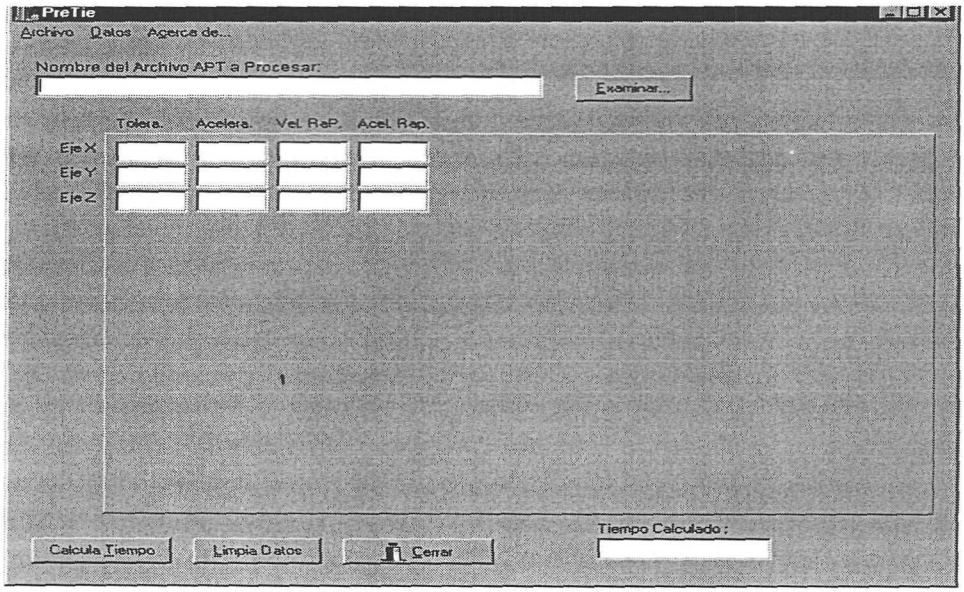


Figura 5.16 Pantalla del Sistema PreTie.

La opción Nuevo APT del menú Archivo realiza la misma función que el botón Examinar es decir, permite al usuario seleccionar un archivo APT que será filtrado y transformado al formato estándar.

El menú Datos presenta dos opciones (Figura 5.17). La primera opción (Guardar) permite grabar los datos capturados en un archivo. La segunda opción permite recuperar los datos previamente guardados. Esto facilita el manejo del sistema al momento de estar realizando pruebas ya que evita el estar capturando los mismos datos una y otra vez.

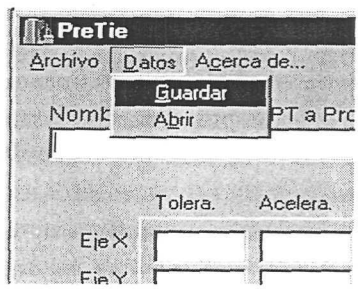


Figura 5.17. Opciones del Menú Datos.

Cabe aclarar que los datos guardados se refieren únicamente a los parámetros de calibración. El archivo compilado se considera como un dato aparte.

. La captura de parámetros de los diferentes ejes se hace a través de una matriz de ventanas de texto dentro de la misma pantalla. La matriz se define de acuerdo al número de ejes de la máquina y a los parámetros que necesitan los modelos matemático empleados. La Figura 5.18 muestra el área de captura de los datos de calibración del PreTie que fue implementado para una máquina de 3 ejes que usa interpolación lineal.

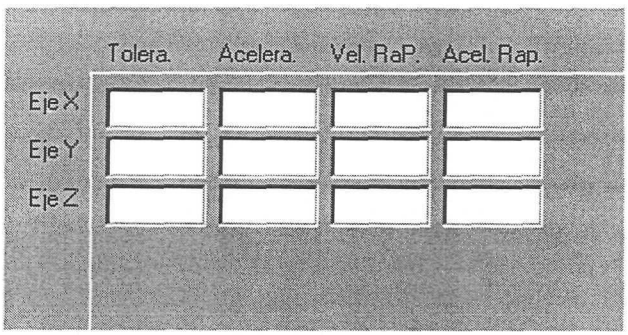


Figura 5.18. Área de Captura de Datos de Calibración

En la parte inferior de la pantalla se encuentran tres botones y un área de texto que es donde se muestra el tiempo de maquinado obtenido (Figura 5.19).

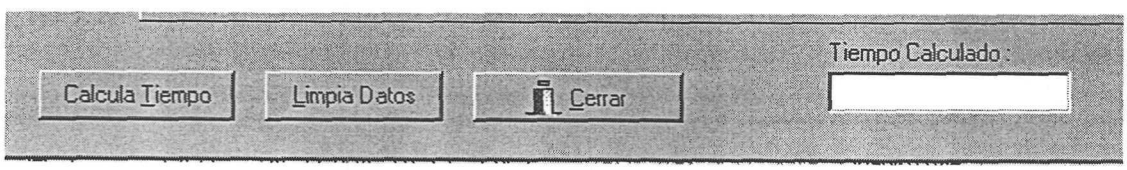


Figura 5.19 Parte Inferior de la Pantalla del Sistema PreTie

El botón Cerrar permite salir del sistema igual que la opción Salir del menú Archivo.

El botón Limpia Datos permite borrar de la pantalla todos los datos de calibración capturados.

El botón Calcula Tiempo es el que llama al proceso de estimación de tiempo y muestra el resultado.

Si los datos no están completos o hay algún error en la captura por ejemplo que en lugar de un número se escribió una letra, el programa muestra un mensaje de error (Figura 5.20).

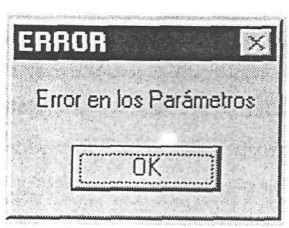


Figura 5.20 Mensaje de Error

La parte de programación que corresponde a la pantalla se realizó aprovechando las facilidades del entorno de desarrollo de aplicaciones rápido (RAD) de BBC4.

5.5.2.2 Implementación de la Arquitectura en capas.

La programación de la estimación de tiempos de maquinado se hizo utilizando la característica de datos parametrizados que soporta C++ a través de templates.

Un template es una manera de indicarle al compilador el algoritmo que debe usar para realizar una función [Arnush; 1996] y permiten construir una familia de funciones o clases relacionadas [Borland; 1999]

En el sistema existe una clase base denominada MAQUINA, la cual incluye un método llamado CalculaTiempo() que recibe un registro del archivo estandarizado de instrucciones de movimiento. Dentro de esta clase, el método CalculaTiempo() no efectúa ninguna función, únicamente sirve como soporte para la integración de nuevas interpolaciones como veremos más adelante. Este método debe existir en todas las capas de interpolaciones que se contemplen dentro de la configuración de la máquina fresadora. Más adelante se explica la razón de esto.

En el archivo de configuración llamado CONFIG.h se registra la configuración de la máquina fresadora. Por ejemplo, si la máquina soporta interpolación lineal y circular existirá una línea en el archivo como la que a continuación se muestra:

```
typedef     ILINEAL3< ICIRCULAR3 < maq > >  mh;
```

Donde *maq* es una definición de la clase MAQUINA explicada anteriormente. El nombre de la interpolación se forma con la letra I seguida del tipo de interpolación y el número de ejes para los que está definida esa interpolación.

Aquí se muestran algunos ejemplos más de definición de configuraciones para diferentes fresadoras:

```
typedef     IPARABOLICA4< ICIRCULAR4 < maq > >  mh;
typedef     IHELICOIDAL5< ISPLINES5<ICIRCULAR5 < maq > > >  mh;
typedef    ILINEAL3< ICIRCULAR3 < IPARABOLICA< ISPLINES < ILINEAL <   maq >
> > > > > mh;
```

Esta forma de definición de una máquina se hace posible gracias a los templates. Un templete recibe un parámetro sin importar su tipo, para usarlo ya sea dentro de una clase o función. Por ejemplo la siguiente función:

```
template< class T>
const T& Menor(const T& a, const T& b, const T& c)
{
    if (a<b)
   {       if (a<c) return a;  }
```

```
    else if (b<c) return b;
    return c;
}
```

podemos llamarla de las siguientes maneras:

```
int x= Menor( 1, 2, 3)
double  y=Menor(1.1,  8.2,  3.5)
```

nos regresa el valor más pequeño de los tres enteros o bien, efectúa el mismo cálculo pero recibiendo datos tipo double.

De este modo no importa que tipo de dato numérico que enviemos, siempre se realiza la operación sin generar ningún error.

Cada tipo de interpolación definida en la configuración deberá tener un método llamado CalculaTiempo() y la estructura mostrada en la Figura 5.21.

```
template <class anterior >
class nombre-del-interpolador: public anterior
{
     cuerpo de la clase;
     .
     .
     CalculaTiempo (char * registro)
       If  (registro[0]= Clave-para-el-tipo-de interpolador)
                                  Se efectúan los cálculos;
       Anterior::CalculaTiempo(registro);
};
```

Figura 5.21 Implementación General para cada Capa de Interpolación.

En el archivo estandarizado de instrucciones de movimiento cada registro tiene como primer carácter una clave que identifica el tipo de interpolación que se va a realizar[7].

Cuando se lee un registro se llama al método CalculaTiempo() de la estructura de capas definida en el archivo de configuración y empieza a recorrerla capa por capa. Esta es la razón por la cual en cada capa de interpolación debe existir este método. Como se observa en la Figura 5.21, si la clave que identifica al tipo de interpolación corresponde con ella se efectúan los cálculos y se llama a la interpolación de la capa inferior definida por el parámetro *anterior* del template. De otro modo sólo se manda llamar a la interpolación definida por *anterior*. La secuencia de llamadas al método CalculaTiempo() termina en MAQUINA.

Cuál es la interpolación anterior, o incluso si no hay otra interpolación, no es importante porque la clase está parametrizada para tomar cualquier interpolación que se defina en la composición como la anterior.

[7] Ver Anexo III

La razón de que MAQUINA tenga un método CalculaTiempo() que no efectúa ningún cálculo, es precisamente darle consistencia a la estructura.

Recordemos también que una fresadora además de distintos tipos de interpolaciones, puede tener de 3 a 5 ejes. Una vez que el programa determina la capa de interpolación donde se van a realizar los cálculos, el siguiente paso es identificar el o los ejes que se están moviendo en la instrucción leída. En base al eje que se mueve se manda llamar otra estructura que también debe definirse en el archivo de configuración.

Como se ilustra a continuación, es necesario tener una configuración para cada eje de la máquina fresadora, debido a que cada eje se relaciona con un modelo matemático específico. Así, para una fresadora con interpolación lineal y 3 ejes tendremos las siguientes configuraciones:

```
typedef  MODELOL31< EJEX < ILINEAL3 <maq>  > >   ejex;
typedef  MODELOL31< EJEY < ILINEAL3 <maq>  > >   ejey;
typedef  MODELOL31< EJEZ < ILINEAL3 <maq>  > >   ejez;
```

Para que lo anterior sea posible, el modelo puede aplicarse a varios ejes al mismo tiempo y la implementación de un modelo será como se muestra en la Figura 5.22.

Como puede observarse, esta implementación no manda llamar a ninguna otra capa ya que aquí la estructura no es en capas sino que cada eje es totalmente independiente de los otros ejes. La parametrización a través de templates se usa para relacionar un eje con el modelo matemático dentro de esa estructura.

```
template <class ejedef >
class nombre-del-modelo: public ejedef
{
     cuerpo de la clase;
     .
     .
     double Calcula (parámetros)
                    Se efectúan los cálculos;
};
```

Figura 5.22 Implementación General para cada Modelo.

5.6. EL GENERADOR DE SISTEMAS DE PreTie.

El ensamblaje manual de todos los componentes del sistema, implica un profundo conocimiento de estos, así como de las limitantes que se pueden presentar al momento de ensamblarlos. Por ejemplo, podemos hablar de una máquina que tiene 3 ejes y soporta interpolación circular, y al momento de elegir un modelo para cada eje se selecciona un modelo diseñado específicamente para trabajar con interpolaciones lineales. Surge entonces una inconsistencia.

La razón de que MAQUINA tenga un método CalculaTiempo() que no efectúa ningún cálculo, es precisamente darle consistencia a la estructura.

Recordemos también que una fresadora además de distintos tipos de interpolaciones, puede tener de 3 a 5 ejes. Una vez que el programa determina la capa de interpolación donde se van a realizar los cálculos, el siguiente paso es identificar el o los ejes que se están moviendo en la instrucción leída. En base al eje que se mueve se manda llamar otra estructura que también debe definirse en el archivo de configuración.

Como se ilustra a continuación, es necesario tener una configuración para cada eje de la máquina fresadora, debido a que cada eje se relaciona con un modelo matemático específico. Así, para una fresadora con interpolación lineal y 3 ejes tendremos las siguientes configuraciones:

```
typedef  MODELOL31< EJEX < ILINEAL3 <maq>  > >   ejex;
typedef  MODELOL31< EJEY < ILINEAL3 <maq>  > >   ejey;
typedef  MODELOL31< EJEZ < ILINEAL3 <maq>  > >   ejez;
```

Para que lo anterior sea posible, el modelo puede aplicarse a varios ejes al mismo tiempo y la implementación de un modelo será como se muestra en la Figura 5.22.

Como puede observarse, esta implementación no manda llamar a ninguna otra capa ya que aquí la estructura no es en capas sino que cada eje es totalmente independiente de los otros ejes. La parametrización a través de templates se usa para relacionar un eje con el modelo matemático dentro de esa estructura.

```
template <class ejedef >
class nombre-del-modelo: public ejedef
{
     cuerpo de la clase;
     .
     .
     double Calcula (parámetros)
                    Se efectúan los cálculos;
};
```

Figura 5.22 Implementación General para cada Modelo.

5.6. EL GENERADOR DE SISTEMAS DE PreTie.

El ensamblaje manual de todos los componentes del sistema, implica un profundo conocimiento de estos, así como de las limitantes que se pueden presentar al momento de ensamblarlos. Por ejemplo, podemos hablar de una máquina que tiene 3 ejes y soporta interpolación circular, y al momento de elegir un modelo para cada eje se selecciona un modelo diseñado específicamente para trabajar con interpolaciones lineales. Surge entonces una inconsistencia.

Con la generación automática de sistemas, se pretende minimizar la posibilidad de error en la elección de los componentes y su correcto ensamblaje dentro del sistema. Como ya se ha mencionado en el capítulo II, una de las características de la programación generativa, es que el usuario puede definir un sistema en términos abstractos, y es el generador el que se encarga de revisar los detalles de implementación tales como los elementos que pueden ir y en que parte pueden ir tales elementos.

En la Figura 5.23 se muestra una representación de lo que es el sistema de PreTies automatizado. Por una parte tenemos al generador y por otra al sistema de PreTie en si, el cual depende directamente del generador en el sentido de que un cambio en las especificaciones dadas al generador, se refleja automáticamente en el PreTie que se está generando.

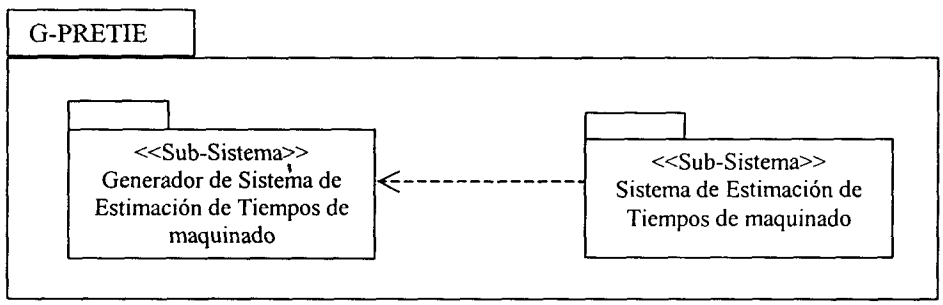


Figura 5.23 Esquema Generativo de sistemas PreTie

El generador trabaja en la construcción de un sistema a partir de las características seleccionadas para que se incluyan dentro del sistema. No es necesario saber de qué manera se ensamblan los diferentes componentes para crear un sistema funcional, sólo estar concientes de que a través de su declaración el sistema obtiene lo que desea.

5.6.1 La Pantalla del Generador de Software.

Básicamente, la pantalla de construcción que ofrece el sistema generador de PreTies es una matriz de elementos seleccionables (Figura 5.24).

Puesto que una fresadora consta por lo menos de 3 ejes, se muestra por defecto la pantalla que permite la elección tanto de los tipos de interpolador que soporta la fresadora, como de los modelos matemáticos que se aplican a cada uno de los 3 ejes. Si se selecciona una máquina de 5 ejes, entonces el sistema permite que se elijan los modelos matemáticos de los dos ejes restantes (Figura 5.25).

En este sentido, el generador ya está restringiendo la entrada de los datos es decir, si la máquina tiene sólo tres ejes, el generador no permite seleccionar modelos para un cuarto y/o un quinto eje. Este tipo de restricciones se da también en la parte de la elección de los

modelos. Se tiene una librería de modelos matemáticos especiales para cada tipo de interpolación. El generador no permite que se seleccione un modelo de interpolación lineal para ser usado en una interpolación helicoidal, ni un modelo circular para usarse en una interpolación splines.

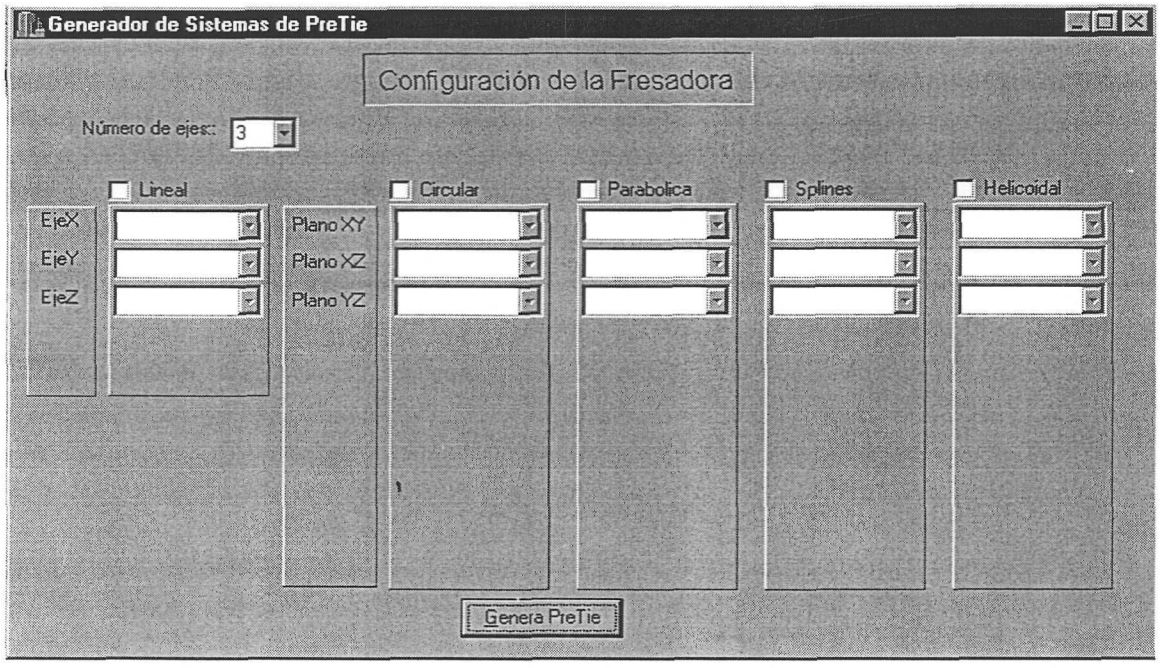


Figura 5.24. Pantalla del Generador de Sistemas PreTie.

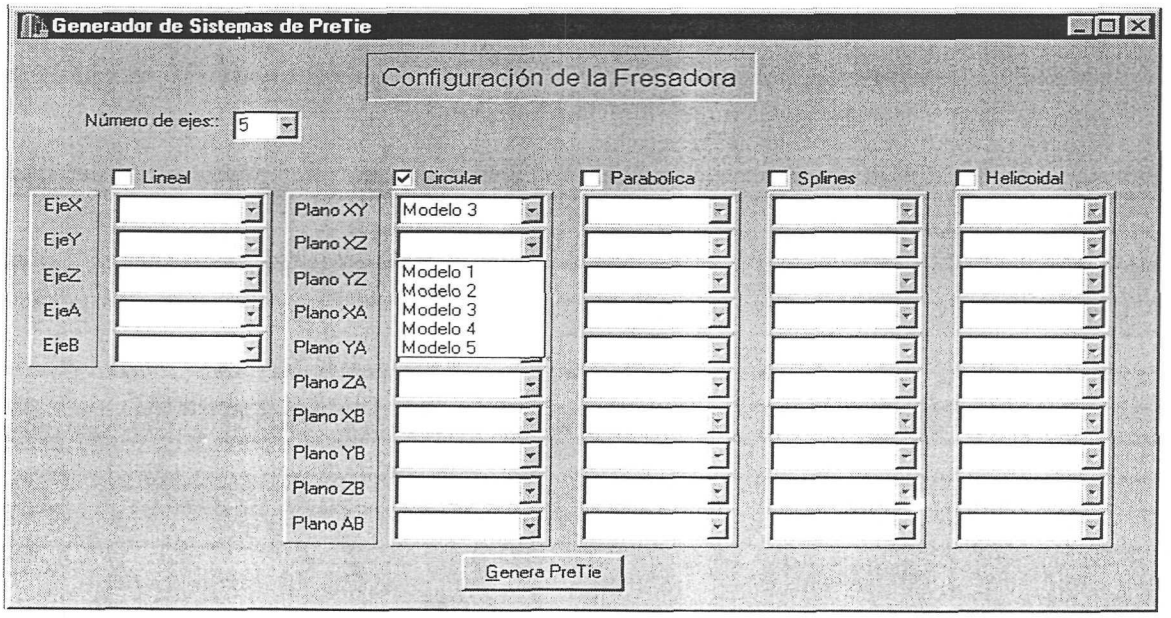


Figura 5.25. Captura de Modelos En Fresadora de 5 ejes con Interpolación Circular.

Una vez que se han indicado las definiciones abstractas del sistema PreTie deseado, el generador realiza el proceso de selección, adecuación y ensamblaje de los componentes necesarios para configurar un PreTie con las características deseadas.

Por ejemplo, la Figura 5.26 ilustra las definiciones necesarias para generar un sistema predictor tal como el mostrado en la figura 5.16.

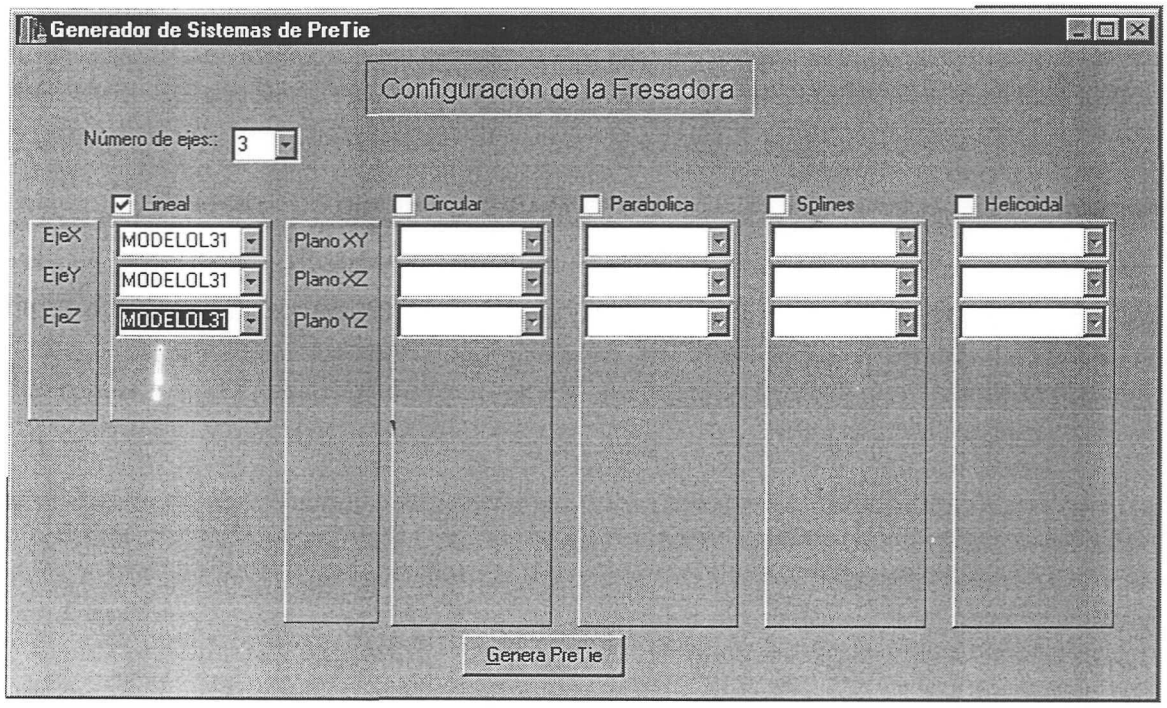


Figura 5.26 Definiciones de PreTie para fresadora de 3 ejes con interpolación lineal, que usa MODELO31 en cada eje.

5.6.2 Implementación del Generador.

El generador tiene un estilo arquitectónico de llamada y retorno, a través de una jerarquía donde el control pasa de un elemento "padre" a otro llamado "hijo" dentro del sistema.

Por otra parte y como se ha venido diciendo, los componentes de la familia de los sistemas PreTie son los diferentes tipos de interpolación, los modelos matemáticos y los ejes de la fresadora. Estos componentes se utilizan en combinación unos con otros para definir sistemas. Para ello podemos construir un conjunto de ecuaciones tipo que definen diferentes sistemas de la familia de productos. En base a estos componentes, podemos definir las siguientes librerías o reinos:

```
I={ ilineal3, icircular3, iparabolica3, ihelicoidal3...}
E={x, y, z, a ,b, xy, xz, yz, xa, ya, za, xb, yb, za, ab}
M={modeloL1...modeloLn, modeloC1.... modeloCn.....}
```

Donde I es la librería de los diferentes interpoladores, E es la librería de los ejes y planos de una máquina y M es la librería de modelos matemáticos.

A partir de la combinación de esos componentes, es que podemos crear sistemas de predicción de tiempos de maquinado. Un sistema se define por una ecuación tipo dentro del contexto de los generadores GenVoca. En la definición de sistemas que nos ocupa, las ecuaciones se definen de la siguiente manera:

Sistema1 = {I[I]} para el conjunto total de interpolaciones soportadas por una fresadora.

EjesSistema1= {M[E[I]} para cada eje o plano de la máquina fresadora que pertenece a un tipo de interpolación.

El generador crea un archivo denominado "Config.h" el cual incluye las ecuaciones tipo del sistema y crea las instancias necesarias de tales composiciones. Además, es en este archivo donde se definen los componentes que deberán ensamblarse en el PreTie generado a través de instrucciones #include de C++.

Existe además otro archivo denominado "Pantalla.dat" que define la pantalla de captura de parámetros que tendrá el PreTie generado. Puesto que la pantalla de captura de parámetros de calibración de una fresadora se define como una matriz en donde solamente se habilitan los renglones que corresponden a la cantidad de ejes o planos que tiene la fresadora y los parámetros que se requieren de acuerdo a los modelos matemáticos usados, es necesario modificar esta matriz para cada PreTie generado. Las columnas de la matriz representan el conjunto de parámetros que pueden manejar los distintos modelos matemáticos y es precisamente del modelo de donde se obtienen las columnas que deberán habilitarse dentro del PreTie generado. Los renglones están determinados por el número de ejes de la fresadora y los tipos de interpolación que soporta. Una fresadora de 4 ejes con interpolación circular habilitará 6 renglones que corresponden cada uno a los planos XY, XZ, YZ, XA, YA, ZA.

Una vez generados estos archivos y definida la pantalla de captura de parámetros, es necesario compilar el PreTie generado.

5.7. EJEMPLOS DE SISTEMAS PreTie GENERADOS.

A continuación se presentan algunos ejemplos de sistemas de PreTie generados. Para cada PreTie se muestra la pantalla de captura y su archivo de configuración.

Ejemplo 1: Maquina fresadora con 3 ejes, interpolación lineal y MODELOL31 en cada uno de sus ejes.

Donde I es la librería de los diferentes interpoladores, E es la librería de los ejes y planos de una máquina y M es la librería de modelos matemáticos.

A partir de la combinación de esos componentes, es que podemos crear sistemas de predicción de tiempos de maquinado. Un sistema se define por una ecuación tipo dentro del contexto de los generadores GenVoca. En la definición de sistemas que nos ocupa, las ecuaciones se definen de la siguiente manera:

Sistema1 = {I[I]} para el conjunto total de interpolaciones soportadas por una fresadora.

EjesSistema1= {M[E[I]} para cada eje o plano de la máquina fresadora que pertenece a un tipo de interpolación.

El generador crea un archivo denominado "Config.h" el cual incluye las ecuaciones tipo del sistema y crea las instancias necesarias de tales composiciones. Además, es en este archivo donde se definen los componentes que deberán ensamblarse en el PreTie generado a través de instrucciones #include de C++.

Existe además otro archivo denominado "Pantalla.dat" que define la pantalla de captura de parámetros que tendrá el PreTie generado. Puesto que la pantalla de captura de parámetros de calibración de una fresadora se define como una matriz en donde solamente se habilitan los renglones que corresponden a la cantidad de ejes o planos que tiene la fresadora y los parámetros que se requieren de acuerdo a los modelos matemáticos usados, es necesario modificar esta matriz para cada PreTie generado. Las columnas de la matriz representan el conjunto de parámetros que pueden manejar los distintos modelos matemáticos y es precisamente del modelo de donde se obtienen las columnas que deberán habilitarse dentro del PreTie generado. Los renglones están determinados por el número de ejes de la fresadora y los tipos de interpolación que soporta. Una fresadora de 4 ejes con interpolación circular habilitará 6 renglones que corresponden cada uno a los planos XY, XZ, YZ, XA, YA, ZA.

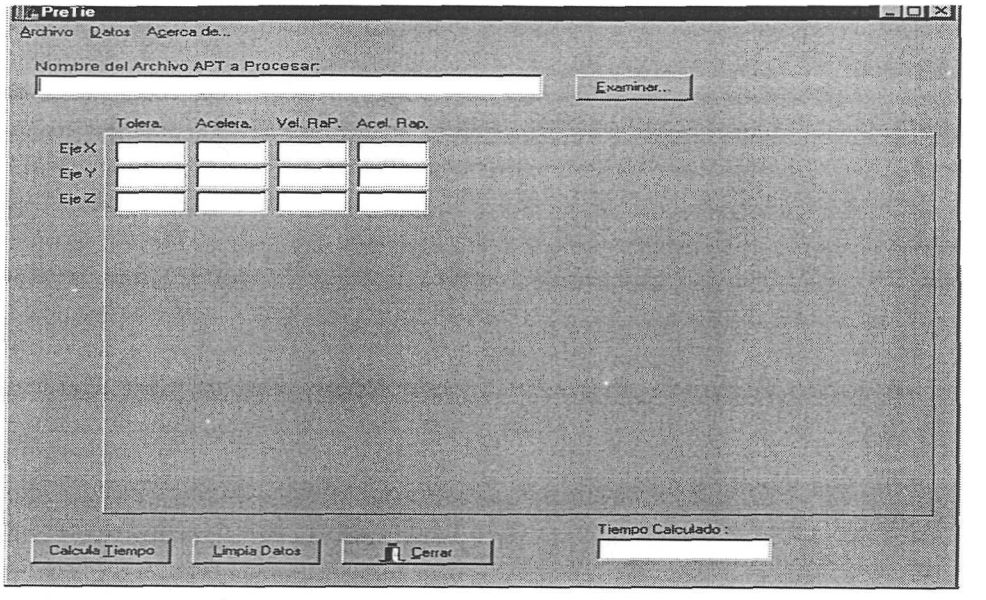
Una vez generados estos archivos y definida la pantalla de captura de parámetros, es necesario compilar el PreTie generado.

5.7. EJEMPLOS DE SISTEMAS PreTie GENERADOS.

A continuación se presentan algunos ejemplos de sistemas de PreTie generados. Para cada PreTie se muestra la pantalla de captura y su archivo de configuración.

Ejemplo 1: Maquina fresadora con 3 ejes, interpolación lineal y MODELOL31 en cada uno de sus ejes.

Pantalla de Captura:



Archivo de Configuración:

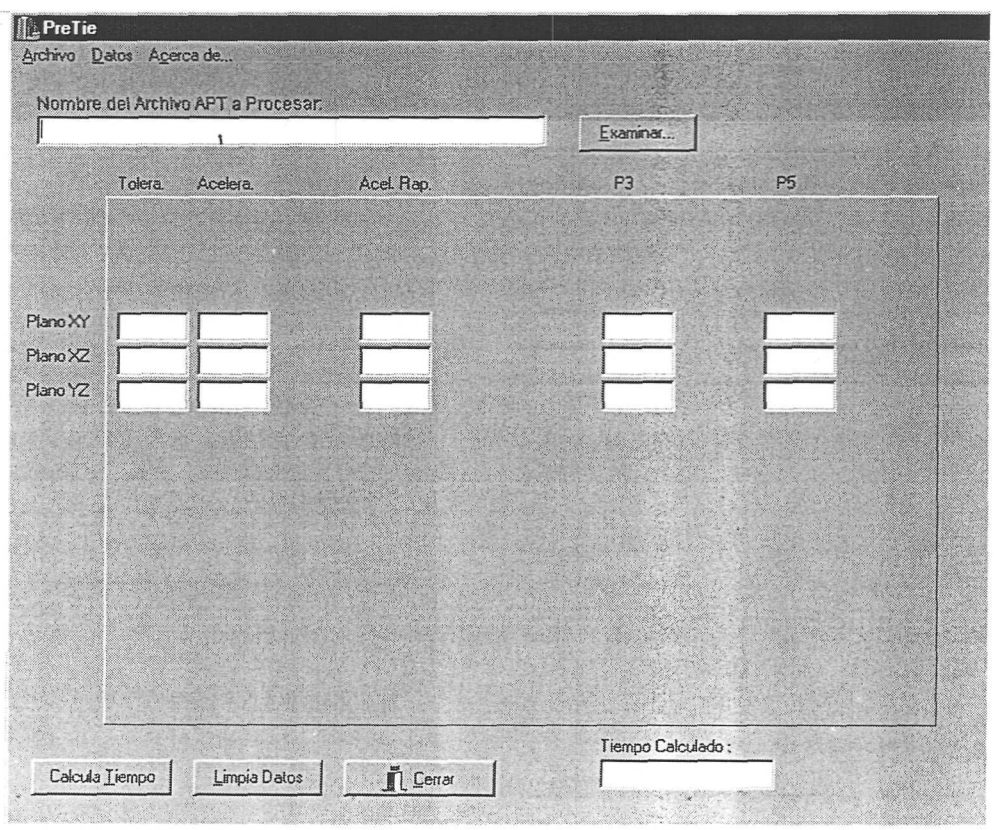
```
#ifndef _CONFIG_H
#define _CONFIG_H

#include "maquina.h"
#include "IL3.h"
#include "EJEX.h"
#include "EJEY.h"
#include "EJEZ.h"
#include "MODELOL31.h"

      typedef MAQUINA maq;

//Interpolaciones

      typedef ILINEAL3<maq> mh;
      mh* MH1=new mh;

//Ejes de la interpolación lineal
      typedef  MODELOL31< EJEX < ILINEAL3 <maq>  > >   ejex;
      typedef  MODELOL31< EJEY < ILINEAL3 <maq>  > >   ejey;
      typedef  MODELOL31< EJEZ < ILINEAL3 <maq>  > >   ejez;

      ejex* EX=new ejex;
      ejey* EY=new ejey;
      ejez* EZ=new ejez;

#endif
```

Ejemplo 2: Maquina fresadora con 3 ejes, interpolación circular y MODELOC31, MODELOC32, MODELO31 para los planos XY, XZ y YZ respectivamente.

MODELOC31 ocupa los parámetros Tolerancia y Aceleración.
MODELOC32 ocupa los parámetros Aceleración en RAPID, P3 y P5.

Cabe aclararar que la implementación de la interpolación circular es simplemente una llamada a los modelos, sin efectuar ningún procesamiento. Los modelos de la interpolación circular simplemente realizan sumas de los parámetros obtenidos. Estas implementaciones se hicieron con la finalidad de realizar pruebas

Pantalla de Captura:

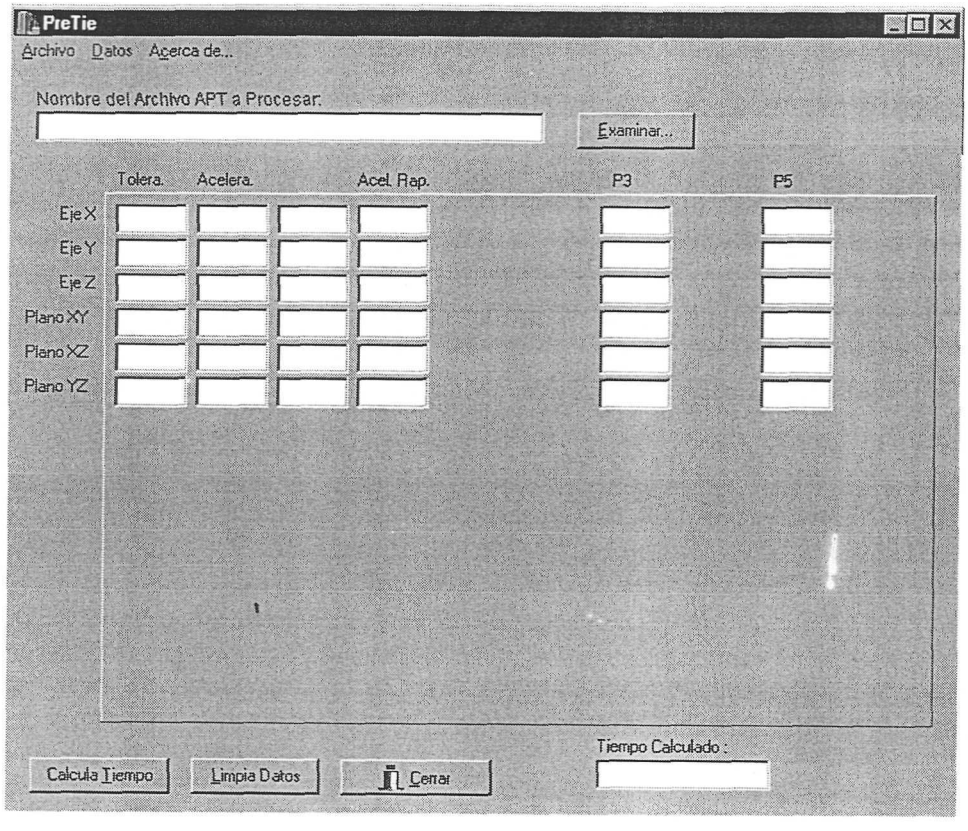


Archivo de Configuración:

```
#ifndef _CONFIG_H
#define _CONFIG_H

#include "maquina.h"
#include "IC3.h"
#include "PLANOXY.h"
#include "PLANOXZ.h"
#include "PLANOYZ.h"
#include "MODELOC31.h"
#include "MODELOC32.h"


      typedef MAQUINA maq;

//Interpolaciones

      typedef ICIRCULAR3<maq> mh;
      mh* MH1=new mh;

//Planos de la interpolación circular
      typedef  MODELOC31< PLANOXY < ICIRCULAR3 <maq>  > >   planoxy;
      typedef  MODELOC32< PLANOXZ < ICIRCULAR3 <maq>  > >   planoxz;
      typedef  MODELOC31< PLANOYZ < ICIRCULAR3 <maq>  > >   planoyz;

      planoxy* PXY=new planoxy;
      planoxz* PXZ=new planoXZ;
      planoyz* PYZ=new planoyz;

#endif
```

Ejemplo 3: Máquina fresadora con 3 ejes, interpolaciones lineal y circular y modelos MODELOL31 para cada uno de los ejes de la interpolación lineal y MODELOC32 para cada plano de la interpolación circular.

Pantalla de Captura:



Archivo de Configuración:

```
#ifndef _CONFIG_H
#define _CONFIG_H

#include "maquina.h"
#include "IL3.h"
#include "IC3.h"
#include "EJEX.h"
#include "EJEY.h"
#include "EJEZ.h"
#include "PLANOXY.h"
#include "PLANOXZ.h"
#include "PLANOYZ.h"
#include "MODELOC31.h"
#include "MODELOL32.h"



      typedef MAQUINA maq;

//Interpolaciones

      typedef ILINEAL < ICIRCULAR3<maq> >  mh;
      mh* MH1=new mh;

//Ejes de la interpolación lineal
      typedef  MODELOL31< EJEX < ILINEAL3 <maq>  > >   ejex;
      typedef  MODELOL31< EJEY < ILINEAL3 <maq>  > >   ejey;
      typedef  MODELOL31< EJEZ < ILINEAL3 <maq>  > >   ejez;

      ejex* EX=new ejex;
      ejey* EY=new ejey;
      ejez* EZ=new ejez;

//Planos de la interpolación circular
      typedef  MODELOC31< PLANOXY < ICIRCULAR3 <maq>  > >   planoxy;
      typedef  MODELOC32< PLANOXZ < ICIRCULAR3 <maq>  > >   planoxz;
      typedef  MODELOC31< PLANOYZ < ICIRCULAR3 <maq>  > >   planoyz;

      planoxy* PXY=new planoxy;
      planoxz* PXZ=new planoxz;
      planoyz* PYZ=new planoyz;

#endif
```

CAPÍTULO VI.

CONCLUSIONES Y TRABAJOS FUTUROS

6.1 CONCLUSIONES.

El objetivo de la tesis se logró al diseñar una arquitectura que permite el manejo de la gran variabilidad de los sistemas de predicción de tiempos de maquinado en fresadoras de alta velocidad. La definición de este dominio como una familia de productos fue de gran ayuda en el proceso de automatización de la creación de nuevos miembros de esa familia.

La implementación de una arquitectura de software de dominio específico basada en componentes, es relativamente sencilla si el análisis de dominio efectuado capta de manera clara la variabilidad que existe en los sistemas o miembros de una familia de productos. Sin embargo, un análisis pobre puede llevar a fracasos dentro de la definición de la arquitectura y por lo tanto de la implementación, mismos que se reflejarán más adelante en la creación de nuevos miembros y componentes deteriorando cada vez más el sistema. Por esto cabe tomar en consideración que el analista debe apoyarse grandemente en personas que tengan un amplio conocimiento del dominio y de ser posible que el mismo analista esté fuertemente ligado al dominio en estudio.

En el desarrollo de este tipo de aplicaciones, es importante conocer las características de las distintas herramientas que en un momento dado pueden emplearse en su implementación, tal es el caso de la tecnología de los compiladores que se usaron para estandarizar un componente empleado como entrada del sistema.

Además los avances que se han dado dentro de los lenguajes de programación permiten explorar nuevas posibilidades. Tal es el caso de los templates de C++ que permiten la parametrización de tipos y con ello posibilitan la implementación de arquitecturas para generación de familias de productos tales como el modelo GenVoca.

Aunque el generador de aplicaciones implementado sólo cuenta dentro de su librería de componentes con un modelo matemático de aplicación a la interpolación lineal, soporta la adición de modelos matemáticos de aplicación no sólo a la interpolación lineal, sino a las interpolaciones circular, parabólica, splines y helicoidal gracias a la arquitectura definida.

6.2 TRABAJOS FUTUROS.

Dentro del área de manufactura queda abierta la tarea de incorporar nuevos elementos dentro del sistema implementado, haciendo reuso tanto del análisis del dominio como de la arquitectura propuesta.

Dentro del área de software queda aún mucho por hacer ya que la cantidad de dominios en que puede aplicarse este tipo de tecnologías es muy amplio y en la práctica está poco explorado. Por lo tanto se proponen la siguiente línea de investigación:

Realizar un análisis de las tendencias en las empresas de desarrollo de software, a utilizar este enfoque de familias de productos y líneas de producción, y proponer una metodología de cambio del enfoque actual de producir una aplicación para una necesidad específica, a producir una familia de sistemas aplicables a una gran variedad de necesidades dentro de un dominio de sistemas. Esta metodología debe estar orientado a permitir a las empresas cumplir con sus desarrollos actuales pero adentrándose al mismo tiempo en este nuevo enfoque de familias de sistemas.

Explorar nuevos dominios de aplicación, con otras características y complejidades aún mayores para validar la aplicación del enfoque de familias en tales dominios. Esto permitirá el desarrollo aplicaciones cada vez mejores en más áreas y la disminución de los costos de producción del software. Así mismo, permitira descartar aquellos dominios en donde este tipo de enfoque no sea aplicable posibilitando la investigación de nuevos enfoque sobre ellos.

Analizar las características de otros ambientes de desarrollo que permitan aplicar esta técnica a fin de ampliar el abanico de posibilidades de implementación en diferentes ambientes o bien, de mejorar los ambientes de acuerdo a las necesidades de los procesos de desarrollo de software de ser más productivos.

ANEXO I

ECUACIONES DEL MODELO MATEMÁTICO

$$P_1 = x_2 - x_1 \qquad P_4 = x_3 - x_2 \qquad P_7 = x_4 - x_3$$
$$P_2 = y_2 - y_1 \qquad P_5 = y_3 - y_2 \qquad P_8 = y_4 - y_3$$
$$P_3 = z_2 - z_1 \qquad P_6 = z_3 - z_2 \qquad P_9 = z_4 - z_3$$

$$L12 = \sqrt{P_1^2 + P_2^2} + P_3^3 \qquad L23 = \sqrt{P_4^2 + P_5^2} + P_6^3 \qquad L34 = \sqrt{P_7^2 + P_8^2} + P_9^3$$

$$ar_1 = P_1 * P_4 + P_2 * P_5 + P_3 * P_6 \qquad ar_1 = P_4 * P_7 + P_5 * P_8 + P_6 * P_9$$
$$ar_2 = L12 * L23 \qquad ar_2 = L23 * L34$$
$$ar = ar_1 / ar_2 \qquad ar = ar_1 / ar_2$$

```
teta1=acos(ar)
```

```
teta2=acos(ar);
```

$$ar = 1 - \cos\left(\frac{teta1}{2} * \frac{\Pi}{180}\right) \qquad ar = 1 - \cos\left(\frac{teta2}{2} * \frac{\Pi}{180}\right)$$

$$Rc1 = e * \left(\frac{\cos\left(\frac{teta1}{2} * \frac{\Pi}{180}\right)}{ar} \right) \qquad Rc2 = e * \left(\frac{\cos\left(\frac{teta2}{2} * \frac{\Pi}{180}\right)}{ar} \right)$$

$$Vcstart = \sqrt{Am * Rc1} * \frac{60}{1000}$$

$$Vcend = \sqrt{Am * Rc2} * \frac{60}{1000}$$

$$Vinidec2\_3 = \sqrt{\left(\frac{Vcend}{60}\right)^2 + 2 * \left(\frac{Am}{1000}\right) * \left(\frac{L23}{1000}\right)} * 60$$

$$Xm = \left(\frac{Vcstart^2 + Vinidec2\_3^2}{4 * \frac{Am}{1000} * \frac{1000}{3600}} \right)$$

$$tm = \frac{\frac{Vm}{60} - \frac{Vcstart}{60}}{\frac{Am}{1000}}$$

$$tXm\_xf = \frac{\frac{Vm}{60} - \frac{Vcend}{60}}{\frac{Am}{1000}}$$

Caso tipo A Vfprog>Vm	Caso tipo B Vfprog<=Vcstart y Vfprog<=Vcend	Caso tipo C
		$t1 = \frac{(Vfprog - Vcstart)*1000}{Am*60}$
tf=tm+tXm_xf	$tf = \frac{\frac{L23}{60}}{Vfprog*1000}$	$X1 = \frac{\left(\frac{Vcstart*t1}{60} + Am*t1^2\right)}{2000}*1000$
		$dX2\_xf = \frac{(Vfprog^2 - Vcend^2)*10000}{2*Am*3600}$
		$tX2\_xf = \frac{(Vfprog - Vcend)*1000}{\frac{Am}{60}}$
		dX1_x2=L23-dX2_xf-X1
		$tX1\_x2 = \frac{dX1\_x2*60}{\frac{Vfprog}{1000}}$
		t2=t1+ tX1_x2 tf= t2+ tX2_xf

ANEXO II

PROGRAMAS FUENTE DEL PREPROCESADOR.

Analizador Léxico

```
%{
 //Archivo cnc1.l
 //Contiene las reglas léxicas

  #include "cnc1.tab.h"
  #include <iostream.h>
 %}

DIGIT [0-9]

%%
{DIGIT}+"."{DIGIT}*          {return ESCALAR;}
RAPID                        {return RAPID;}
FEDRAT                       {return FEDRAT;}
SPINDL                       {return SPINDL;}
GOTO                         {return GOTO;}
CIRCLE                       {return CIRCLE;}
IPM                          {return PULGADA;}
MMPM                         {return MILIMETRO;}
RPM                          {return REV_MIN;}
OFF                          /*Desecha el OFF*/

 /* Hasta el primer * toma las lineas que empiezan con $$ seguido de cualquier
caracter excepto nueva linea y $. Si el comentario continua en más líneas busca a
partir del $seguido de nueva línea seguido de cualquier caracter menos \n y $.
Esto ultimo se presenta 0 veces cuando el comentario no continua */

"$$"[^\n\$]*("$"\n[^\n\$]*)*     {return COMENTARIO;}
"$"                              {return CONTINUA;}
"-"                              {return Neg;}
[\n]                             /*Desecha new line*/
[ ]                              /*Desecha blancos*/

 /* El punto indica cualquier caracter excepto new line*/

.                                /*Desecha cualquier otro caracter*/

%%

int yywrap(void) {return 1;}
```

Analizador Sintáctico

```
%{
  // Archivo cnc1.y
  // Contiene las reglas de gramatica para el parser

  #include "lex.yy.c"
  #include <iostream.h>
  #include <fstream.h>
  #include <stdio.h>

  //esta libreria llama los atoi() atof() etc.
  #include <stdlib.h>

  const unsigned TAM_LINEA=128;
  int error=0;
  char vf[],vs[], vx[], vy[], vz[], vxr[], vyr[],vzr[],vo[],*num;
  char linea[TAM_LINEA];
%}

%start programa

%token  ESCALAR
%token  RAPID
%token  FEDRAT
%token  SPINDL
%token  GOTO
%token  CIRCLE
%token  PULGADA
%token  MILIMETRO
%token  REV_MIN
%token  OFF
%token  CONTINUA
%token  COMENTARIO
%left   Neg

%%

programa: lista_comandos
;

lista_comandos: /*string vacio*/
              |comando
              lista_comandos
;

comando: lista_lineas
;

lista_lineas:renglon
           |renglon
            CONTINUA
           lista_lineas
;

renglon: proposicion
       | COMENTARIO
       | punto
```

```
        numero      { vo=num;
                      linea[]="0\t0\t0\t"+
                      cout<<"0 0 0 "<<vxr<<" " <<vyr<<" "<<vzr<<" ";
                      cout<< vx<<" "<<vy<<" "<<vz<<" " <<vo<<endl;
                      f.getline(linea, TAM_LINEA);
                     }
;

proposicion: RAPID  {linea[]="0\t 0\t 0\t 0\t 0\t 0\t 0\t 0\t 0\t 0";
                     f.getline(linea, TAM_LINEA);
                     cout<<linea<<endl;
                     }
         |FEDRAT
          numero     {vf[]=*num;}
          unidad     {linea[]=CONCATENATE(yytext," ", vf,
                                  "0\t0\t0\t0\t0\t0\t0");
                     f.getline(linea, TAM_LINEA);
                     cout<<linea<<endl;
                     }
         |SPINDL
          parametros
         |GOTO
          punto      {cout<<"0 0 0 0 0 0 "<<vx<<" "<<vy<<" "<<vz<<"
                      "     0"<<endl;}
         |CIRCLE
           radio     {vxr=*vx; vyr=*vy;vzr=*vz;}
;

numero:Neg
       ESCALAR                    {*num=CONCATENATE("-",yytext);}
      |ESCALAR                    {*num=yytext;}

unidad: PULGADA
      | MILIMETRO
      | REV_MIN
;

parametros: unidad
          numero            {vs[]=*num;
                             linea[]=CONCATENATE("0\t0\t",vs," 0\t 0\t 0\t
                                          0\t 0\t 0\t 0 ");
                             f.getline(linea, TAM_LINEA);
                             cout<<linea<<endl;
                            }
         |OFF
;

punto: numero               {vx=*num;}
       numero               {vy=*num;}
       numero               {vz=*num;}
;

radio:punto
;
%%

yyerror()
{
```

```
 cout<< "El ultimo token reconocido es:" <<yytext<<endl;
 return;
}

main(int argc, char *argv[])
{
   fstream f;
   f.open("salida.out", ios::out)
   ++argv;--argc;
   if (argc<1)
     cout<<"EL archivo no se pudo abrir"<<endl;

   yyin=fopen(argv[0],"r");

   salida=fopen(argv[1],"w");

   cout<<"Va a ejecutar el yyparse"<<endl;
   yyparse();
   cout<<endl<<" Ya ejecuto el yyparse"<<endl;

   f.close();
   return 0;
}
```

ANEXO III

FORMATO DEL ARCHIVO ESTANDARIZADO DE INSTRUCCIONES DE M0VIMIENTO.

Columna	Contenido	Descripción
1	C G S	Se trata de una interpolación circular. Se trata de una interpolación lineal. Se trata de una instrucción spindl
2	RPM MMM	Son las unidades que se manejan
3	0 <> 0	Representa un Rapid Representa el Fedrate de la instrucción
4	Dato numérico	Es un parámetro del spindl
5, 6, 7	Dato numérico	Representan las coordenadas X, Y, Z de una instrucción GOTO
8, 9, 10	Dato numérico	Representan las coordenadas del radio de una instrucción CIRCLE
11	Dato numérico	Es un parámetro de la instrucción CIRCLE

Bibliografía.

[Aho;1990] Aho Alfred V, Sethi Ravi, Ullman Jeffrey D. Compiladores. Principios, Técnicas y Herramientas. Addison- Wesley Iberoamericana. 1990.

[Arnush; 1996] Arnush, Craig. Aprendiendo Borland C++ 5 en 21 días. Ed. Prentice Hall Hispanoamericana. México. 1996.

[Barroca; 1999] Barroca Leonor, Hall Jon, Hall Patrick (eds). Software Architectures. Advances and Applications. Ed. Srpinger. London 1999.

[Bass; 1998] Bass Len, Clements Paul, Kazman Rick. Software Arquitecture in Practice. Ed. Addison-Wesley. March 1998.

[Batory; 1997] Batory Don, Geraci Bart J. Composition, Validation and Subjetivity in GenVoca Generators. Software Engineering, IEEE Transactions on , Volume: 23 publicación 2, 1997

[Batory;] Batory Don. Software System Generators, Transformation Systems, and Compilers. University Of Texas.

[Batory; 1995] Batory Don, Coglianese Lou, Goodwin Lou, Shafer Steve. Creating Reference Architecture : An example from avionics. Symposium on Software Reusability 1995, Seatle, Washington.

[Booch; 1999] Booch Grady, Rumbaugh James, Jacobson Ivar. The Unified Modeling Language User Guide. Addison-Wesley Ed. Feb. 1999.

[Borland; 1999] Borland C++ Builder v. 4.0 (Build 14.4). Inprise Corporation.

[Butler; 1968] Butler R., Kerr E. An Introduction to Numerical Methods. Pitman Publishing Co. 1968.

[Calderón; 1994] Macías, Marcos Calderón y Nieto Edgar Herrador. C++: Un Lenguaje de Programación Orientado a Objetos. Soluciones Avanzadas. México, D. F Noviembre 1994. Infolatina.

[Chang; 1998] Chang Tien-Chien, Wysk Richard A., Wang Hsu-Pin. Computer-Aided Manufacturing. Second Edition. Ed. Prentice Hall. New Jersey. 1998.

[Czarnecki; 1999] Czarnecki, K. and Eisenecker, U. Components and Generative Programming. 7th. ACM SIGSOFT Symposium on the Foundations of Software Engineering. Tolouse, France. 1999

[Eclac; 1998] Inversión Extranjera Directa En La Industria Automotriz Latinoamericana: Respuestas A Desafios Globales. Revista de la Comisión Económica para América Latina y del Caribe (CEPAL) http://www.eclac.cl/espanol/Publicaciones/inver98/notaautos.htm. 1998.

[Flores, 1985] Flores, Benito PH. D., González V. Fernando PH. D., Franco Morones Raúl. Apuntes de Sistemas de Producción. Instituto Tecnológico y de Estudios Superiores de Monterrey. Departamento de Ingeniería Industrial. Febrero 1985.

[Garlan; 1994] Garlan David, Shaw Mary. An Introduction to Software Architecture. Tech, report CMU-CS-84-166, Carnegie Mellon University, Enero 1994.

[González; 1998] González Jaime. El Discreto Encanto del Metamodelo de UML. Soluciones Avanzadas. Infolatina. México, D.F. Agosto 1998.

[Greiff; 1994] Greiff, Warren R. Paradigma vs. Metodología; El Caso de la Programación Orientada a Objetos. Soluciones Avanzadas. México, D. F. Enero 1994. Infolatina.

[IBM; 1991] IBM. AIX Workstation/6000. Installing and Using APTWS/6000. Release 1.0. March 1991.

[Ibargüengoitia;1999] Ibargüengoitia Guadalupe, Marcellin Sergio. Tecnología Orientada a Objetos para el Comercio Electrónico. Soluciones Avanzadas. Infolatina. México, D. F. Mayo 1999.

[Katrib; 1994] Katrib Miguel Mora. PRINCIPIO ABIERTO CERRADO: C++ vs. EIFFEL, Rounds 2 y 3. Soluciones Avanzadas. México, D. F. Marzo 1994. Infolatina.

[Kief; 1988] Kief Hans B. Manual CN/CNC 1998. Gran Duc, s.l. 1988

[Luckham1995] Luckham David C., Vera James, Meldal Sigurd. Three concepts of System Architecture. July 19, 1995.

[Mompín; 1988] Mompín Poblet, José et. al., Sistemas CAD/CAM/CAE. Diseño y Fabricación por computadora. Publicaciones Marcombo, S.A. México-Barcelona. Serie:Mundo Electrónico

[Perry; 1992] Perry Dewayne E., Wolf Alexander L. Foundations for the Study of Software Architecture. ACM SIGSOFT. Software Engineering Notes. Vol. 17 no. 4 Oct. 1992 Page 40.

[Pressman; 1993] Pressman, Roger S. Ingeniería del Software. Un enfoque práctico. 3era. Edición. Ed. McGrawHill. México. 1993.

[Rodríguez; 1998] Rodríguez Ciro. Predicción del tiempo de maquinado para operaciones de fresado de alta velocidad. Proyecto de Instalación. Protocolo de Investigación. División de Ingeniería y Arquitectura Campus Monterrey. ITESM. Monterrey NL. Octubre 1998

[Rodríguez; 1999] Rodríguez Ciro. Estimation of Machining Time in High Speed Milling of Prismatic Parts. Technical Papers of the North American Manufacturing Research Conference, Society of Manufacturing Engineers. U. of California at Berkeley. Department of Mechanical Engineering. Instituto Tecnológico y de Estudios Superiores de Monterrey. Monterrey, NL. Méx. May 1999.

[Ron; 1991] Ron McCain. Reusable Software Component Construction a product-oriented Paradigm. Citado por: Rubén Díaz Prieto en Domain Analysis and Software System Modeling.1991.

[Weiss; 1999] Weiss, David M.; Robert Lai, Chi Tau. Software Product-Line Engineering. A Famili-Based Software Development Process. Adison Wesley Press.

[Wilson; 1963] Wilson Frank W. NC in Manufacturing. McGraw-Hill. 1963