UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

METODOLOGIA PARA EL DISEÑO DE LENGUAJES DE ALTO NIVEL

T E S I S D E L I C E N C I A T U R A

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO MECANICO ELECTRICISTA

PRESENTA:
DANIEL RIOS ZERTUCHE ORTUÑO

DIRECTOR DE TESIS:
ING. LUIS G. CORDERO B.

MÉXICO, D.F. 1986

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

INTRODUCCION

Luchamos con el lenguaje

Estamos en lucha con el lenguaje

Ludwig Wittgenstein 1931

Ante la necesidad de apoyo informático del Departamento de Investigación Económica, se observó que una de las áreas que mayor atención requería era la Unidad de Macroeconometria, tanto para la integración de los modelos como para su explotación.

El modelado ha sido una herramienta muy difundida en áreas que incluyen virtualmente el rango completo de las ciencias físicas y sociales. Como un esfuerzo de modelado se debe considerar cualquier intento para reducir la descripción de un fenómeno a un conjunto de relaciones estilizadas, las cuales se aproximen a los hechos observados.

Desde este punto de vista un amplio rango de investigaciones humanas pueden verse como intentos de construcción de modelos. Los tipos de modelos con los que los economistas tienen que ver, sin embargo, pueden generalmente ser caracterizados como capaces de ser representados matemáticamente; más aun para un econometrista, estos modelos son estocásticos y sujetos a refutación empirica. En el estado mas grande y complejo, ellos son simultáneos, dinámicos, no lineales e involucran cientos de relaciones.

Los modelos de la economía como un todo se empiezan a usar en la década 1930 - 1940 y son especialmente útiles al explicar

los modelos teóricos de la economía Keynesiana y los intentos independientes para construir modelos dinámicos del ciclo económico; de ahí hubo una progresión natural hacia los modelos estimados estadísticamente con el propósito de analizar las condiciones del mundo real.

En la teoria económica, particularmente la Teoria macroeconómica y la Teoría del ciclo de los negocios, los modelos compactos y abstractos son la norma actualmente.

El pronóstico económico fue un reto antes del advenimiento de los modelos: La práctica estandar fue el análisis de series de tiempo, esto es, muchas series de tiempo individuales eran analizadas por separado en términos de su propia historia a fin de proyectarlas hacia el futuro. Otra práctica común era el utilizar el análisis de una muestra del mercado.

En vez de confinar el análisis a la historia contenida en una serie de tiempo o en varias series de tiempo firmemente interrelacionadas, el cuerpo total del pensamiento económico se lleva a soportar el problema de construir el mejor conjunto de interrelaciones, el número de ecuaciones es determinado por el requerimiento que debe haber tantas relaciones como magnitudes económicas a ser estimadas.

Un modelo puede usarse, en un sentido mecánico tal y como las relaciones formales de las series de tiempo son empleadas, excepto, que el modelo deberá resolverse simultáneamente.

Hay dos diferencias muy significativas que son ventajosas para el método de modelado. Primero, hay un intento sistemático para considerar los efectos de variables no económicas o determinadas externamente que tienen importantes efectos económicos. Esas variables pueden ser determinadas por decisiones en materia de política fiscal, monetaria o por acontecimientos en el mundo externo al modelo. Otro grupo lo constituyen los instrumentos de política económica, tal como los montos de presupuesto, los impuestos, la tasa de interés, las reservas monetarias y otros elementos de política económica que pueden alterarse por decisión política. La Otra diferencia importante asociada con el método de modelado es que éste puede ser usado para estudios rápidos de los efectos de políticas económicas alternativas. El modelo toma en cuenta no solo los efectos directos, sino también los indirectos que son generalmente difíciles de detectar y mas de calcular, así un modelo de ecuaciones simultáneas es el camino natural para examinar y estimar estos efectos indirectos, este el el caso en que mas se necesita un modelo.

También en las proyecciones de largo plazo para investigación económica de problemas como suministro de

energía, producción de alimentos, recursos ecológicos, etc., un modelo sirve para mantener la consistencia y el balance de los pronósticos. A menos que se tenga formalmente estructurado un modelo, no habrá seguridad de que las distintas variables ajusten en las proyecciones. El cálculo del modelo implica un intento serio por imponer una disciplina de consistencia en el conjunto de las diversas magnitudes económicas proyectadas.

De la semblanza anterior del papel que juega el modelado en la Macroeconometría resaltan varios puntos; primero, los modelos son un puntal para el proceso de toma de decisiones racionales; segundo, la labor de mantener un modelo afinado y operable implica el esfuerzo continuo de muchas personas con un nivel de especialización alto; tercero, los modelos forzosamente están en continua evolución; cuarto, se tiene experiencia y se conocen bien las herramientas matemáticas que se emplean en la construcción de estos modelos; quinto, se trata de buscar nuevas herramientas matemáticas que superen las deficiencias de los actuales, el modelado que se utiliza prácticamente hace uso de un conjunto estable de herramientas.

A lo largo del análisis se identifico como una solución factible, el diseñar un lenguaje de alto nivel para atender esta necesidad de forma específica. Encontrandose que no se contaba

con una metodología enfocada a este fin, aquí se presenta una metodología y su aplicación al caso de la Macroeconometría.

SOFTWARE PARA USUARIO FINAL

> Cuando oimos a un chino, nos inclinamos a considerar su lenguaje como un balbuceo inarticulado. Pero quien entiende el chino reconocerá allí el lenguje
>
> Ludwig Wittgenstein, 1914

El fin del Software para computadora es proveer un marco dentro del cual el usuario puedá manejar sus problemas computacionales específicos. Los equipos modernos requieren que parte de los usuarios tengan un profundo conocimiento del Software de propósito general con que se cuenta, siendo que un gran número de usuarios han desarrollado la habilidad suficiente para manejar estos recursos, un número aún mayor de usuarios potenciales en diversas áreas de aplicación generan grandes demandas de Software para uso específico. La década de los setenta fue testigo de un importante incremento en la importancia del Software de aplicación; como resultado de esto un nuevo grupo de programadores ha surgido, llamados programadores de aplicaciones. Este tipo de programador generalmente tiene su origen en los usuarios de la computadora con amplia experiencia en un área de aplicación. Siendo su objetivo final desarrollar Software de aplicación específica que permita a los usuarios de diferentes áreas resolver sus problemas sin gran esfuerzo adicional.

Así el propósito fundamental de el Software de computadoras es proveer una conexión exitosa entre el Hardware de la computadora y la estructura y naturaleza del mundo real; esto

puede ser a través de la traducción de las formas explicitas e implicitas proporcionadas por los expertos.

Hasta cierto nivel el Ingeniero en Software y Hardware puede apoyarse en los esfuerzos y buena voluntad de los usuarios para ayudar en esta traducción. Pero conforme la tecnología se hace más avanzada y complicada, el universo crece y se requiere mayor capacidad de aprendizaje y de procesamiento cognoscitivo, por lo que se vuelve mas importante proveer a los usuarios de ayuda adecuada a fin de evitar la sobrecarga cognoscitiva.

Conforme más se conoce acerca de la traducción de información entre el medio ambiente y el individuo para una región particular de la experiencia humana, es más deseable y posible crear algoritmos para traducir esta entre la máquina y la persona, y hacer explicitos los modelos, el conocimiento y la forma de crear estructuras usadas para tal traducción. A fin de reducir el esfuerzo requerido para el desarrollo de aplicaciones individuales y de incrementar la eficiencia operativa es ventajoso representar estos direcatmente como lenguajes.

A través de las hojas de cálculo y otros lenguajes de muy alto nivel para aplicaciones de negocios, los cuales están de moda de varios años a la fecha, se ha tenido ya oportunidad de satisfacer los requerimientos de los profesionales con lenguajes para computadora de uso especifico. Entre estas necesidades

debemos señalar: Diseño por Computadora, Automatización de oficina, Robotica, Hoja de Cálculo Electrónica y muchas más.

Al tratar de manejar los sistemas de computadora (Hardware y Software) con respecto a los patrones de trabajo de la gente en su medio ambiente , debemos preocuparnos por diseñar lenguajes que los ayuden. El punto es entonces cómo estructurar las interfases con los usuarios y con qué fines los lenguajes de computadora deben ser rediseñados partiendo de nuevos principios. En relación a esto es necesario tomar la decisión respecto a que parte del conocimiento, es decir, que patrones del pensamiento deben ser directamente representados en el lenguaje de la computadora y cuales deben representarse de otra forma como son enfoques de Base de Datos, representaciones analógicas, etc.

En este entorno actualmente se puede observar gran actividad en el desarrollo de productos con fines comerciales. Debe hacerse notar que un buen producto ofrecerá una herramienta técnicamente útil, que cubra las necesidades del trabajo en su medio ambiente, así como las demandas cognoscitivas y de estilo del usuario.

Los lenguajes como el ensamblador son en la mayor parte extremadamente adaptables de tal forma que en principio es

posible crear lenguajes de muy alto nivel a la medida de los requerimientos de una aplicación o profesión dada. Sin embargo, la determinación adecuada de los requerimientos y una alta calidad en el diseño para una aplicación requiere: Ingenieros en Software para la programación, expertos en factores humanos para aclarar la naturaleza de la interacción entre el hombre y computadora Hardware/Software y el problema a ser resuelto, y grupos de profesionales en el área en cuestión para identificar mas claramente y de manera adecuada como ellos piensan o que su pensamiento refleja en las aplicaciones en las cuales la computadora va a ser utilizada.

Teniendo en cuenta lo anterior, se puede decir que no existe un lenguaje de computadora perfecto, solo hay lenguajes los que están más o menos adaptados a una operación en particular. Todos los lenguajes de computadora tienen beneficios particulares, así como desventajas con respecto a patrones de trabajo para los cuales no fueron diseñados. Esto es aplicable tanto a los lenguajes de muy alto nivel como a los de bajo nivel.

Es importante reflexionar que entendiendo como error de Software todo aquel comportamiento de los programas que no es esperado podemos aislar como la causa más común de esos errores a las equivocaciones cometidas durante la traducción de la información esto es, se puede describir a la producción de Software simplemente como un conjunto de procesos que inicia con

un problema y con una gran colección de instrucciones detalladas que dirigen a la computadora en la solución de tal problema. En otras palabras, como ya se expuso anteriormente la programación es la solución de problemas y el software es una colección de información describiendo la solución de un problema. La producción de Software es entonces un conjunto de procesos de traducción en los que se traduce el problema inicial a más soluciones intermedias hasta llegar a el conjunto detallado de instrucciones para la computadora. Los errores de Software se introducen cuando se falla al traducir una representación del problema a la solución de este ó a otra representación más detallada.

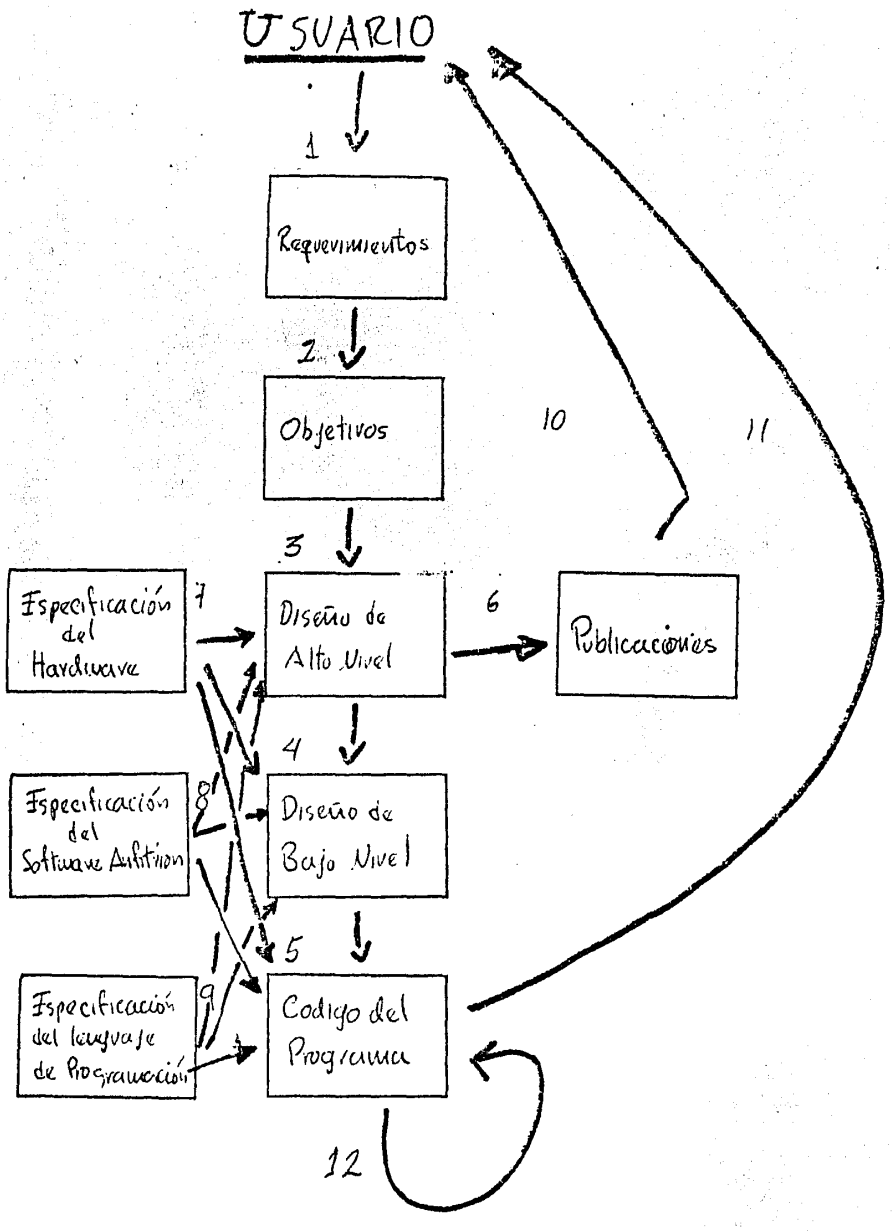
Podemos distinguir dos modelos de traducción involucrados en la realización de Software y estos son el Macroscopico y Microscopico.

Según Myers [MYER 76] los errores de Software pueden ser examinados en detalle observando las distintas clases de traducción que ocurren en el proceso de traducción del Software; como es mostrado en la figura 1.

Antes de discutir el modelo se deben tener en cuenta varios detalles. El proceso de diseño es representado únicamente por dos pasos, aunque en la práctica se manejan más. La intención del modelo no es delinear un conjunto de procesos

como recomendación, sino ilustrar los tipos de traducción que se tienen.

Figura 1


USUARIO
1
Requerimientos
2
Objetivos
3
Especificación del Hardware
7
Diseño de Alto Nivel
6
Publicaciones
4
Especificación del Software Auditivo
8
Diseño de Bajo Nivel
5
Especificación del lenguaje de Programación
9
Codigo del Programa
10
11
12

Examinemos brevemente cada tipo de traducción mostrado en la figura 1.

1.- El proceso inicia con el desarrollo de una descripción del problema a resolver. Esta descripción se convierte en un enunciado de los requerimientos del usuario final. En algunos casos el usuario escribe un enunciado de requerimientos. En otros casos el fabricante del Software es el que plantea el enunciado de requerimientos, ya sea por entrevista al usuario, por investigación de las necesidades del usuario o una combinación de estas. En cada caso existe una gran posibilidad de error, por ejemplo, el usuario puede ser incapaz de expresar adecuadamente sus necesidades; las necesidades del usuario pueden ser mal interpretadas o puede haber una falla al reconocer los problemas del usuario. Los errores que ocurren en esta etapa son extremadamente costosos.

2.- Este proceso involucra la traducción de requerimientos del usuario en objetivos para el producto de Software. Aunque este paso no implica una cantidad significativa de traducción, esto involucra la identificación y el sopesar una cantidad considerable de decisiones. Aquí los errores de Software pueden introducirse durante este

paso por malinterpretación de requerimientos, falla en identificar algún "tradeoff" por hacerlo de forma incorrecta y fallando en establecer los objetivos necesarios, los cuales generalmente no se encuentran de manera explícita en los requerimientos del usuario.

3.- El tercer paso se refiere a la transformación de objetivos en una especificación externa, una descripción precisa de la conducta completa del sistema desde el punto de vista del usuario. En términos de la cantidad de traducción que debe ocurrir, este paso es el más significativo en el desarrollo del Software. Esto hace a este paso el más propenso a los errores tanto en número como en gravedad.

4.- Este paso representa un gran número de procesos de traducción los que empiezan con una descripción externa de el producto y terminan produciendo un diseño detallado el cual describe un gran número de enunciados de programa los cuales deben ser ejecutados a fin de implantar un sistema el cual se comporta de la manera especificada. Este paso incluye procesos tales como la traducción de descripciones externas a una estructura de componentes de Software y la traducción de estos a una descripción procedural. Debido a que en cada paso se manejan mayores volumenes de información acerca del

producto de Software, la posibilidad de introducir errores de Software durante estos procesos es extremadamente grande.

5.- El último proceso de diseño es la traducción de la lógica de la especificación del Software en los enunciados del lenguaje de programación. Aunque aquí se pueden generar un gran número de errores de Software, estos errores tienden a tener un efecto menos fácil de detectar y corregir. Un segundo tipo de traducción ocurre durante este proceso, la traducción de el lenguaje fuente a el código objeto. Esta traducción es llevada a cabo normalmente por un programa independiente llamado compilador.

6.- Un proyecto de Software tiene dos productos: el Software propiamente dicho y los documentos que describen el uso de él. Los documentos normalmente toman la forma de manuales escritos, aunque pueden ser almacenados dentro de la computadora. La Documentación es usualmente escrita, llevando a cabo una traducción de las especificaciones de Software externas en material que está mas orientado a los usuarios.

La Documentación tiene gran influencia en la confiabilidad del Software ya que si se cometen errores al escribir los documentos, estos no describirán el

comportamiento del Software. El usuario al leer los documentos y usar el Software descubrirá que este no se comporta como es de esperarse, condición que fue definida como error. Por supuesto aún y cuando concuerden el Software y su documentación podría existir una condición de error generada antes de este punto.

7.- Durante el proceso de desarrollo, las especificaciones para el Hardware del sistema sirven como otra forma de entrada, por ejemplo al diseñar un sistema operativo se depende de la descripción del Hardware y todos los periféricos conectados al equipo. El diseñador de aplicaciones depende frecuentemente de la especificación de terminales y líneas de comunicación. Toda mala interpretación de estas especificaciones resultará en un error de Software.

8.- El Software de aplicación frecuentemente interactua con el Software anfitrión en el Sistema de cómputo para servicios tales como entrada-salida y asignación dinámica de recursos. La mala interpretación de los servicios de Software anfitrión es otra fuente de error.

9.- El producto de Sofware es expresado como enunciados fuente en al menos un lenguaje de programación. Una mala interpretación en la sintáxis o semántica de el lenguaje de programación es otra fuente de error.

10.- Hay dos formas de comunicación entre el usuario y el producto de Software: la documentación describiendo el uso del producto y el uso del producto en sí mismo. Este paso representa a el usuario leyendo las publicaciones y traduciendo su contenido en una comprensión de como él desea utilizar el Software.

A fin de entender como la confiabilidad es afectada por este proceso, asumimos que tenemos un producto de Software con un número de errores desconocidos. Si el usuario intenta llevar a cabo una función usando el producto y no puede determinarlo adecuadamente examinando la publicación, él puede intentar experimentar con el sistema. Estos intentos de usar el sistema en forma no esperada frecuentemente incrementa la probabilidad de encontrar los errores de Software aún ocultos. De aquí que la calidad de la documentación del usuario, aunque no tiene que ver con el número de errores de Software presentes en el producto, puede afectar la confiabilidad del producto.

11.- Este paso representa la interacción directa del usuario con el Software, por ejemplo al introducir datos por la terminal y examinar la salida. Si los factores humanos no fueron tomados en cuenta, la probabilidad de que el usuario cometa un error es grande. Los errores cometidos por el usuario frecuentemente llevan a condiciones no esperadas previamente por el sistema, incrementando la posibilidad de encontrar más errores.

12.- Como se mencionó anteriormente, hay muchos lazos de realimentación en el diagrama, sin embargo, uno de estos lazos es sumamente importante.

Un porcentaje considerable de los esfuerzos en el mundo del Software involucra la modificación del Software existente. Aunque el modelo de traducción que se esta analizando, es también una buena representación de este proceso, un paso principal en el proceso de modificación, es la lectura y comprensión del Software existente a fin de comprender cuándo y cómo llevar a cabo estos cambios. Esta actividad ocurre durante el mantenimiento. Debido a esto, el Software en sí mismo no es usualmente el fin del proceso de traducción; la gente tiene que traducir el Software para corregirlo y agregar nuevas funciones. Obviamente en este paso se pueden introducir errores, es por lo

tanto importante poner atención en las características del lenguaje de programación así como en el estilo con el que se lleve a cabo.

Este modelo es importante ya que describe las causas fundamentales de la falta de confiabilidad. A fin de obtener una mejor comprensión de los procesos de traducción analicemos el modelo microscopico, que involucra los procesos que se llevan a cabo en la persona, para llevar a cabo esto realizamos cuatro pasos básicos.

1.- Se recibe la información usando los mecanismos de lectura, que involucran las áreas del cerebro que controlan los ojos y el oido.

2.- Se almacena la información en la memoria.

3.- Se recupera esta información de la memoria, se recupera información adicional de la memoria describiendo el proceso de traducción, se lleva a cabo esta y se envia la información al mecanismo de escritura.

4.- La información se plasma fisicamente por medio de la escritura y del habla.

Aunque este modelo es un tanto simple, aquí podemos localizar varios de los puntos de sobrecarga cognoscitiva en los cuales es muy fácil caer!

Una de las grandes habilidades de la mente humana es su habilidad para identificar patrones que recibe comparándolos con la gran cantidad de información almacenada proveniente de su educación y experiencias anteriores. Este es el principio atrás de la habilidad para leer entre líneas, comprender oraciones que son incorretas gramaticalmente, etc. Desafortunadamente, esta poderosa habilidad es la causa de muchas impresiciones en la traducción que nos llevan a errores de software, y estas mismas habilidades son las que llevan a la corrección de errores. Problemas tales como el sobrecargar la memoria temporal exigiendo el manejo de un número de símbolos superiores de que se puede manejar confiablemente, exigiendo el aprender y memorizar gran cantidad de información sin contar con los apoyos adecuados como son símbolos mnemónicos, o la terminología ya en uso por el usuario o la persona involucrada en el proceso de traducción.

De lo anterior se deduce que una manera de aumentar la confiabilidad y la eficiencia del proceso de traducción, en minimizar la complejidad y número de pasos de taducción necesarios para el usuario, de forma tal que la imagen que este persive del sistema de cómputo sea la de un lenguaje afín a su

problemática, en vez de un sistema que lo forze a llevar en la mente una gran masa de información. Estos lenguajes representan un nivel linguistico superior el cual para una implantación practica debe estar basado en una serie de máquinas abstractas. La mayoria de los programas usan únicamente dos niveles, un lenguaje de alto nivel (FORTRAN ó COBOL) y el otro el lenguaje de máquina, aunque algunas veces se reconoce un nivel informal como es un diagrama de flujo.

Las Máquinas Abstractas como Niveles Linguisticos

Una jerarquía linguistica está definida por sucesivos niveles de software en donde cada nivel de esta jerarquía es un sistema de computadora caracterizado por los tipos de dato y operaciones primitivas del lenguaje. Cada nivel es o potencialmente es el sistema anfitrión para la definición de un nuevo nivel linguistico a través de añadir más sistemas de software.

La jerarquía es una herramienta de la Ingeniería de Software la cual si es usada adecuadamente, permite que los componentes de varios niveles puedan ser diseñados y desarrollados por separado. Por supuesto, el desarrollo independiente de niveles del sistema es posible únicamente si los lenguajes corresponden a los umbrales entre las capas de software que han sido especificadas correctamente y estan de acuerdo. Para tener éxito al implementar un sistema de software no debe ser necesario alterar cualquier componente del lenguaje anfitrión. Tal necesidad puede exponer incompletitud e ineficiencia en el lenguaje anfitrión para los objetivos del sistema de software.

Se distinguen tres técnicas para definir un nuevo nivel linguistico para un sistema de software: extensión, traducción e interpretación. Frecuentemente se emplea una combinación de las tres técnicas.

Extensión. Al definir un nuevo nivel linguistico por extensión procedural, el sistema de software añadido a el sistema anfitrión es simplemente una colección de procedimientos que expresan las operaciones primitivas de el nuevo nivel en términos de las operaciones primitivas del sistema anfitrión. Se implantan nuevos tipos de dato o clases de estructura en esta forma, poniendo al alcance del usuario un sistema extendido además de las primitivos y tipos de dato del sistema anfitrión. Al usar extensiones la representación interna para procedimientos a ambos niveles, anfitrión y nuevo son idénticas sintáctica y semánticamente.

Traducción. El definir un nuevo nivel linguistico por traducción consiste en escribir un compilador que corra en el sistema anfitrión y traduzca los programas escritos en el nuevo nivel linguistico en programas en el lenguaje del sistema anfitrión. La necesidad de compilación como un paso en correr un programa es caracteristica de esta técnica. Las representaciones del programa expresadas en el lenguaje de el nuevo nivel no son directamente ejecutables.

Interpretación. Definir un nuevo nivel linguistico por interpretación consiste en escribir un interprete para el lenguaje de el nuevo nivel en términos de los tipos de dato y operaciones primitivas de el sistema anfitrión. Los programas

en el nuevo nivel linguistico son representados en forma directamente ejecutable.

La definición del nuevo nivel puede o no limitar el acceso de el usuario a parte o a todas las caracteristicas linguisticas del anfitrión. La diferencia entre el uso de extensión e interpretación es que en extensión las primitivas de el sistema anfitrión continuan accesibles en el nuevo nivel mientras que este no es el caso cuando se utiliza interpretación.

Debe observarse que a la extensión procedural no se le debe considerar como la creación de un nuevo nivel linguistico, a menos que los procedimientos agregados se agrupen en forma tal que se definan relaciones jerarquicas. Este es el caso de los lenguajes de comandos del sistema operativo en el que una colección de procedimientos define un nuevo nivel linguistico y comunmente se previene a los usuarios de no utilizar procedimientos fuera de la colección siendo entonces esta colección básicamente un interprete del nuevo nivel linguistico.

La traducción e interpretación son básicamente diferentes en el siguiente aspecto: dos compiladores para distintos lenguajes fuente, se implantaron para el mismo sistema anfitrión, produciendo procedimientos compilados en el lenguaje del anfitrión. Si se usa un estandar para las interfaces con

el anfitrión, entonces los programas expresados en los dos lenguajes fuente pueden ser operados juntos con éxito. En contraste, la interpretación es usualmente llevada a cabo debido a la necesidad de utilizar una forma fundamentalmente diferente de organización de datos en el anfitrión, o para obtener la capacidad de monitorear y controlar el programa de manera imposible al nivel del anfitrión. Esto es el nivel del anfitrión es incompleto para los objetivos de el sistema de software. Si los interpretes para dos lenguajes fuente se escriben en el lenguaje del anfitrión, entonces la comunicación entre procedimientos en los dos lenguajes puede ser dificil si no imposible, a menos que se realicen cuidadosos planes de coordinación para la implantación. Cada interprete usará una representación completamente diferente para la representación de tipos de datos equivalentes, de aquí que cada llamada a un procedimiento expresado en el otro lenguaje causará un cambio de interpretes y traducción de todos los datos a ser comunicados.

El diseño y construcción de un sistema de software es fundamentalmente la creación de una descripción completa y precisa de el sistema. La descripción de un sistema de software es una colección de descripciones de sus componentes de software y hardware.

La descripción completa y precisa de los componentes del sistema es en realidad un programa expresado en un lenguaje de programación bien definido. Si este lenguaje es el lenguaje del sistema anfitrión o la traducción de el programa a el nivel linguistico definido por el anfitrión es una operación rutinaria; entonces la preparación de los programas completa el proceso de construcción de los componentes del sistema. De otra forma la implantación de la componente del sistema esta incompleta hasta que una representación correcta del componente se prepare en el nivel linguistico de el sistema anfitrión.

Además de lo anterior requerimos dos descripciones más: una descripción de el sistema anfitrión y una descripción de el nivel linguistico del sistema de software que se pretende realizar. La semántica de el nivel linguistico de el sistema anfitrión debe ser conocida antes de que los componentes del software exterior puedan tener representaciones exactas. Por supuesto, los objetivos de el sistema deben ser conocidos antes de que todos los componentes puedan ser especificados.

Lenguajes y Niveles de abstracción.

Cada nivel de abstracción en un sistema ordenado jerarquicamente introduce un nuevo lenguaje de programación. El esqueleto de este lenguaje esta dado por el catálogo de operaciones admisibles en tal nivel. Otros conceptos, tipos de

dato, recursos etc. son introducidos como los atributos de parametros de estas operaciones. El conjunto de operaciones puede ser visto como el conjunto de instrucciones de una computadora y es esta visión la cuál nos lleva a el término 'máquina abstracta'. Por supuesto para ser una herramienta de programación conveniente el lenguaje debe tener algo mas.

La consideración de niveles de abstracción como lenguajes de programación introduce un conjunto de criterios, los cuales son de gran importancia en el desarrollo de software de aplicación.

Como se expuso anteriormente el lenguaje de programación correspondiente a una máquina abstracta A (i + 1) puede obtenerse a través de tres técnicas diferentes a partir de el lenguaje correspondiente a Ai: extensión procedural, traducción e interpretación.

El objetivo principal de un nuevo lenguaje en la jerarquía es la introducción de las operaciones nuevas, tipos de dato y estructuras de dato correspondientes a la nueva máquina abstracta. Mediante el uso de traducción e interpretación se puede lograr protección contra el mal uso de herramientas que ya no se tengan al nivel A(i + 1).

PARADIGMAS DE PROGRAMACION

Quien solo se adelanta a su época,
será alcanzado por ella alguna vez.

Ludwig Wittgenstein, 1930

CLASIFICACION PARADIGMATICA DE LOS LENGUAJES DE COMPUTADORA

La idea de aplicar los paradigmas de Kuhn [KUHN62] a disciplinas relacionadas con las Ciencias de la Computación no es nueva, Walker [WALK 81] la utilizó con éxito en la administración de proyectos de Software, incluso en el caso específico de los lenguajes de programación este enfoque ha sido planteado en el número de enero de 1986 de la revista "Software" del IEEE; el editor en la itroducción nos comenta que el término paradigma aunque es muy usado en la actualidad, tiende a quedar en desuso, debido a que su significado es un tanto incierto. Aquí se retomará el término paradigma y para evitar la problemática que acertadamente comenta el Dr. Shriver se delimitará el contexto y la razón por la cual se hablará en términos de paradigmas de programación.

Si buscamos el término paradigma en el diccionario, se encuentra que significa modelo, patrón o ejemplo, lo que se encuentra en términos literales cerca de la idea que nos plantea Tomas Kuhn, un historiador de la ciencia quien a través del concepto de paradigma explica el progreso y proceso de la ciencia [KUHN 62]. De acuerdo con Kuhn un paradigma es un

'modelo aceptado o patrón'. De esos modelos evolucionan tradiciones coherentes y específicas de la investigación científica. Dentro del trabajo original publicado por Kuhn algunos estudiosos han encontrado que el término paradigma es usado en diversos contextos con significados aparentemente distintos, sin embargo como lo aclara el propio Kuhn es un trabajo posterior [KUHN 77] la escencia de la utilidad del concepto de paradigma descansa en que la Ciencia o la Ingeniería o en términos generales cualquier disciplina normal, están basadas en logros anteriores y estos logros son reconocidos y suministan el fundamento para llevar a cabo la práctica subsecuente de la disciplina. El poder de un paradigma viene de su éxito para forzar la realidad en un patrón preformado relativamente inflexible el cual el paradigma modela. Por lo tanto, un paradigma debe se capaz tanto de atraer y conservar un grupo de seguidores como de motivar al grupo de adeptos a emplearlo constantemente en las actividades de su disciplina.

Los paradigmas tienen éxito y ganan una posición conforme parecen mejores para resolver los problemas, que sus competidores. Sin embargo es importante notar que el poder de un paradigma está intimamente relacionado con su capacidad para enfocar cierto tipo de problemas, Kuhn también nos plantea que un paradigma lleva consigo la semilla de su propia destrucción ya que conforme aparescan nuevos fenómenos dentro de la

disciplina los cuales el paradigma no pueda enfrentar, se crea una crisis, la cual solo podrá ser resuelta cuando se cuente con un paradigma alternativo el cual resuelva las anomalías del anterior. Sin embargo, la adopción de un nuevo paradigma puede no necesariamente hacer que desaparesca el anterior, o sea que bajo ciertas condiciones varios paradigmas pueden substituir en la misma disciplina, tal vez sujetando al menos poderoso a ciertos supuestos los cuales en caso de no darse requieran forzosamente la utilización del otro paradigma, este caso se puede dar en las Ciencias Naturales. En el caso de las Ciencias Sociales, la situación aparentemente es más compleja ya que incluso por largo tiempo pueden subsistir en paralelo diversos paradigmas contrapuestos, los cuales son defendidos por diversos grupos en la disciplina.

Aparentemente el caso de los lenguajes de programación tiende a comportarse dentro del segundo caso lo que da como consecuencia la precencia de diversos paradigmas simultáneamente.

Aunado a lo anterior es importante hecer referencia a las ideas del Arquitecto Kenneth E. Boulding [BOUL 56] quien ha hecho interesantes observaciones sobre la relación entre la conceptualización y la conducta, y quien sostiene que "la conducta depende de la imagen y dicha imagen se construye como resultado de las experiencias anteriores del que posee la imagen", y esta imagen puede ser afectada a través de mensajes

los cuales consisten en información y siendo el significado de este mensaje el cambio que se lleve a cabo sobre la imagen. Básicamente nos dice Boulding, estos mensajes pueden ocasionar la creación de una nueva imagen la cual puede ser identica a la anterior, sufrir un cambio definido, ser reorganizada o revolucionada, todo esto implica que si el sujeto es continuamente bombardeado con información hostil seguramente aumentará su incertidumbre y su imagen sufrirá frecuentes cambios. Esto es de particular interés en el caso de los lenguajes de programación ya que la gran diversidad de paradigmas y la intensa actividad en el campo de los últimos años, han dejado en la mayoria de los profesionales una intranquilidad respecto a validez de el lenguaje o tipo de lenguaje que utilizan, propiciando un ambiente voluble y en el que muchas veces se toman decisiones precipitadas.

Como se expondrá en seguida existe un fuerte vinculo entre los paradigmas de programación y los mecanismos de abstracción empleados, principalmente se puede concebir que tanto un paradigma puede inducir un lenguaje como que cierto lenguaje puede inducir un paradigma.

PARADIGMAS INDUCIDOS POR EL LENGUAJE

Lenguaje de Programación: La Sintáxis y Semántica (terminología y significado) que define una notación particular usada para comunicar una solución de un problema a una computadora.

Paradigma de Programación: Una vista abstracta de una clase de lenguajes de programación que describe los medios de uso de resolver problemas de programación. Por ejemplo, tanto Pascal como Algol forzan el uso de programas secuenciales estructurados en bloques de aquí que ellos pertenescan a el mismo paradigma.

LENGUAJES INDUCIDOS POR PARADIGMA

Paradigma de Programación. Una manera de enfocar un problema de programación. Una manera de restringir el conjunto de solución. Por analogía, la programación estructurada restringe al programador a no usar todas las construcciones no estructuradas con los que cuenta un lenguaje de programación convencional. Un paradigma permite al programador usar únicamente un conjunto restringido de conceptos.

Lenguaje de Programación: Una combinación de uno o más paradigmas junto con la sintáxis concreta y semántica para implantar las nociones de aquellos paradigmas.

Actualmente teniendo los modelos abstractos que existen atrás de los lenguajes concretos, se puede diseñar un lenguaje nuevo

partiendo de los conceptos fundamentales que le darán forma, siendo por tanto imprescindible para el diseñador de un lenguaje tener conocimiento de los paradigmas y las caracteristicas de estos pueden ayudar en una aplicación especifica.

Taxonomía de los Paradigmas de Programación.

La clasificación de los distintos paradigmas es una labor sumamente compleja, ya que siendo el punto de partida para esto contar con la enumeración y definición de todos ellos lo que es difícil en el estado actual del campo; por tanto sin tratar de calificar esto como exaustivo se propondrán las categorias más comunes en la literatura y se darán ejemplos de algunos lenguajes que entren en estos rubros, debe hacerse notar que tal vez haya un lenguaje que encaje en varias categorías y el caso de otro que parcialmente encuadre en varias, esto no es un problema sino más bien el objetivo de ofrecer varios paradigmas dentro de un mismo lenguaje.

Los paradigmas de programación comunmente usados son imperativo, procedural, aplicativo, Lambda-free, orientado a estructura de datos, orientado a objetos.

Imperativo: El estilo de programación imperativo incluye comandos tales como asignación y ramificación. Los lenguajes que soportan este estilo tienen los medios para evaluar formulas

(expresiones) involucrando computaciones lógicas y aritméticas. Estos lenguajes tienen construcciones cercanamente relacionadas a los conjuntos de instrucciones que se encuentran tipicamente en una arquitectura Von Neumann. Ejemplos de estos lenguajes son el Ensamblador y el Basic.

Procedural: El estilo procedural de programación incluye facilidades de programación imperativas, pero suplementando estas con un mecanismo de abstracción para construir procedimientos que generalicen los conceptos de un comando y una expresión. Este también incluye el uso de construcciones de control para lograr iteracción y selección, ejemplos de estos lenguajes son Algol, Fortran, PL/1, Pascal, C, Euclid y Ada. Los más recientes también permiten la definición de nuevos tipos de dato.

Los lenguajes procedurales proveen la capacidad pa usar el poder de una arquitectura Von Neumann sin tener que especificar los detalles asociados con una computadora particular. Ellos forman el caudal principal de los lenguajes de programación.

Aplicativo: El estilo de programación aplicativo usa la aplicación de funciones y definición recursiva de funciones como los medios principales de la computación. El más importante de los lenguajes de esta clase es el Lisp. El Lisp ilustra que las operaciones infix de la aritmética tales como +

o * son interpretadas naturalmene como una aplicación de alguna función, así, las expresiones aritméticas de los lenguajes imperativos y procedurales son aplicativos en naturaleza.

Los mecanismos de abstracción de los lenguajes procedurales proveen la misma capacidad semántica de las formas Lambda de Lisp. La principal diferencia es la naturaleza dinámica de las estructuras de datos Lisp comparadas con las estáticas en la mayoría de los lenguajes de programación procedurales.

Así, mientras los lenguajes procedurales pueden ser usados en un estilo aplicativo, la capacidad de manipulación de datos restringida de estos lenguajes limita su poder expresivo cuando son usados de esta manera. El hecho de que tales lenguajes usualmente no pueden pasar estructuras de datos completas como argumentos o regresarlas como resultados es la principal fuente de limitaciones.

Los vínculos entre conceptos aplicativos e imperativos ha sido un punto central en el estudio de la semántica formal de los lenguajes de programación. Las matemáticas de la interfase entre comandos y expresiones es complicada, ya que los enfoques sencillos que pueden tomarse para ya sea un lenguaje puramente aplicativo o uno puramente imperativo no pueden ser usados cuando se combinan los dos estilos.

Este hecho ha llevado a el estudio intensivo de técnicas para evitar totalmente las construcciones imperativas y para diseñar lenguajes puramente aplicativos. Ejemplos son el ISWIM y Lucid.

Lambda-free. Este estilo de programación va más allá y limita el uso de mecanismos funcionales a dos niveles, llamados funciones en datos y formas de combinación que constituyen funciones a partir de otras.

Los mecanismos para denotar funciones en el lenguaje FP de Backus, no incluyen el mecanismo de abstracción procedural del cálculo Lambda. La motivación para esta limitación es que se proclama, elimina la dependencia de la arquitectura Von Neumann. Más recientemente, Turner ha investigado un lenguaje aplicativo en el cual las formas Lambda son compilados usando combinadores.

Orientado a Estructura de Datos. Emplea una solución por el camino de una estructura de datos poderosa tales como listas (LISP), arreglos (APL), conjuntos (SETL) y Bases de Datos Relacionales (SQL). A la fecha la importancia de estos enfoques particulares los han llevado a conformar un paradigma para cada una de estas estructuras. Mismos se detallan a continuación.

Orientado a arreglos. El estilo de programación orientado a arreglos usa estructuras de datos como los valores en el dominio

de datos y tiene operaciones que mapean estos valores como un todo. El APL es el principal lenguaje de este tipo. Tiene funciones de segundo orden, llamados operadores, esto es similar a las formas de combinación de FP.

Debido a que APL contiene unicamente construcciones imperativas de control primitivas, su naturaleza altamente aplicativa es frecuentemente ignorada.

Relacional. En el estilo relacional de programación, el programador provee una descripción de un problema y un interprete fundamental deduce la solución basada en alguna semántica presumida. El ejemplo primario de este enfoque es Prolog, el cual usa una forma restringida de enunciados de la lógica predicada para expresar el problema.

La interpretación esta basada en el principio lógico de resolución, usando una busqueda en el árbol de implicación. La programación es lógica con estilos aplicativos e imperativos para proveer la flexibilidad requerida para construir sistemas prácticos que exhiban inteligencia.

Orientados a Objetos. El estilo orientado a objetos de programación usa el paso de mensajes entre objetos que tienen un estado y pueden llevar a cabo acciones como el medio primario

de programación, Smalltalk es el lenguaje principal diseñado desde este punto de vista.

Existe una dualidad entre el paso de mensajes y la aplicación de funciones; y es que la aplicación de una función a un argumento puede ser vista como mandar el nombre de la función como mensaje a el argumento visto como un objeto. En general, un mensaje es acompañado por parámetros que son otros objetos, y uno puede ver los mensajes como funciones que son genéricos en su primer argumento.

La semántica del paso del mensaje en el sentido del Smalltalk puede ser simulado en un lenguaje aplicativo que pueda manejar explicitamente los medio ambientes o en uno en uno en el cual el ámbito lexicológico es usado con medio ambiente retenido. El primer enfoque se ha tomado en Zeta Lisp y el segundo ha sido ilustrado por Schema, una variante de Lisp con ámbito lexicológico.

En APL y en lenguajes orientado a objetos, el papel de los datos es más central que en los lenguajes procedurales. En APL, las operaciones son manejadas por datos; esto es, el perfil de los datos determinan el modo en que la operación se comporta. En Smalltalk, los datos determinan como un mensaje es interpretado. Los lenguajes procedurales están más involucrados con proveer medios para llevar a cabo la manipulación de datos que con las estructuras de datos como objetos.

La mayoría de los programadores trabajan en un lenguaje y usan un solo estilo de programación. Ellos programan en un paradigma forzado por el lenguaje que usan. Frecuentemente ellos no han sido expuestos a fórmas alternativas de pensar acerca del problema, y por esto tienen dificultad en ver la ventaja de seleccionar un estilo más apropiado para el problema en cuestión. Si ellos desean usar otro estilo, ellos usualmente tienen que usar otro lenguaje.

Debido a las dificultades inherentes en administrar y combinar el software escrito en diferentes lenguajes, hay poco incentivo para buscar el estilo más adecuado para un problema en particular.

Lenguajes Relacionales: Limitaciones y Perspectivas

El éxito del paradigma Relacional para las aplicaciones en Base de Datos ha llevado a los investigadores a considerar su uso en otras aplicaciones como Base de Datos para diseño por computadora, Bases de conocimiento y modelado de interfase con el usuario.

Estas aplicaciones no tradicionales del Modelo Relacional han llamado la atención a varias debilidades en el modelo. Un enfoque prometedor para tratar con estas debilidades es la

incorporación de paradigmas de lenguajes alternativos dentro de los lenguajes relacionales.

Entre las extensiones propuestas se tienen las relaciones anidadas y la programación funcional.

Recientemente los investigadores han tratado de aplicar el paradigma del lenguaje relacional y el modelo de datos relacional a aplicaciones que no estan basadas en el estilo de las aplicaciones de procedimiento de datos tradicional. Estas aplicaciones incluyen:

Interfase de usuarios.

Elementos del medio ambiente de un usuario de la computadora, tales como directorios y buzones son modelados como relaciones. El usuario interactua con su medio ambiente usando lenguajes relacionales de alto-nivel.

Base de Datos de diseño

Un medio ambiente de diseño puede ser modelado como una Base de Datos relacional. Esto requiere el almacenamiento de componentes de diseño en la Base de Datos. Entre los medio ambientes de diseño siendo

modelados usando relaciones estan el CAD y el Desarrollo de Software.

Estos esfuerzos de investigación han puesto de manifiesto los siguientes problemas del Modelo Relacional:

a) La uniformidad de las relaciones (esto es una relación contiene un conjunto de tuples de un formato particular) es una restricción demasiado fuerte para tipos de dato como, datos de diseño, datos de imagenes y datos de audio.

b) Ciertas restricciones en el conjunto de relaciones admisibles que son aceptables en las aplicaciones de procesamiento de Base de Datos no son supuestos razonables para aplicaciones no tradicionales.

Lenguajes de Especificación

Este tipo de lenguajes nos permite expresar la especificación de un problema formalmente, este problema podria ser un programa de computadora que se desea implantar en un lenguaje de bajo nivel por ejemplo C.

Al preocuparnos por los lenguajes de especificación adquiere importancia el clasificarlos como operacionales o funcionales. Una descripción operacional es aquella en la cual la conducta del sistema es definida en términos de un programa abstracto. La descripción funcional es aquella que define la conducta de acuerdo con algún vinculo lógico entre los argumentos y el resultado de los operadores.

El problema con las especificaciones operacionales esta en la posibilidad de sobreespecificación ya que no solo se define que operaciones realizar sino también como llevarlos a cabo. Aspectos de lo que comunmente se consideran detalles de implantación entran en la descripción.

Sin embargo, las descripciones operacionales tienen ventajas como el hecho que la forma de tales descripciones es familiar al usuario y puede ser facilmente entendida. Las especificaciones funcionales, en especial las más abstractas no son tan ampliamente aceptadas como sus contrapartes operacionales. En ciertos dominios las descripciones funcionales se han hecho comunes; por ejemplo en el uso de axiomas y prueba de reglas para definir la semántica de los lenguajes de programación. En el caso de entidades mas complejas como los sistemas operativos, diversas formas de especificación funcional, relativamente menos austeras se han propuesto, como la definición de Módulo de Parnas [PARN 72].

Descripción Procedural y No procedural

La mayoría de los lenguajes actuales son procedurales. Esto es existe un ordenamiento implícito en la ejecución de los enunciados y este ordenamiento es impuesto por la estructura del texto en el programa. Asociado con esta estructura se tiene un contador implícito de instrucciones que durante el transcurso de la ejecución de el programa, automaticamente se incrementa excepto cuando se presenta un cambio en el flujo de control.

Los lenguajes procedurales son una consecuencia de la llamada arquitectura tipo Von Neumann. Junto con el modelo en si mismo, se ha mantenido como el paradigma dominante en el diseño de los lenguajes de programación. Hay sin embargo varias áreas de los sistemas de computo, en los que la computación procede en forma noprocedural. Por esto se pretende decir que el orden de los enunciados no implica necesariamente el orden de su ejecución. Así no existe un contador el cual sea automaticamente incrementado o modificado conforme la ejecución se lleva a cabo. En vez de esto, una acción dentro de la computación noprocedural se lleva a cabo, cuando y solo cuando se satisface una condición particular.

Los sistemas y computos no procedurales se encuentran en gran variedad de formas. En el campo de la inteligencia Artifical, por ejemplo un modelo particular de procesamiento de información toma la forma de sistema de producción el cual escencialmente describe programas no procedurales. En los últimos años una nueva clase de computadoras, llamadas máquinas de flujo de datos, han sido propuestas, donde una unidad operacional es activada cuando todos los valores de entrada necesarios para la operación ya se tienen disponibles en las líneas de la unidad. En el contexto del Hardware, los computos no procedurales tienen un interés particular, ya que los sistemas digitales han sido y pueden ser ampliamente modelados y diseñados como máquinas de estados.

Lenguajes de Muy Alto Nivel

Un lenguaje de programación de muy alto nivel difiere de uno de alto nivel, en que el primero permite al programador usar términos y construcciones que están más cerca de la terminología informal y forma de hablar comunmente usada en la discusión de algunas clases de problemas importantes.

Hay razones para creer que la investigación llevada a cabo actualmente en los lenguajes de muy alto nivel nos lleva a transponer el umbral de una nueva técnica de programación. Estos nuevos lenguajes están enmarcados por las siguientes caracteristicas:

a) En general un lenguaje de muy alto nivel es aquel que va más allá de los lenguajes de alto nivel comunmente aceptados FORTRAN, PL/1, ALGOL-60 etc., reduciendo la distancia entre el enunciado inicial del problema y su solución eventual.

b) Siempre que sea posible, un lenguaje de muy alto nivel tratará de permitir al programador especificar "qué" desea hacer, en vez de "cómo" se va a obtener. Por ejemplo, en un lenguaje de muy alto nivel se tendría

Resuelva para X y Y

X = 3Y + 7

X = 9Y - 8

Notece que esta instrucción esta matemáticamente definida, aún y cuando menos explicita en términos de un lenguaje procedural. Como lo han planteado Sammet y Leavenworth, "un programa no procedural es una receta para resolver un problema, sin preocuparse por los detalles de como será resuelto"

c) Un lenguaje de muy alto nivel ofrecerá un conjunto de operadores de agregación, los cuales implicarán un

número indefinidamente grande de operaciones a nivel máquina. Un lenguaje que provee operadores de agregación toma parte de la carga cognocitiva del programador permitiendole pensar en sus problemas en términos de agrégados en base a los cuales puede construir agregados más complejos.

d) El permitir referencias asociativas, nos da la facilidad de accesar los datos simplemente, a través del establecimiento de una propiedad que caracterice el dato o datos deseados. En contraste podemos analizar lenguajes como el FORTRAN en el cual se tiene que direccionar directamente el dato a usar. La capacidad de referencia asociativa es también una caracteristica importante de los lenguajes de recuperación de Base de Datos. Estos lenguajes tipicamente admiten enunciados asociativos del tipo, "lista los alumnos de la carrera de computación y de la generación 1985", utilizando de aqui la propiedad " de la carrera de computación" y "generación 1985", para construir una lista que únicamente podria ser integrada en un lenguaje de alto nivel a través de un procedimiento de busqueda complejo el cual seguramente requeriria navegación.

e) Se busca la eliminación de una secuencia arbitraria, los lenguajes como FORTRAN forzan a que el ordenamiento

de todas las operaciones se especifique en detalle, aún y cuando la especificación de algunas de ellas sea irrelevante. La naturaleza de esta restricción esta en que tales lenguajes nos llevan a escribir una lista inherentemente desordenada de 'cosas por hacer', en un estilo artificial, en el cual no solo se especifica que hacer sino también el orden en el cual esto debe llevarse a cabo de muy alto nivel en contraste, permitirá que los enunciados se escriban en cualquier orden.

f) Programación no deterministica esto implica un poderoso mecanismo semántico el cuál involucra busquedas complejas, por medio de la especificación de una regla para la generación de una nueva posición a partir de la anterior y de una lista de posiciones aceptables. Un lenguaje no determinístico puede generar las posibles rutas de busqueda.

Es importante también comentar otras caracteristicas de estos lenguajes, como son la capacidad de llevar a cabo busquedas guiadas por un patrón, y la habilidad explicita para coordinar procesos múltiples, cada uno capaz de reaccionar a eventos dentro de otros; lo que puede ser importante. Sin embargo, el criterio establecido en primer lugar es el fundamental; un lenguaje es de muy alto nivel si acorta en grado

significativo la distancia entre la formulación y solución de alguna clase significativa de problemas de programación.

METODOLOGIA PROPUESTA

> Actualmente, la diferencia entre un buen arquitecto y uno malo estriba en que éste cede a cualquier tentación, mientras que el primero, le hace frente
>
> Ludwig Wittgenstein, 1930

A fin de enmarcar la metodología adecuadamente se inicia este capítulo definiendo los términos centrales continuando con la definición detallada de los pasos propuestos:

Método: procedimiento ordenado para llevar a cabo una tarea específica

Metodología: Sistema de métodos que aplican los principios básicos del razonamiento a la indagación científica

Metodología de Diseño: Colección de herramientas y técnicas empleadas dentro de un marco organizacional, que pueden ser aplicadas de forma consistente a proyectos sucesivos.

Como Freeman lo señala [FREE 80], un método de diseño es una forma de llevar a cabo el diseño. Mas específicamente esto involucra la identificación de decisiones a ser tomadas, el como tomarlas, así como el orden adecuado para esto. Ejemplos típicos de estos métodos son arriba-abajo, fuera-dentro, dentro-fuera, abajo-arriba etc. Es importante notar que estos

métodos difieren básicamente en la forma en que dirigen el desarrollo del proceso de diseño.

En proyectos que involucran diseños extremadamente complejos como es el caso de gran número de proyectos de Software, enfrentamos el problema que la selección y aplicación de un método único es inadecuado. Dependerá esto del rigor que el diseñador desee imponer al proceso del contexto del problema y de la organización del equipo de diseño. En base a lo anterior será importante contar con uno o mas de las siguientes técnicas:

1.- Una vista formal o modelo de la situación completa del diseño

2.- Un conjunto de herramientas para análisis y documentación

3.- Técnicas de administración y control para el proceso de diseño en general.

Conjuntando lo anterior podemos decir que una Metodología de diseño implica un conjunto de métodos organizado y coherente, modelos, herramientas de descripción y técnicas de administración que pueden ser aplicadas simbióticamente en el desarrollo sistemático y racional de un diseño.

El objetivo de una especificación formal es la descripción de la conducta externa del programa o componente del programa sin describir o restringir su implantación interna. La conducta externa del programa o componente del programa debe incluir:

Que operaciones debe realizar.

Que información se le debe dar.

Que resultados se obtendrán, incluyendo las indicaciones de error.

Los efectos de las operaciones sobre las subsecuentes.

El método de especificación debe ser capaz de definir completamente la conducta externa de el programa, ya que programas que no son comprendidos del todo o en parte pueden provocar efectos laterales. Pero es muy deseable que la especificación de un programa no restrinja la subsecuente implantación de el programa, siempre que su comportamiento externo sea el especificado.

La utilización de métodos de especificación formal es importante por:

La presencia de una especificación precisa provee una interfase sólida entre las partes de un sistema complejo. Los programas que implantan una especificación pueden ser diseñados e

implantados y posteriormente modificados sin considerar el uso que se vaya a hacer de ellos. Simultáneamente los programas que invoquen a otros pueden ser diseñados e implantados sin preocuparse por ningún detalle específico de la implantación. La especificación define completamente las interacciones entre los programas y esconde lo demás.

El rigor de la especificación formal garantiza un diseño más completo, y un mejor enunciado y comprensión de los detalles. Las decisiones importantes no se pueden pasar por alto o tomar a la ligera, así mismo todas las decisiones son claramente documentadas.

La especificación puede servir como una interfase entre el diseñador y el implantador, así como entre el implantador y el usuario.

La especificación provee una referencia para la documentación de el programa, para la construcción de pruebas, para el mantenimiento y desarrollo de el programa así como para la reimplantación del mismo.

Un proceso de diseño se dice que es formal si satisface las siguientes características mínimas:

1.- Existe un método formal o método para describir el diseño.

2.- Existen los principios formales para validar lo correcto del diseño.

3.- El diseño puede ser manipulado y evaluado en sus características de desempeño.

4.- Existe una metodología que se puede enseñar y comunicar.

El corazón del proceso formal de diseño es entonces el contar con una o más fotografías formales del diseño. Fotografías que son objetivas y formalizan la intuitiva imprecisa y subjetiva conceptualización que es característica del diseño informal.

El proceso de diseño formal, está centrado en la descripción. Sin embargo, debe observarse que tan pronto como la idea de descripción formal se introduce el concepto y la posibilidad de una metodología de diseño aparece conjuntamente. En el proceso de diseño informal, la ausencia de una descripción formal casi automaticamente elimina toda posibilidad de metodología; el proceso formal de diseño la admite inmediatamente.

La metodología propuesta consiste de los siguientes pasos:

1) Enunciado del problema.

2) Requerimientos del sistema.

3) Diseño de un sistema notacional adecuado para definir el problema.

4) Especificación del lenguaje.

5) Diseño de la implantación.

6) Implantación.

7) Sistema operacional.

8) Experiencia en el empleo.

Como todo proceso de diseño práctico, involucra iteracción entre las etapas y depende de la experiencia de los diseñadores para predecir las consecuencias de sus decisiones.

Hasta el momento no se conoce ningún método riguroso para describir la actividad de diseño y no se puede tampoco decir como desempeñar la actividad. Sin embargo, se tienen lineamientos, basados en la experiencia, para guiar la actividad de diseño, pero los problemas son de gran tamaño y aún el evaluar la calidad del diseño es difícil.

Actualmente únicamente la especificación del lenguaje la implantación y posiblemente el sistema operacional son objetos formalmente definidos. Por lo tanto, no debe sorprender que en el pasado la atención ha sido dirigida principalmente hacia

estas etapas en el desarrollo de sistemas. Igualmente está claro que el problema y la experiencia en la utilización del producto no estará formalmente definida en el futuro cercano. Sin embargo, las etapas de requerimientos, especificación y diseño de implantación deben ser capaces de definirse de forma precisa y formal.

Enunciado del Problema

La descripción del problema, enmarcado en el medio ambiente operacional, marca el punto de inicio de la metodología. El producto final de esta etapa será una descripción tal vez en forma narrativa en la que se expongan los antecedentes, la situación actual y las perspectivas al futuro tanto conteniendo una descripción de la forma de trabajar manualmente en el sistema , o en el caso de no contar con un sistema la definición de lo que se pretende y como se pretende hacer. En esta etapa es sumamente importante buscar la definición de un mecanismo de expresión formal o una notación estandar para el problema en uso ya en el ámbito de la disciplina.

Debe hacerse notar aquí la importancia de no intervenir en la definición tanto de la terminología como de la notación que el usuario proponga, ya que si el usuario en un momento dado con el fin de ayudar al diseñador utiliza notación o terminología no estandar en la disciplina, el producto no será de la calidad

esperada. Es importante detectar el estilo cognoscitivo y la técnica de resolución de problemas empleada por el usuario, tanto al plantear el problema como al proponer soluciones.

Se debe así mismo solicitar referencias bibliográficas en el estado del arte de la disciplina particular o en general documentos aceptados por la disciplina como tal, a fin de contar con un punto de referencia.

La información se debe ampliar solicitando al usuario detalles acerca de herramientas de computo que haya utilizado, de forma que se pueda identificar la experiencia que dentro de la disciplina se tiene con estos productos, anotando los comentarios que el usuario haga. Todo a fin de detectar el conjunto mental con que cuenta en relación con las herramientas de la computadora, obteniendo así una linea base para la definición del diseño.

Debe tenerse especial cuidado en esta etapa, ya que el objetivo final es diseñar una interfase hombre-máquina, la cual debe ajustar adecuadamente al estilo del usuario. así mismo la detección de los posibles subproductos que permitan administrar el sistema dentro del medio ambiente organizacional deberán ser detectados y descritos en el documento de definición del problema.

Requerimientos del Sistema.

La definición de requerimientos debe expresar lo que el usuario desea del sistema, las ideas expresadas en el enunciado del problema ya organizadas en términos de los efectos que el usuario desea pero sin predeterminar el comando o secuencia de comandos a usar, el formato de las salidas o el manejo de errores.

La posibilidad de realimentación por parte del usuario al diseñador, se habilita a traves del documento de requerimientos en donde el usuario ve plasmados sus necesidades. Esta realimentación puede verse complicada si a el usuario se le dificulta analizar los requerimientos, ya que son por definición abstractos.

Diseño del Sistema Notacional

Un sistema notacional es una herramienta que permite de manera formal expresar una idea y consiste en un sistema de símbolos más regular que el lenguaje ordinario, y mejor adaptado a fin de asegurar la exactitud en la deducción, porque permite lo escencial solamente, el contenido conceptual, en oposición a la importancia dada a la retórica en el lenguaje natural.

Los lenguajes de programación son sistemas notacionales que facilitan la programación. Esto lo logran proveyendo una notación concisa para funciones útiles, operadores de estado y estructuras de control (lo cuales son escencialmente funciones, en funciones y operadores de estado), junto con facilidades para definir nueva notación, a estas últimas los llamaremos mecanismo de abstracción o mecanismos definicionales.

De lo anterior se deduce que la diferencia entre un sistema notacional y un lenguaje de programación, consiste en que la notación sea o no implantada en una computadora. Tomando un par de ejemplos se puede ver esto; primero tenemos el caso de los sistemas notacionales definidos por Frege y Russell. Gottlob Frege (1848-1925) escribió una serie de obras sobre los fundamentos de la aritmética, en su primer libro Begriffssehrift (Escritura de los Conceptos e Ideografía), (1879) construye un lenguaje formalizado del pensamiento puro, el cual en sus propios términos tenía como intención no el representar una lógica abstracta mediante fórmulas, sino expresar un contenido mediante símbolos escritos de una manera más precisa y más clara de lo que sería posible utilizando palabras. Bertrand Russell (1872, 1970) en su obra Principia Mathemática intenta desarrollar un lenguaje simbólico elaborado, basandose en el de Peano. Los sistemas notacionales de estos dos grandes matemáticos no han sido implantados en una

computadora por lo que no se refiere a ellos como lenguajes. Por otro lado Kenneth Iverson propone un sistema notacional para manejo de arreglos que se conoce más bien en terminos del lenguaje de programación APL.

Es difícil pensar que todos los lenguajes de programación tienen su origen en un sistema notacional previamente concevido en forma independiente de la computadora ya que la mayoria de los lenguajes de programación son concevidos pensando directamente en la computadora, sin embargo, el desarrollo de nuevos paradigmas y lenguajes de muy alto nivel nos ofrecen la posibilidad de desarrollar una notación de forma independiente de la máquina y luego implantarla en esta, aprovechando nuestra experiencia en las jerarquias de abstracción y el conocimiento de los estilos cognoscitivos empleado en las distintas áreas de actividad del ser humano.

De la historia de la civilización humana conocemos la existencia de una relación entre la cultura y el lenguaje natural, así como que la cultura está muy relacionada con la forma de pensar. De tal modo que se puede observar una influencia del lenguaje en el pensamiento y esta influencia se conoce como Hipótesis de "Whorf-Sapir": En palabras de Edward Sapir, "los seres humanos no viven el mundo objetivo ni en el mundo de la actividad social como es comunmente entendido, sino que están a la merced de un lenguaje particular el cual ha sido el medio de

expresión para su sociedad", en palabras de Benjamín Lee Worf, "se encontró que la gramática del lenguaje no es solamente un instrumento para reproducir vocalmente las ideas, sino en si mismo un conformador de las ideas, el programa y la guía para la actividad mental del individuo, para su análisis de impresiones, para la síntesis del contenido de su mente. La formulación de ideas no es un proceso independiente, estrictamente racional en el sentido antiguo, sino parte de una gramática particular y difiere entre diferentes gramáticas".

Llevando estas ideas del campo del lenguaje Natural a los lenguajes de programación de computadora, citando a Edsger Dijkstra en su obra Discipline of Programming, "un aspecto importante, aunque también difuso de cualquier herramienta es su influencia en los hábitos de quienes son entrenados para su uso. Si la herramienta es un lenguaje de programación, su influencia es, queramos o no, una influencia en nuestros hábitos de pensar". Como una conclusión de los comentarios anteriores debe tenerse el hecho, de que en cada disciplina se tienen mecanismos de expresión propios que no solo permiten a los distintos profesionales realizar su trabajo, sino que los recién llegados a la disciplina son moldeados en términos de esta terminología, que pasa a integrar una parte fundafmental de su forma de pensar en términos de la propia disciplina, y tal vez podria decirse que este lenguaje propio de la disciplina

hereda la tradición de una generación a otra. Es por tanto vital para reducir la distancia conceptual entre la computadora y la disciplina, el proveer a ésta de un sistema notacional afín a la manera de pensar de los profesionales dentro de la disciplina.

Especificación del Lenguaje

Una vez identificado un sistema notacional que satisface los requerimientos planteados por la aplicación, esta notación se debe refinar en el marco de un lenguaje que se pueda procesar eficientemente en la computadora, esto requiere definir una gramática particular. Una forma de hacer esto podría ser buscando en la literatura una gramática parecida o que se aproxime al sistema notacional, o definir un conjunto de características las cuales tipifiquen un tipo de lenguajes poderosos y sencillos de implantar. Esto último representa la ventaja de que el diseñador ocupará más tiempo en el diseño de las características del lenguaje en particular en vez de diseñando la estructura general del procesador.

Con este fin se toma la metodología de diseño utilizada por los diseñadores del lenguaje Pascal, la cual debido a su generalidad y simplicidad ofrece una gran ayuda para la implantación de traductores.

La Metodología del Pascal abarca tres clases de elementos escenciales:

Medios para descripción del lenguaje.

Reglas para la definición del lenguaje.

Reglas para implantar el traductor.

Estos tres elementos independientes constituyen un sistema de reglas para el desarrollo de un lenguaje y su traductor.

Los medios para la descripción del lenguaje de la metodología del Pascal son equivalentes a la notación de la forma de Backus y Naur (BNF); así ésta nos permite definir lenguajes cuya sintáxis es de contexto libre. Esta difiere de la BNF en que las construcciones sintácticas son definidas en forma gráfica por medio de los llamados diagramas de sintáxis.

El diseñador del lenguaje describe la sintáxis a través de una colección de diagramas. Las ventajas de la notación de diagramas es su simplicidad, lo compacto y la transparencia de la presentación. Los diagramas de sintáxis no solo son de gran ayuda para el usuario, sino lo que es más importante, estos son un medio conveniente para documentar el desarrollo tanto del diseño del lenguaje como del traductor. Esta última característica está intimamente relacionada con el enfoque de programación estructurada empleado.

Las reglas de definición del lenguaje reflejan el supuesto básico hecho por los diseñadores de Pascal; ya que su meta era construir un compilador eficiente de un solo paso; ellos

decidieron usar un algoritmo del parse de arriba a abajo y que tomara un simbolo hacia adelante sin regreso. Esto por supuesto influencia el tipo de lenguajes que pueden ser implantados. Estas caracteristicas del parse se pueden expresar como dos reglas informales:

Regla 1. Para cada producción de la forma,

<V>::=<A >|<A >| |<A >

el conjunto de los simbolos iniciales para <A > {x,y} y para <A > {x,y} no son disjuntos.

Regla 2. Para cada producción <A> la cual genere una secuencia vacia , su conjunto de simbolos iniciales y el conjunto de simbolos el cual puede seguir cualquier secuencia generada a partir de <A> debe ser disjunto.

Como ejemplo las producciones

<V>::=<A>x

<A>::=x|y|

violan la regla 2 ya que el conjunto inicial de simbolos de <A> {x,y} y el conjunto de simbolos que siguen <A> {x} no son disjuntos.

La línea divisoria entre la sintáxis es un tanto arbitraria, aquí se toma la posición de que todos aquellos aspectos de un lenguaje de progamación los cuales se puedan explicar razonablemente en términos de secuencias de símbolos únicafmente, pertenecen al ámbito de la sintáxis y todos los demás aspectos de interés pertenecen a los ámbitos de la semántica y pragmática. Toda técnica de especificación semántica debe en algún sentido asociar un significado a los programas y construcciones de el lenguaje. Los teóricos de los lenguajes de programación han formulado diferentes conceptos de lo que es significado en este contexto, y esta es la razón por la cual hay una amplia variedad de enfoques en la formulación de la semántica, sin embargo, la semántica generalmente ha sido expresada en forma informal en lenguaje natural, aunque actualmente hay una fuerte tendencia hacia la representación formal de la semántica de los lenguajes. Por lo que respecta a la pragmática aún no existe un consenso respecto a la necesidad de especificarla.

Las dos reglas de definición anteriores planteadas en notación BNF pueden ser fácilmente transformados en las reglas correspondientes para los diagramas de sintáxis.

Regla 1. En cada ramificación la rama buscada puede ser seleccionada viendo únicamente al siguiente símbolo en la

rama. Esto implica que dos ramas no comenzarán con el mismo simbolo siguiente.

Regla 2. Ninguna gráfica puede ser recorrida sin leer un simbolo de entrada, entonces esta "rama nula" debe ser etiquetada con todos los simbolos que pueden seguir a A (esto afectará la decisión tomada al entrar en la rama).

Las reglas de traducción plantean la utilización de la metodología para desarrollo de sistemas conocida como refinamiento a pasos o programación estructurada de arriba a abajo. O sea iniciando con la definición general del problema se llevan a cabo un número de pasos de refinación hasta que el último programa resuelve el problema completo.

En este momento ya se tiene definida la forma del lenguaje pero no se ha definido su significado o sea la semántica, esto constituirá nuestro siguiente objetivo.

Implantación

Una vez que se ha definido el lenguaje se procede a su implantación. Con este fin toma primordial importancia los comentarios anteriores respecto a las jerarquías de abstracción fundamentalmente respecto a tres puntos:

Confiabilidad en la Implantación: La importancia de los niveles de abstracción es que estos permiten al diseñador enfrentar la complejidad combinatoria en la que se ve involucrado al construir un sistema en el que se tienen muchos componentes básicos. El precio que se tiene que pagar es el de las especificaciones bien documentadas de las características externas de cada nivel, estas especificaciones corresponden a las interfases abstractas interpuestas entre niveles, tales como las especificaciones entre componentes que interactúan dentro de un nivel llamadas interfases de comunicación. En estos casos si el diseñador escogió bien la interfase, se pueden ignorar los detalles del otro lado de la interfase.

El clasificar las jerarquías de estructura en "actual" y "conceptual" es importante para los efectos de confiabilidad y de esfuerzo de desarrollo; se dice que es "actual" si, cuando el sistema esté en su etapa de operación, las reglas del uso de las interfases de abstracción, son en cierto grado vigiladas. Mientras mayor sea este cuidado mas actual es la estructura. El objetivo de esta preocupación es tratar de prevenir errores que en un momento puedan invalidar la abstracción que se pretende lograr en un cierto nivel. Es por esto que entre más "actual" sea la interfase mayor confiabilidad podremos esperar del sistema.

Portabilidad del producto. Portabilidad es la medida de que tan fácilmente un programa puede ser transportado de un medio ambiente a otro. Si el esfuerzo es mucho menor que el requerido para implantarlo inicialmente decimos que es altamente portable.

Una razón obvia para procurar una gran portabilidad en un programa es la facilidad para transportarlo a una nueva computadora. Una instalación cuyos programas son altamente portables no está atada a un tipo de computadora en particular.

El mecanismo tradicional para incrementar la portabilidad de un programa es usar un lenguaje tal como FORTRAN, ALGOL Y COBOL. Este es un enfoque perfectamente válido, siempre y cuando se satisfagan ciertas condiciones:

Las operaciones básicas y los tipos de dato requeridos se tienen en el lenguaje elegido.

El lenguaje seleccionado tiene una definición estandar y ésta está ampliamente difundida.

Se tenga cuidado en evitar construcciones que no son aceptadas en el estandar, aunque el dialecto local las permita.

Economía en el Desarrollo. La selección adecuada de las máquinas abstractas pueden llevar a una gran economía en los casos de desarrollo del Software, ya que se puede usar Software

con el que ya se cuente y el personal dedicado al proyecto requerirá menos entrenamiento y supervisión, aumentando así la productividad.

En la definición de estos diferentes niveles, es recomendable partir del nivel linguistico más alto, que contenga las primitivas requeridas para el lenguaje que se trata de implantar. Esta recomendación debe evaluarse en términos de las caracteristicas enumeradas anteriormente. Antes de continuar se definirá lo que son las primitivas y el enfoque telescópico para la generación de Software portable.

Primitivas. Las primitivas de un lenguaje son los elementos que no pueden ser explicados usando los conceptos del lenguaje. Todo lenguaje tiene tales elementos: el punto en la geometría plana, en número real en Fortran. Las primitivas son definidas en base a intuición, experiencia, o algún sistema formal.

Generación Telescópica. Aquí una pieza de Software S1 es utilizada en la creación de una segunda pieza S2, la que a su vez se usa en la creación de una tercera y así sucesivamente. Algunas veces cada nivel es diseñado como un superconjunto del nivel anterior.

Todo proceso de generación de Software portable se basa en un lenguaje ya existente o involucra el diseño de lenguajes intermedios así como la máquina abstracta asociada a éstos.

Descripción General de un Traductor

Un traductor es un programa que es usado para transformar un programa, el "programa fuente", de un lenguaje fuente en su equivalente "programa objeto" en el lenguaje objetivo. Frecuentemente, aunque no siempre, el lenguaje objetivo será lenguaje de máquina, sin embargo, es importante pensar en el lenguaje objetivo como en algún otro nivel linguistico.

Esta implantación puede verse como una secuencia de dos procesos como lo muestra la figura 2

Programa --> Traducción --> Programa --> Ejecución --> Resultados
Fuente objeto

figura 2

Generalmente estos dos procesos están separados, pero esta división puede variar considerablemente. La fase de traducción puede sencillamente transformar el programa fuente en alguna forma interna la cual puede corresponder a la arquitectura de una máquina abstracta. Posteriormente la ejecución, corresponderá a la interpretación realizada por otra parte del traductor.

El proceso de traducción puede dividirse en dos subprocesos: análisis y generación.

Durante la fase de análisis del programa se analiza la estructura de las oraciones que lo integran, y se integra una representación abstracta. La representación abstracta del programa puede tomar diversas formas dependiendo de la estructura del traductor.

En su forma original el programa fuente, se representa por una cuerda de caracteres de una sola dimensión, aunque un programa realmente tiene dos dimensiones. A fin de representar esto el programa fuente contiene símbolos sintácticos cuya única función es indicar la estructura bidimensional del programa, por ejemplo la utilización de un punto y coma (;) a fin de separar los enunciados y los símbolos o palabras llave nos indican la estructura fundamental del programa. En su forma abstracta, el programa no contiene éstos símbolos

sintácticos debido a que está representado en una estructura bidimensional y ya no se requieren.

Durante la fase de generación del compilador, el programa abstracto se convierte en un programa objeto equivalente. Esta fase comunmente se conoce como generación de código. Aunque se ha presentado la generación de código como una fase independiente del análisis, esto no es siempre el caso. Si el análisis y la generación de código se combinan, el traductor frecuentemente es llamado compilador de un solo paso. La mayoría de los traductores para lenguajes grandes requieren de más de un paso. La decisión de usar un compilador de un solo paso depende de las partes del lenguaje que se utilizan en un lenguaje particular. Estas restricciones aseguran que se tenga información suficiente para generar el código objeto, antes de que el análisis del programa fuente sea completamente terminado.

La fase de análisis puede a su vez dividirse en tres subprocesos scan, parse y terminación figura 3

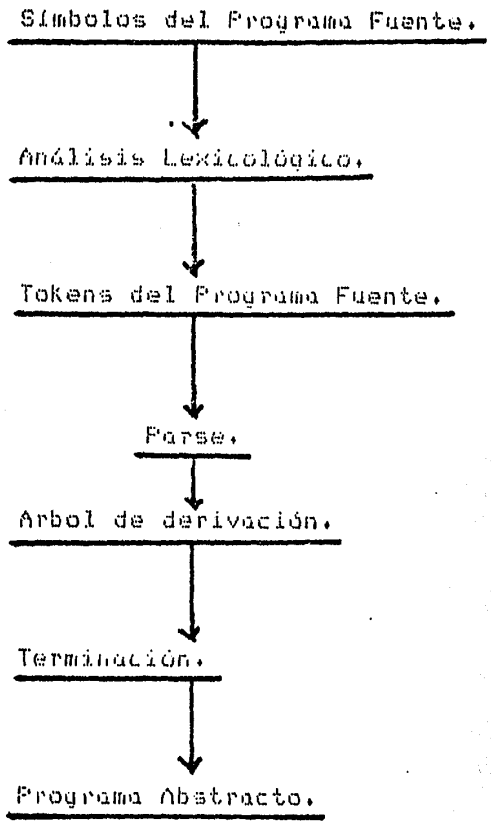

Símbolos del Programa Fuente.
Análisis Lexicológico.
Tokens del Programa Fuente.
Parse.
Arbol de derivación.
Terminación.
Programa Abstracto.

Figura 3

Durante la fase de análisis lexicológico "scan", el programa fuente es dividido en tokens, o sea en las unidades sintácticas básicas con las que el programa es construido. Cada token representa una secuencia de caracteres que pueden ser tratados como una entidad lógica única.

Se utilizan dos tipos de técnicas de parse arriba-abajo y abajo-arriba. Cada una de estas se caracteriza por el orden en el cual las producciones del árbol de derivación se reconocen. En la metodología del Pascal se emplea arriba-abajo.

El programa para analizar un lenguaje en especial se lleva a cabo partiendo de los diagramas de sintáxis ya definidos. Estos diagramas escencialmente representan el diagrama de flujo del programa, básicamente lo que se debe hacer es seguir al pie de la letra las siguientes reglas:

1.- Reduzca el sistema de gráficas al menor número posible, llevando a cabo las substituciones adecuadas.

2.- Traduzca cada gráfica en una declaración de procedimiento de acuerdo con las reglas 3 a 7.

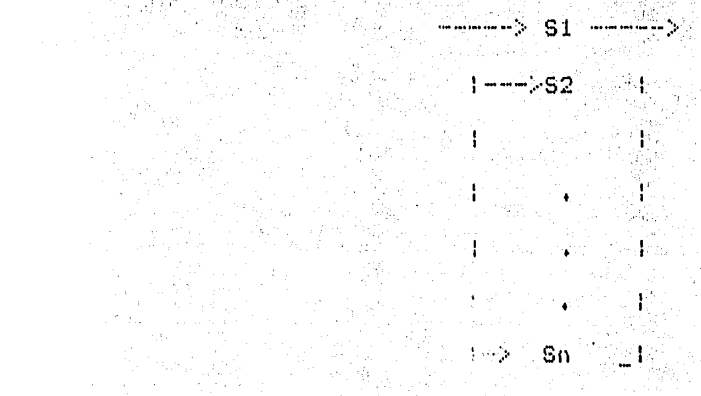
3.- Una secuencia de enunciados

S1 --> S2 --> ... --> Sn -->

se traduce en el enunciado compuesto

begin T(S1); T(S2); ... ; T(Sn) end

4.- Una elección de elementos



es traducida en el enunciado selectivo o condicional

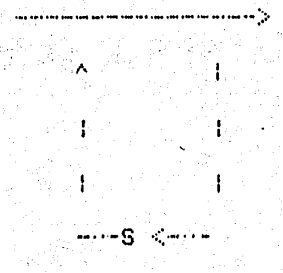
```
case ch of

   L1 : T(S1);
   L2 : T(S2);
      . . .
   Ln : T(Sn)

end
```

```
if ch in L1 then T(S1) else
if ch in L2 then T(S2) else
                . . .
if ch in Ln then T(Sn) else
error
```

donde Li se refiere al conjunto de simbolos iniciales de la construcción Si (Li = first (Si))

Notece: si Li consiste de un simbolo único a, entonces "ch in Li", debe expresarse como "ch = a".

5.- Un lazo de la forma



se traduce en el enunciado

While ch in L do T(S)

donde T(S) es la traducción de S de acuerdo a las reglas 3 a 7, y L es el conjunto L = first (S)

6.- Un elemento de la gráfica mostrando otra gráfica A

se traduce en un enunciado de invocación al procedimiento A

7.- Un elemento de la gráfica que involucre un elemento terminal

se traduce en el enunciado

```
if ch ← x then read (ch) else error.
```

donde error es una rutina llamada cuando se encuentra una construcción mal formada.

Los compiladores de varios pasos mapean el lenguaje fuente gradualmente en el lenguaje objeto. El primer paso mapea el lenguaje fuente en un primer lenguaje intermedio. El segundo paso mapea el primer lenguaje intermedio en un segundo y así sucesivamente hasta llegar al lenguaje objeto.

Existen diversas ventajas en usar formas de código intermedio. Un código intermedio puede representar un código objetivo más adecuado que ensamblador o código de máquina. Ciertas

técnicas de optimización pueden ser llevadas a cabo más fácilmente en formas intermedias que en código fuente o en código de máquina. Del mismo modo un código intermedio independiente de máquina es importante en cuanto a la portabilidad.

La mayor desventaja del uso de lenguajes intermedios es que el código producido puede ser menos eficiente que el producido directamente a nivel de lenguaje de máquina. Obviamente la razón por la que se dice esto es que los lenguajes intermedios necesitan otro nivel de traducción o sea que se ejecutan en una máquina abstracta. Sin embargo los tipos de optimización que se pueden llevar a cabo debido a la existencia del código intermedio pueden más que exceder el costo de los varios pasos de traducción.

Sistema Operacional

La implantación de un sistema, ya sea nuevo o una modificación, consiste básicamente en entrenamiento y conversión. El entrenamiento de todos los involucrados en la utilización de éste, proveyendo información recibiendo información o utilizandolo directamente es vital para el correcto aprovechamiento. El entrenamiento no solo debe estar

dirijido hacia como utilizar el sistema sino a como interpretar los problemas y que pasos tomar para resolverlos.

La conversión es el proceso de un sistema a otro, comunmente se utiliza uno de los cuatro métodos siguientes para la conversión: sistemas paralelos, corte directo, enfoque piloto y fase de entrada del proyecto. De estos el de sistemas paralelos es el que ofrece mayor seguridad, sin embargo solo es aplicable cuando se cuenta con un sistema operacional. En contraste corte directo es el que mayor riesgo entraña. Cuando el sistema involucra un gran proyecto la entrada gradual del sistema es una buena idea. Cuando se tratan nuevas ideas es importante el uso de proyectos piloto a fin de recibir realimentación de los usuarios.

Experiencia en el empleo

Después de que el sistema ha quedado instalado y la conversión terminada, se debe llevar a cabo una revisión a fin de determinar si el sistema está cubriendo las expectativas y donde se requieren afinaciones. La calidad del sistema, la confianza de los usuarios así como las estadisticas del sistema se deben evaluar. esta revisión es una valiosa fuente de información para posteriores proyectos.

APLICACION AL MODELADO EN ECONOMIA

> Ninguna confesión religiosa ha pecado tanto por el mal uso de expresiones metafísicas como las matemáticas
>
> Ludwig Wittgenstein, 1929

Enunciado del problema

El modelado en economía es un proceso complejo el cual involucra tareas creativas en al menos ocho áreas:

1. Teoría Económica. La Econometría es la ciencia que reune la teoría y los datos. Como se enseña en todo curso de estadística, la correlación no establece causalidad. Así el primer paso de todo esfuerzo de modelado da como resultado una abstracción del proceso económico a ser estudiado. Aunque conceptualmente esta abstracción es puramente teórica, en la práctica varias restricciones existen tanto en la exactitud como en la posibilidad de obtener datos, los cuales permitan de manera adecuada la estimación. En esta etapa el modelista debe establecer un compromiso entre lo que es teóricamente correcto y lo que es factible en la práctica.

2. Manipulación de Datos. Los datos crudos tienen su origen en diversas fuentes. Aún cuando estas fuentes corresponden a entidades del Gobierno Federal, las metodologías y niveles de

agregación y periodicidad difieren grandemente en la práctica. Los datos preliminares, revisados y ajustados por estacionalidad, plantean serios problemas al economista y al profesional de los datos. Una vez realizada la especificación preliminar del modelo, el economista debe relacionar las series de datos con los conceptos a incluir en el modelo. Estas series pueden no checar en la frecuencia deseada, o pueden estar ajustados por estacionalidad o pueden presentarse serias inconsistencias entre las series. La recolección de datos puede representar una tarea , así como requerir de un gran esfuerzo de procesamiento para convertirla a una forma útil para el modelista. Estos datos son clasificados como exógenos o endógenos. Una variable exógena es un dato que esta predeterminado en el sentido de que su valor debe ser especificado antes que el modelo sea resuelto y su valor no es alterado por el proceso de solución, de aquí se deduce que un gran esfuerzo dentro del pronóstico econométrica se dedica a desarrollar pronósticos para las variables exógenas. El conjunto de las variables endógenas está compuesto por aquellas variables explicadas por el modelo. Después de que se han generado supuestos acerca de lo que va a pasar con las variables exógenas, el modelo determina valores para estas variables.

Algunas variables endógenas son determinadas por identidades. Otras son determinadas por ecuaciones de comportamiento las que son construcciones teóricas las cuales tienen coeficientes de desconocidos o parámetros a los que se les asignan valores antes de resolver el modelo.

3. Estimaciones. El modelista procede a estimar en forma individual o en conjunto las ecuaciones que integran el modelo. En la práctica esta es una labor compleja que involucra la busqueda iterativa de la especificación correcta de una ecuación. Una gran variedad de técnicas de estimación puede ser necesaria. Idealmente estas deben presentarse como opciones directas para el modelista. La estimación es sumamente importante en términos de recursos de computo.

En los últimos años un gran esfuerzo se ha dedicado a tratar de contestar la pregunta acerca de los métodos adecuados para estimar sistemas de ecuaciones simultáneas, los modelos grandes son comunmente estimados empleando mínimos cuadrados ordinarios con una sola ecuación, a pesar de los bien conocidos problemas estádisticos de consistencia.

Aunque los métodos de estimación sofisticados son más atractivos teóricamente desde el punto de vista estadistico, la mayoria de los modelos grandes no han

sido estimados usando los métodos de información completa, como mínimos cuadrados de tres etapas o máxima verosimilitud. El problema consiste en que el número de relaciones de comportamiento de estos modelos es mayor que el número de observaciones con que se cuenta para estimación. La matriz de covarianza del error muestral es singular en estas condiciones. Aún independientemente de este problema, se considera que será utilizada con gran confianza la estimación de los modelos grandes por mínimos cuadrados ordinarios en una ecuación; esto básicamente por la gran sensitividad de los métodos de información total a los problemas de especificación, las no linealidades que parecen ser esenciales en muchas relaciones y la frecuencia con la que las ecuaciones son modificadas o agregadas al modelo. Las revisiones de datos y cambios en la estructura del modelo nos llevan a reestimar parte o todo el modelo con una frecuencia que haría bromoso el empleo de métodos de estimación más complicados.

La utilización de mínimos cuadrados ordinarios crea varios problemas para la validación de un modelo econométrico. Aún si se tiene un conjunto de estimaciones que se comparte adecuadamente, el problema de generar medidas específicas de bondad de ajuste y estadísticos de prueba se encuentra en general sin

solución. Dadas las propiedades de las estimaciones por mínimos cuadrados ordinarios, esto es, que puede esperarse que sean inconsistentes y sesgadas, los dos métodos típicos para validar los modelos son el examen de las propiedades de la simulación, el análisis de multiplicadores y el resultado de los pronósticos. En general no hay métodos de inferencia estadística para la validación de modelos de pronóstico, lo que hace que la decisión de si un cierto modelo es una buena herramienta de pronóstico o no, es materia de buen juicio.

4. Formación del Modelo. Dadas las fuentes básicas de datos y una lista de las variables que se desea pronosticar, se pueden construir un gran número de modelos. comunmente el modelo se plantea en un diagrama de flujo en el que se indican las principales direcciones de causalidad, quedando siempre dividido en varios bloques, como podrían ser bloque fiscal, financiero, precios, salarios, demanda final, etc. Estos bloques son formados comunmente por grupos independientes y se integran poco a poco en el modelo final. En el manejo independiente de los bloques los datos que provienen de otros bloques se consideran exógenos desde el punto de vista del primero.

5. Solución del Modelo. Este es la tarea más mecánica del proceso. El modelo debe ser simulado ya sea en el

período histórico o sobre el período de pronóstico deseado. Este es un problema básicamente de métodos numéricos. Los algoritmos de solución son bien conocidos y han sido sujetos a un estudio detallado. Aún así debido a que cada modelo tiene sus propias características numéricas y debido a los algoritmos aritméticos de longitud finita, pueden presentarse dificultades al tratar de resolver el modelo.

Típicamente para encontrar la solución se emplea alguna variante del algoritmo de Gauss-Seidel [ORTE 70] para resolver este tipo de sistemas. El algoritmo de Gauss-Seidel se ha encontrado muy eficiente aún en la solución de grandes sistemas no lineales. Aún si el sistema fuera lineal, si fuera del tamaño de los modelos que habitualmente se tienen, probablemente será más eficiente usar el algoritmo de Gauss-Seidel que llevar a cabo la inversión de una matriz de grandes dimensiones.

El consumo de recursos durante la solución del modelo depende de el orden en el que se presentan las ecuaciones, ya que los algoritmos de solución pueden verse afectados. Obviamente el ordenar a mano las ecuaciones de un modelo grande es una labor sumamente costosa. existen ya técnicas de análisis de flujo que ayudan a llevar a cabo esto directamente por la computadora. Nótece que en términos

de eficiencia en el cálculo, es necesario únicamente evaluar una pequeña parte de el sistema en iteraciones sucesivas. Todas las variables predeterminadas y exógenas solo necesitan evaluarse una vez. Se puede lograr una eficiencia aún mayor si se pasa al lado derecho el valor de la última iteración para la variable o variables clave más de una vez durante la pasada. El uso de valores para las variables rezagadas y endógenas para la primera iteración también reduce el número de iteraciones necesarias para lograr la covergencia, pero en general no alterarse la solución si el criterio de covergencia es lo suficientemente estricto.

Con el objetivo de validar el modelo la simulación tiene que ofrecer facilidades especiales. Una vez que se ha realizado la simulación de un período donde el conjunto total de ecuaciones es resuelto simultáneamente, el ejercicio de simulación puede tomar dos caminos. Podemos continuar resolviendo el modelo para el siguiente período usando los valores históricos que se tienen tanto para las variables endógenas rezagadas, o podemos usar los valores generados en la solución para las variables endógenas. A la primer opción se le llama simulación estática y a la otra simulación dinámica. Ambas involucran la solución del modelo en forma simultánea y el uso de las mismas variables exógenas, pero en el primer caso las variables

rezagadas siempre asumen sus valores históricos actuales, y en el otro caso asumen los valores calculados por el propio modelo.

Una simulación dinámica es obviamente una prueba más drástica de la calidad del modelo y es equivalente a un ejercicio de pronóstico. Esta es la prueba que un modelo debe pasar antes de ser considerado apto para fines de pronóstico. Interesan tanto las rutas de simulación tanto estática como dinámica, debido a que si el modelo no se comporta adecuadamente en la segunda, la primera nos servirá para buscar la o las relaciones que están generando el problema. A fin de poder aislar el problema se debe contar con un mecanismo para asignar un valor exógeno a la variable del lado izquierdo de las expresiones sospechosas, o sea un mecanismo para exogenizar variables.

6. Generación de Reportes. Una vez obtenida la solución del modelo, el modelista requiere presentar resultados, esto no solo involucra la generación de reportes y graficar, sino una manipulación de datos importante ya que muchos reportes y gráficas involucran comparaciones entre soluciones alternativas según diversos escenarios o entre las soluciones y la historia.

7. Mantenimiento del Modelo. Los modelos cambian constantemente en el tiempo, así como deben ser adaptados a nueva información. Nuevas ecuaciones deben ser probadas y evaluadas. El modelo debe ser reestimado periódicamente. Los cambios en las fuentes de datos pueden exigir modificaciones en el modelo etc.

8. Administración del Modelo. Con muy pocas excepciones la mayoría de los modelos del mundo real son administrados en dos formas:

   a) La especificación de valores futuros para las variables exógenas.

   b) Estableciendo factores que se ajustan para incorporar información recientemente obtenida en relación a fuerzas no modeladas. El modelista no solo necesita un medio fácil para manejar estos nuevos valores en el modelo, sino también un conjunto amplio de herramientas analíticas por medio de las cuales pronosticar sus valores.

De los puntos tratados anteriormente se puede deducir que la labor del modelista es una tarea de administración de recursos de información. Esta tarea de administración de información complica los componentes conceptuales del modelado en varias ordenes de magnitud, siendo esta

complejidad creciente en forma exponencial con el tamaño de los modelos.

Respecto a la experiencia anterior en sistemas de cómputo, la mayoría de los econometristas, manejan bien un paquete para manejo de series de tiempo desarrollado originalmente en el M.I.T. llamado TSP (Time Series Processor), este paquete consta de un lenguaje cómodo para la labor del econometrista y será usado a lo largo del proyecto para evaluar la especificación de las ecuaciones y llevar a cabo las estimaciones independientes. La mayoría tienen conocimientos básicos del Fortran aunque no experiencia en desarrollo de programas de más de 50 a 100 líneas.

Se detectó una gran inquietud por poder seguir paso a paso todo lo que suceda a lo largo de la solución del modelo.

El mantener correctamente documentado el modelo y la posibilidad de contar con herramientas que permitan analizar la causalidad en su estructura fueron también temas recurrentes en las entrevistas.

Los futuros usuarios estuvieron de acuerdo en diseñar y colaborar en las pruebas del sistema.

Requerimientos.

Como consecuencia de las entrevistas con el grupo de Econometría se detectó la necesidad de un sistema que facilite el planteamiento, especificación, simulación y administración de modelos macroeconométricos grandes.

El sistema debe cumplir con las siguientes características a fin de satisfacer las inquietudes de los econometristas:

a) El sistema permitirá definir un modelo, el que en su estado de máximo desarrollo implicará 1200 variables con aproximadamente 1000 coeficientes estimados.

b) El sistema contará con medios para definir los modelos, estructurados en bloques, ecuaciones e identidades.

c) Los modelos utilizarán básicamente información en forma de series de tiempo, las observaciones serán números reales.

d) Toda información histórica deberá existir en los Bancos de Datos, en estos se identificará por una etiqueta simbólica y ésta a su vez se utilizará en el modelo para referirse a la serie. En otras palabras la conexión entre el modelo y la información se llevará a cabo a través de las etiquetas de las variables.

e) Toda variable que corresponda a ecuación, identidad ó exógena, debe forzosamente existir en el Banco de Datos.

f) Dentro del modelo se requiere definir variables auxiliares, las cuales en su mayoría se calculan a partir de información histórica. Estas variables auxiliares son útiles para la estimación de las ecuaciones y generalmente tienen un uso específico en el modelo. Es útil tener acceso a éstas variables antes de llevar a cabo la estimación y también durante la validación del modelo, por lo cual se dán las facilidades para que todas las variables auxiliares tengan una etiqueta definida por el modelista directamente.

g) El modelo deberá estar plenamente autodocumentado.

h) La definición del modelo deberá quedar independiente de su explotación, o sea debe conservarse la especificación del modelo sin alteración durante la validación y simulación a fin de garantizar que el modelo en su estructura no haya sido modificado.

i) Se requiere un mecanismo de asignación de supuestos flexible y confiable.

j) En las expresiones con componentes de error estocástico, a fin de ayudar en la étapa de validación se requiere poder

agregar el componente de error. Esto como una opción para el econometrista durante la validación.

k) Se desea poder estimar las ecuaciones independientemente o dentro del modelo, ya sea en forma selectiva o total.

l) A fin de que sea rápida la adopción del nuevo simulador se conservará, en la medida de lo posible el conjunto mental y la estructura conceptual de los econometristas.

m) Se requieren reportes de causalidad y referencias cruzadas entre el modelo y los Bancos de Datos.

n) Se requiere independizar la definición del modelo de los detalles computacionales de la solución de éste.

o) El conjunto de métodos de estimación quedará reducido a las formas funcionales de mínimos cuadrados ordinarios con ajuste autocorrelación por cochrane-orcult y polinomios de Almon.

p) Las estimaciones de cada ecuación es deseable efectuarlas en un período particular. La restricción será que estos períodos deberán estar contemplados en el período general del modelo.

Diseño del Sistema Notacional

El problema de diseñar un sistema notacional para los modelos macroeconométricos, es sumamente sencillo debido a que como se planteó anteriormente éstos modelos pueden ser representados matemáticamente, además existe ya bastante experiencia en sistemas notacionales para estadística, por lo cual en base a esto se simplifica enormemente la tarea.

En el detalle final los modelos están integrados de un conjunto de ecuaciones algebráicas. Tomando como ejemplo particular un modelo Keynesiano simple para una economía cerrada tenemos lo siguiente:

$$Y = C + I + G$$

$$C = a_0 + a_1 Y$$

$$I = b_0 + b_1 r + b_2 Y$$

donde:

Y = ingreso

C = consumo

I = inversión

G = campos Gobierno

r = tasa de interés

$$y = Y_t - Y_{t-1} = \text{cambio en } Y$$

Aquí a los coeficientes desconocidos o parámetros (a_0, a_1, b_0, b_1, b_2) se les debe asignar un valor antes de resolver el

modelo. Y estos parámetros pueden ser estimados a través de una serie de técnicas estadísticas. Independientemente del método usado se tendrá un error residual en la ecuación de tal forma que generalmente se tendrá una ecuación de la forma:

$$C = a + a\ Y +$$

donde es una medida de la desviación de C del valor a + Y. El término es llamado disturbio estocástico y de esta forma se le llama estocástica la relación, por lo que se puede sencillamente representar las expresiones estocásticas de la forma:

$$C = Fe\ (u,Y)$$

donde:

Fe es una forma funcional ya previamente definida como mínimos cuadrados ordinarios.

u variable unitaria a la que una vez estimada le corresponderá el coeficiente a ,

en base a lo anterior se ve que el modelo puede sencillamente expresarse como

$$Y = C + I + G$$

$$C = Fe \quad (u\ ,\ Y)$$

$$I = Fe \quad (u,\ r,\quad Y)$$

Es importante hacer notar que esta forma de denotar las expresiones estocásticas no solo nos sirve para integrar la expresión para simulación, sino que también contiene la información suficiente para llevar a cabo la estimación de los parámetros, los que en este caso no están explícitos en el modelo, pero esto no ocasiona ninguna dificultad.

Especificación del lenguaje.

A fin de diseñar un lenguaje para la definición de modelos macroeconométricos en forma independiente de la solución, así como que contenga esta definición la información suficiente para la estimación y solución eficiente.

El lenguaje se pretende sea de muy alto nivel incluyendo la característica de no proceduralidad, a fin de darle más libertad al econometrista para que defina su modelo en la forma que más le convenga independientemente de la representación en la computadora.

El modelo se identifica por un nombre así como cada uno de los bloques en que se estructure. El modelista tendrá la posibilidad de incluir comentarios en la definición a fin de esclarecer esta, en lo posible. Se podrá hacer uso de sangrías en forma indiscriminada a fin de aumentar la legibilidad.

Se definen dos tipos básicos de variables: las generales y las auxiliares. Las variables generales serán las exógenas, las definidas mediante una ecuación o identidad las ecuaciones e identidades se declararán de forma explícita las exógenas serán aquellas que no son definidas en ningún punto del modelo. Las variables auxiliares son aquellas que se requieren para linealizar o agregar ciertas variables a fin de integrarlas en las ecuaciones.

Como se planteó anteriormente es necesario para la validación del modelo poder exogenizar bloques completos, identidades y ecuaciones estocásticas. A fin de poder llevar a cabo esto de manera confiable, sin crear efectos de segundo orden, se define un ámbito para las variables auxiliares o generadas, ya que el econometrista puede incurrir en errores al suponer que cierta variable está o no siendo recalculada. Con la definición de ámbito se aclara la semántica del lenguaje para los usuarios.

Sintáxis del lenguaje

Se optó por una sintáxis en la que los enunciados fueran delimitados por punto y coma (;) e iniciaran con una palabra reservada.

Se emplearon palabras reservadas ya que esto permite una detección de errores más fácil, las palabras reservadas son similares a las que emplea el T.S.P cuando las funciones existen, esto es importante debido a que evita problemas semánticos a los usuarios; por otro lado para los conceptos que no tienen una contraparte semántica en el T.S.P se buscaron términos en Inglés que tuvieran un significado semántico adecuado. Se buscó que no hubiera ningún conflicto semántico entre los términos del lenguaje y simulación y el T.S.P.

La razón por la que se utilizaron palabras provenientes de la Lengua Inglesa, fue buscando una homogeneidad en el conjunto de términos. Por otro lado el tener palabras reservadas en Inglés, ofrece un nivel de diferenciación con los nombres de variables y términos propios del modelo la cual en experiencias con otros lenguajes de computadora se ha visto de utilidad.

La generación de rezagos y adelantos motivó especial atención, debido a que la sintáxis propia del T.S.P de colocar después del nombre de la variable entre parentesis un número entero positivo para los avances y negativo para los rezagos genera muchos problemas dado que el lenguaje pierde su

verticalidad, ya que esto implica un operador mas que el valor de una función. La aplicación de lo anterior facilita el enseñarla a nuevos usuarios que deseen resolver problemas más complejos ó que tengan experiencia en lenguajes como Fortran y Pascal, los problemas ocasionados por la semántica del lenguaje se vuelven muy complejos.

Para manejar los rezagos, adelantos, primeras diferencias etc. se diseñó una sintáxis de tipo funcional en la que todas las funciones llevan un solo argumento y los nombres de las funciones son palabras reservadas también. En las funciones predefinidas se tiene cubierto ya las primitivas, mas las funciones comunmente requeridas para las actividades de modelado.

Aunque no hay una limitante teórica para anidar funciones ó pasar como argumentos a las funciones expresiones, el deseo de que todas las variables intermedias se pudieran analizar tuvo como consecuencia el evitar el anidamiento, con ventajas desde el punto de vista de la implantación del lenguaje. La desventaja obvia es la necesidad de tener gran número de variables auxiliares.

Para los comentarios se define una sintáxis similar a la del lenguaje Ada; es comentario cualquier cuerda que inicia con un signo de numeral (#) de donde se encuentre este hasta el fin de la línea, esto tiene la ventaja que se puede documentar sobre

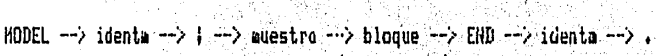
la misma línea del lado derecho. El lenguaje definido es un lenguaje LL1 tal y como lo pide la metodología del Pascal.

En la figura 4 se presenta la sintáxis completa del lenguaje en en diagramas de sintáxis.

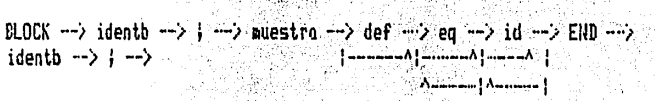
Figura 4

SINTAXIS DEL LENGUAJE

MODELO

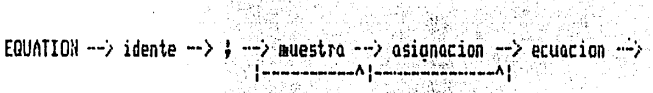


BLOQUE

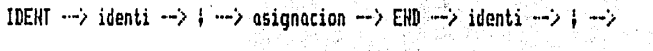


DEF

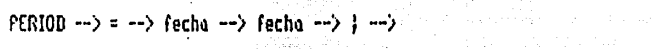
EQ
--



ID
--



MUESTRA

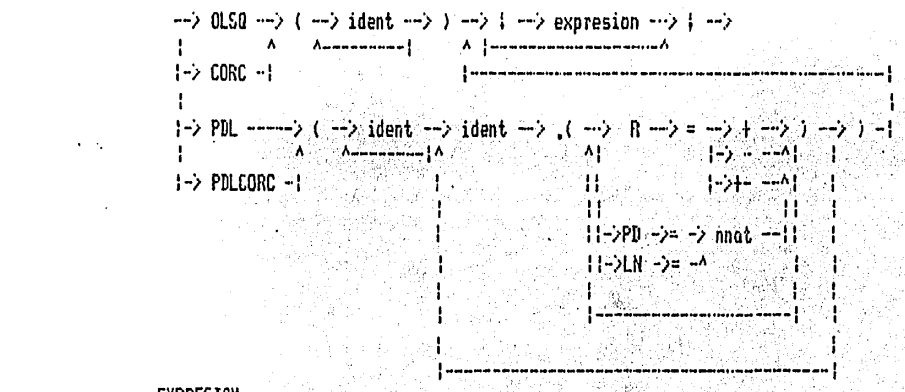


ASIGNACION

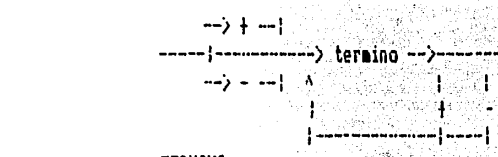
ECUACION

Figura 4 continuacion

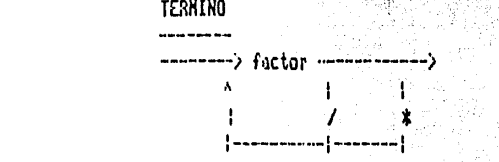
RSECUACION



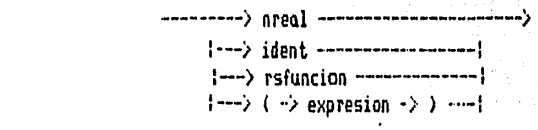
EXPRESION



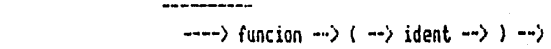
TERMINO



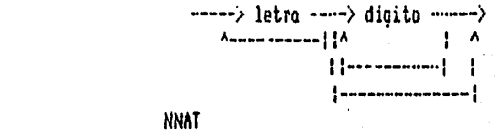
FACTOR



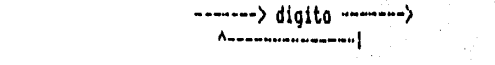
RSFUNCION



IDENT



NNAT



NREAL

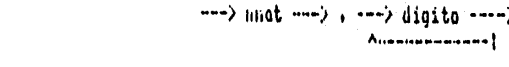


Figura 4

Semántica del Lenguaje.

El nombre del modelo no podrá utilizarse con otro fin en el modelo, lo mismo es válido para los nombres de bloque.

Un nombre de variable cuando más aparecerá una vez del lado izquierdo de un signo de asignación en todo el modelo. Esto es obvio ya que no tiene sentido tener dos ecuaciones definiendo la misma variable en el modelo. Las variables que no se encuentren definidas, automáticamente serán categorizadas como exógenas.

Todas las variables definidas explícitamente como ecuación o identidad así como las implícitamente definidas como exógenas deberán encontrarse en el Banco de Datos.

El periodo definido para el modelo será el de default para todo el modelo a excepción que se defina un nuevo periodo para el bloque. Es importante señalar que todos las definiciones de periodo deben encontrarse dentro del periodo definido para el modelo.

Las variables que se definan en el bloque global, serán consideradas como globales dentro de todo el modelo, las variables que se definan en la sección global de cada bloque se

podrán utilizar en el bloque sin restricción de ámbito. La razón de lo anterior es que el bloque global no se puede exogenizar igual que la sección global del bloque.

Para la estimación de las ecuaciones se utilizará el período muestral activo para el ámbito de la ecuación.

Se considerará como error si a una variable declarada expresamente como ecuación o identidad no se le asigna un valor dentro de la sección.

También será un error si dentro de una sección definida como ecuación no se define una expresión que involucre una expresión estocástica.

Diseño de la implantación.

Aún y cuando la estructura del lenguaje parece hacer factible emplear un traductor de un solo paso, se optó por hacer uso de varios pasos a fin de tener todo preparado para la generación de diversos subproductos deseados por los usuarios, así como para tener la posibilidad de intentar posteriormente diversas técnicas de optimización.

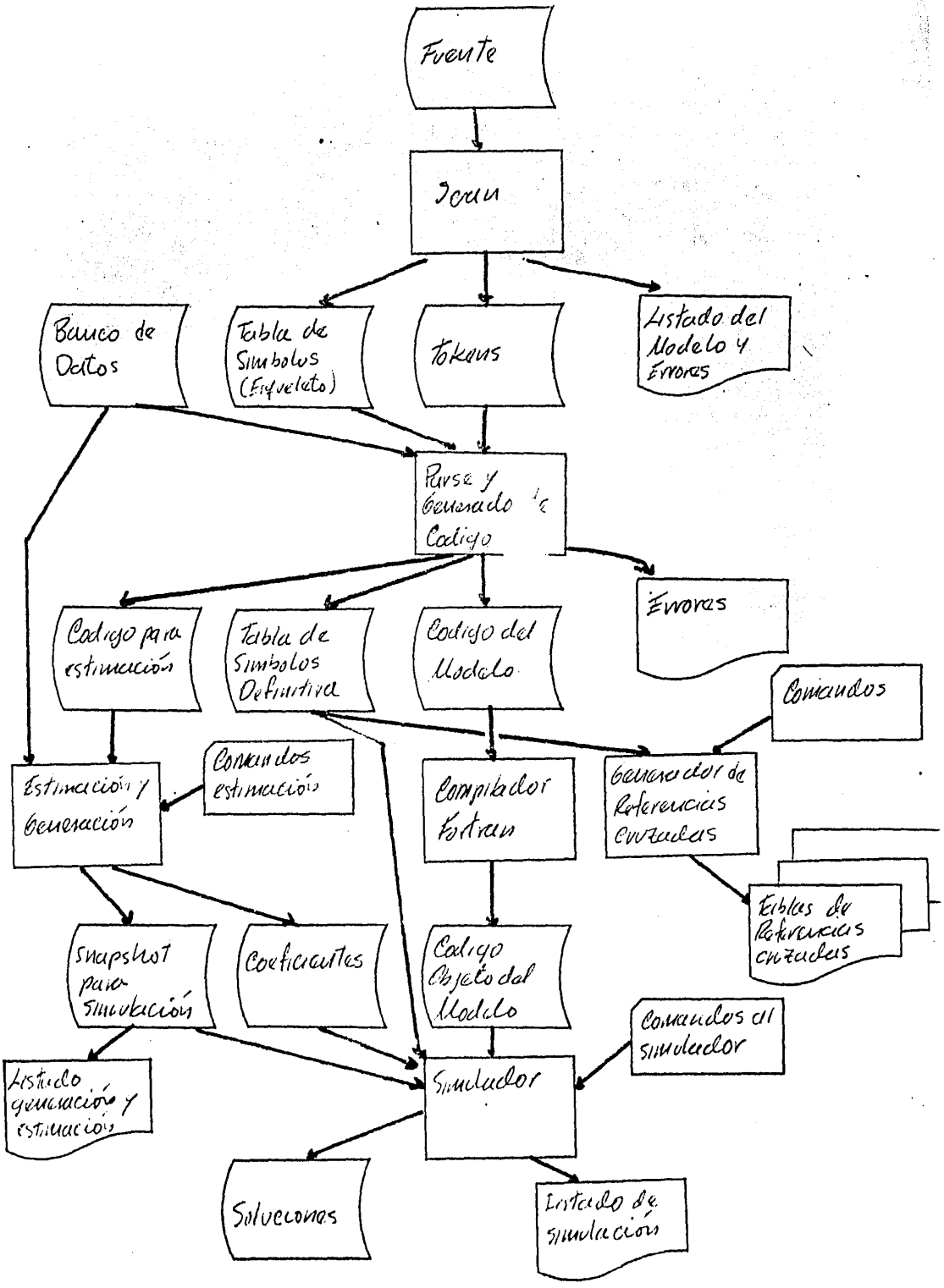
La definición del modelo implica tanto su estructura como la forma de estimar los parámetros para las ecuaciones, de aquí se tiene que se debe generar código para dos máquinas

abstractas. La primera será en la que corra la simulación y la segunda la que lleve a cabo la estimación de parámetros.

Para la simulación se define una máquina abstracta Fortran IV en el estandar 1966. La elección de este nivel linguistico implica muchas ventajas, en principio el que esta máquina sea "actual", en el sentido que se definió anteriormente es muy útil para la validación del traductor, del mismo modo que la calidad de código que genera el compilador Fortran de la máquina real.

A fin de llevar a cabo la estimación de parámetros se genera código para una máquina abstracta, la cual no es más que un paquete estadistico el cual interpreta los comandos generados durante la traducción, estos comandos se generan como cápsulas etiquetadas con el nombre de la ecuación, para permitir la estimación de ecuaciones en forma selectiva; de la misma forma se puede omitir la estimación aprovechando este código para únicamente generar las variables auxiliares que se tienen en el modelo. La generación de variables auxiliares es útil para efectos de contar con datos para la validación

Figura 5
Estructura General del Sistema



de la simulación, así como para tener ya un valor a partir del cual iniciar el proceso de solución del modelo, lo que disminuye el número de iteraciónes necesarias para lograr la convergencia.

A través de un registro de coeficientes se tendrá comunicación entre ambas máquinas.

El tipo de trabajo requerido para preparar un modelo requiere necesariamente que en el momento que se está trabajando en su concepción se congelen los datos a fin de que el modelista tenga control completo sobre el medio ambiente en el que su modelo se desenvuelve. A fin de no entorpecer la labor del grupo de administración de datos, se ofrecerá una opción de "snap-shot", con la cual en el momento de la creación del modelo se creará una versión independiente de los datos que el modelo utiliza, esto se complementará con un conjunto de utilerías independientes que le permitirán al modelista comparar sus datos congelados con los actualizados por el grupo de administración de datos.

Análisis Lexicográfico

En esta fase se lleva a cabo el análisis lexicográfico, el reconocimiento de las palabras reservadas y la integración de

la tabla de símbolos. Los productos serán los mensajes de error correspondientes al análisis lexicográfico, un listado del programa fuente al que se le agregarán números de línea a fin de referir posteriormente los errores, un archivo conteniendo los tokens así como la tabla de símbolos ya armada, ésta se almacenará posteriormente en un archivo con el fin de utilizarla en un proceso auxiliar en la generación de tablas de referencias cruzadas, junto con el archivo intermedio.

Parse y Generación de Código

La identificación de la estructura sintáctica así como la semántica del lenguaje y la generación del código para ambas máquinas abstractas se llevará a cabo en ese paso. Los productos serán mensajes de error, dos archivos de código objeto, así como la tabla de símbolos completa con información respecto al entorno de las variables, así como su categoría.

Funciones Auxiliares

Dentro de éstas funciones auxiliares tenemos la generación de los procedimientos en el lenguaje de control a fin de preparar todo el medio ambiente, generar archivos, etc. para integrar lo

necesario para llevar a cabo la simulación, todos estos serán personalizados en base al nombre del modelo.

Simultáneamente se generarán un conjunto de lanzadores a fin de que los usuarios puedan interactuar directamente con el sistema, llevando a cabo generación de reportes, corridas de simulación, mantenimiento de archivos etc.

Implantación

La implantación se llevó a cabo en lenguaje FORTRAN IV utilizando archivos secuenciales y directos según convino, para el manejo de la tabla de símbolos se tienen arreglos en memoria y para pasar la tabla de una fase a otra, así como para almacenarla en forma definitiva se empleó un archivo secuencial.

La fase de Scan o reconocedor lexicográfico prepara la tabla de símbolos en la etapa inicial, cargando las palabras reservadas del lenguaje y asignando un número entero el cual corresponde al token específico, la estructura permite que en caso de requerirse nuevas palabras reservadas sea sencillo llevar a cabo ampliaciones tanto en el lenguaje como en los procedimientos interconstruidos. Así mismo dentro del Scan se tiene una tabla de caracteres lexicográficos como son signos de puntuación, paréntesis, etc., a los cuales se les ha asignado un número entero negativo, la razón para asignarles un número entero negativo fue básicamente el de diferenciar entre los símbolos

contenidos en la tabla y estos caracteres. El reconocedor lexicográfico toma el archivo en el que se encuentra el programa fuente, le asigna a cada línea de éste un número y genera un listado, al generar el listado se marcan en él los errores de sintáxis que quedan en el ámbito del reconocedor lexicográfico, básicamente errores en la formación de los identificadores y constantes así como la presencia de caracteres fuera del conjunto del lenguaje, cuerdas demasiado largas etc.

Los productos del generador lexicográfico son por tanto un listado con número de línea y el texto del programa fuente, un archivo secuencial en donde cada registro consiste de n+1 números enteros, donde el primer número corresponde al número de línea y los demás a los n tokens identificados en la línea (el número de línea es importante para identificar los errores identificados en el parse con el código fuente original), así como un archivo secuencial conteniendo el esqueleto de la tabla de símbolos.

El segundo paso del traductor implica la etapa de parse y la generación de código objeto, tanto el código para la generación de variables intermedias y estimación, como lo que propiamente es el código objeto del modelo. En la etapa de inicialización del programa se prepara la tabla de símbolos cargando el esqueleto generado durante el paso de reconocimiento

lexicográfico, y se busca en el Banco de Datos la existencia de los identificadores asignándose dentro de la tabla de símbolos el número de variable correspondiente en el Banco a cada Serie de Tiempo, una vez terminado esto a los identificadores que no se encuentran en el Banco se les asigna un número entero negativo cuyo valor absoluto no corresponde a una Serie de Tiempo utilizada en el modelo, esta posición será utilizada en el Banco de Datos denominado "shapshot para simulación" para colocar las variables generadas requeridas para la estimación y como valores iniciales para acelerar el proceso de simulación y validación.

Una vez interada la tabla de símbolos se procede a la etapa de parse, si se realiza un análisis cuidadoso de la sintáxis más el requerimento del usuario de manejar expicitamente las variables intermedias, se observa que se puede usar un algoritmo de parse recursivo descendente el cual solo requerirá ser recursivo en lo que se refiere al manejo de expresiones. Por un lado es importante notar que aunque se puede manejar este tipo de parse en FORTRAN IV esto no es fácil por la imposibilidad de manejar llamadas recursivas en el lenguaje, por otro lado al ser el lenguaje objeto FORTRAN el cual deberá compilarse después del proceso de traducción, se tiene en base a esto que no es necesario analizar las expresiones ya que con sólo checar que la expresión este bien formada y convertirla a código FORTRAN se tendrá resuelto el problema. Así la estructura del parse es

básicamente la de los diagramas de sintáxis a excepción de las expresiones las cuales se simplifican notablemente. El parse debe tener en cuenta la detección de las situaciones que ocasionan posibles llamadas recursivas a fin de marcarlas como error y evitar un error de ejecución del traductor.

En la fase final de la segunda pasada se revisa la tabla de símbolos a fin de constatar que todas las variables que implícitamente quedan como exógenas al no aparecer en el lado izquierdo de una asignación, existan en el Banco de Datos. Así mismo que a todas las variables declaradas como identidades y ecuaciones aparezcan en el lado izquierdo de una asignación, con esto se da por terminada la traducción del modelo, se salva la tabla de símbolos completa en un archivo secuencial, mismo que se utilizará como entrada para los programas generadores de referencias cruzadas y reportes de variables empleadas en el modelo.

Las pruebas del traductor se diseñaron tanto teniendo en cuenta la estructura del lenguaje como su significado, se compararon los resultados de los procedimientos estadísticos involucrados con resultados de paquetes confiables, se probaron por separado las dos fases tanto en base a pruebas de detección de errores como el efecto que tenía la presencia de un error en la detección de los subsecuentes.

Para la prueba del parse se utilizo tanto archivos de tokens producto del reconocedor lexicográfico, como archivos diseñados especialmente para las pruebas, este tipo de pruebas corroboró la ventaja de utilizar un traductor de varios pasos.

Sistema Operacional.

Para establecer el sistema ya en forma operacional se capacitó a los usuarios por medio de sesiones de instrucción en pizarrón, durante los cuales se analizó la implantación de un modelo teórico sencillo, Ejemplo 1, en base al cual se mostraron las bondades del sistema y el cómo explotarlo, se capacitó a los usuarios a través de este ejemplo en la utilización del producto a lo largo del ciclo de vida del modelo y se les llevó por inducción a ejercitar el sistema a fin de obtener su confianza.

La conversión del sistema no implicó mayor problema ya que las interfases de usuario, las estructuras de Banco de Datos y el generador de reportes se conservan intactos, la relación del nuevo sistema con lo ya existente no requirió capacitación ya que el sistema genera sus propios lanzadores.

Experiencia en el Empleo

El lenguaje se ha utilizado para la integración, puesta a punto y explotación de cinco modelos macroeconométricos. Este lenguaje ha mostrado ser cómodo para los usuarios. La integración de tres de los cinco modelos fue realizada por modelistas que no participaron en las reuniones de especificación y consulta para el diseño de la herramienta.

El mecanismo de entrenamiento básicamente ha consistido en sesiones tutoriales con los modelistas. El problema básico que se enfrenta es que la mayoría considera exagerados los mecanismos de protección, así como las facilidades para exogenizar variables y bloques.

Durante el desarrollo de los modelos se encuentra resistencia por parte de los usuarios a aprovechar las facilidades de documentación que ofrece el sistema, es interesante ver que el problema es similar al que se presenta en los programadores de cualquier lenguaje de computadora, los cuales piensan que primero el programa debe funcionar y ya después lo documentarán. A fin de ayudar a mantener los modelos dentro de un estilo propuesto se piensa implantar un editor orientado a la sintáxis que a la vez que ayude a preparar los modelos ofresca opciones de formateo automático.

En cuanto a la estructura del lenguaje en sí, la sugerencia comun recogida entre los usuarios se refiere a lo bromoso de manejar las estructuras EQUATION, IDENT cuando la ecuación o

identidad se pueda expresar como una sola expresion, a fin de resolver esto se plantean dos opciones la primera, extender la sintáxis en forma tal de permitir la definición de varias ecuaciones e identidades en un solo bloque.

```
EQUATION * ;                    IDENT * ;

      E1 = . . . ;                    I1 = . . . ;
           .                               .
           .                               .
           .                               .
      En = . . . ;                    In = . . . ;

END * ;                         END * ;
```

Esta propuesta plantea el inconveniente que dificulta la creación de código autodocumentado. La seguna opción será conservar la sintáxis original y resolviendo el problema apoyado en el editor orientado a sintáxis planteado antes.

Se tiene pensado integrar todo el sistema como un medio ambiente para modelado macroeconométrico, sin embargo esto queda supeditado a situaciones ajenas al proyecto.

Para los usuarios que vienen de otros sistemas se ha encontrado que tienen dificultades al enfrentarse a un sistema no procedural. Debido a que mentalmente están preparados a utilizar como opciones de modelado, una serie de alteraciones en el algoritmo de solución o alteraciones dinámicas en la estructura del sistema de ecuaciones. Para estos usuarios el sistema en principio parece poco flexible; esta dificultad se ha sorteado mediante sesiones de asesoría en la que se les muestra la ventaja de plantear en forma explícita dentro de la filosofía del sistema, la estructura del modelo.

```
1.-   MODEL KMENTA ;
2.-   ##################################################################
3.-   #                                                                #
4.-   #          SECRETARIA DE PROGRAMACION Y PRESUPUESTO              #
5.-   #       DIRECCION GENE[illegible] DE POLITICA ECONOMICA Y SOCIAL  #
6.-   #                    [illegible]CCION DE COMPUTO                 #
7.-   #                         KMENTA/SMITH                           #
8.-   #                                                                #
9.-   ##################################################################
10.-  #
11.-  #       MODELO TRIMESTRAL
12.-  #       PERIODO 1954 A 1963
13.-  #
14.-  #
15.-  PERIOD 19541,19634 ;
16.-   BLOCK GLOBAL ;
17.-    DEFINE ;
18.-     C = 1 ;
19.-     S1 = LAG(S) ;
20.-     S2 = LAG(S1) ;
21.-     S3 = S1 - S2 ;
22.-    END DEFINE ;
23.-   END GLOBAL ;
24.-   BLOCK 1 ;
25.-    #
26.-    ##########################################
27.-    #                                        #
28.-    #      ECUACIONES DE COMPORTAMIENTO      #
29.-    #                                        #
30.-    ##########################################
31.-    #
32.-   EQUATION CT ;
33.-    # GASTO POR CONSUMO
34.-    # DONDE ;
35.-    #      Y = PRODUCTO NACIONAL BRUTO
36.-    #      L2 = DEPOSITOS DE DINERO LIQUIDO (PRIMERAS DIFERENCIAS)
37.-    #      C1 = CONSUMO REZAGADO UN PERIODO
38.-    PERIOD 19541,19634 ;
39.-     L1 = LAG(L) ;
40.-     L2 = L - L1 ;
41.-     C1 = LAG(CT) ;
42.-    #
43.-     CT = OLSQ(C,Y,L2,C1) ;
44.-    #
45.-   END CT ;
46.-    #
47.-   EQUATION IDT ;
48.-    # GASTO SOBRE PLANTAS Y EQUIPO
49.-    # DONDE ;
50.-    #      RT = TASA DE RENDIMIENTO PORCENTAGE ANUAL
51.-    #      S3= VENTA FINAL DE BIENES Y SERVICIOS
52.-    #          (PRIMERAS DIFERENCIAS REZAGADAS UN PERIODO)
53.-    #      IDT1 = GASTO SOBRE PLANTAS Y EQUIPO
54.-    #            (REZAGADA UN PERIODO)
55.-    #      TT = TIEMPO EN TRIMESTRES
56.-    #            (PRIMER TRIMESTRE DE 1954=0)
```

Ejemplo 1

122 b

```
57.-                #
58.-                PERIOD 19541,19634 ;
59.-                  IDT1 = LAG(IDT) ;
60.-                #
61.-                  IDT = OLSQ(C,RT,S3,TT,IDT1) ;
62.-                #
63.-              END IDT ;
64.-                #
65.-              EQUATION IRT ;
66.-                # CONSTRUCCION DE RESIDENCIAS
67.-                # DONDE :
68.-                #      RT = TASA DE RENDIMIENTO
69.-                #      S3 = VENTA FINAL DE BIENES Y SERVICIOS
70.-                #           (PRIMERAS DIFERENCIAS REZAGADAS 1 PERIODO)
71.-                #      TT = TIEMPO EN TRIMESTRES
72.-                #      IRT1 = CONSTRUCCION DE RESIDENCIAS
73.-                #             (REZAGADAS UN PERIODO)
74.-                #
75.-                PERIOD 19541,19634 ;
76.-                  IRT1 = LAG(IRT) ;
77.-                #
78.-                  IRT = OLSQ(C,RT,S3,TT,IRT1) ;
79.-                #
80.-              END IRT ;
81.-                #
82.-              EQUATION IIT ;
83.-                # INCREMENTO EN INVENTARIOS
84.-                # DONDE :
85.-                #      RT = TASA DE RENDIMIENTO
86.-                #      S3 = VENTA FINAL DE BIENES Y SERVICIOS
87.-                #           (PRIMERAS DIFERENCIAS REZAGADAS UN PERIODO)
88.-                #      TT = TIEMPO EN TRIMESTRES
89.-                #      IIT1 = INCREMENTO EN INVENTARIOS
90.-                #             (REZAGADA UN PERIODO)
91.-                #
92.-                PERIOD 19541,19634 ;
93.-                  IIT1 = LAG(IIT) ;
94.-                #
95.-                  IIT = OLSQ(C,RT,S3,TT,IIT1) ;
96.-                #
97.-              END IIT ;
98.-                #
99.-              EQUATION RT ;
100.-               # TASA DE RENDIMIENTO
101.-               # DONDE :
102.-               #      Y = PRODUCTO NACIONAL BRUTO
103.-               #      M = SUMINISTRO DE DINERO
104.-               #      M1 = SUMINISTRO DE DINERO
105.-               #           (REZAGADA UN PERIODO)
106.-               #
107.-               PERIOD 19541,19634 ;
108.-                 M1 = LAG(M) ;
109.-               #
110.-                 RT = OLSQ(C,Y,M,M1) ;
111.-               #
112.-             END RT ;
113.-               #
```

Ejemplo 1 (continuación)

122 e

```
114.-                ##########################################
115.-                #                                        #
116.-                #      ECUACIONES DE IDENTIDAD           #
117.-                #                                        #
118.-                ##########################################
119.-                #
120.-              IDENT Y ;
121.-                # PRODUCTO NACIONAL BRUTO
122.-                # DONDE :
123.-                #     G = GASTO DE GOBIERNO EN BIENES Y SERVICIOS
124.-                #
125.-                  Y = CT + IDT + IRT + IIT + G ;
126.-                #
127.-              END Y ;
128.-                #
129.-              IDENT S ;
130.-                # VENTA FINAL DE BIENES Y SERVICIOS
131.-                #
132.-                  S = Y - IIT ;
133.-                #
134.-              END S ;
135.-                #
136.-              IDENT L ;
137.-                # DEPOSITO DE DINERO LIQUIDO
138.-                # DONDE :
139.-                #     M = SUMINISTRO DE DINERO
140.-                #     TD = DEPOSITOS DE TIEMPO EN BANCOS COMERCIALES
141.-                #
142.-                  L = M + TD ;
143.-                #
144.-              END L ;
145.-                #
146.-            END I ;
147.-              #
148.-          END KMENTA.
******** COEFICIENTES EMPLEADOS PARA MODELO =23

        VARIABLES QUE SALIERON DEL MODELO EN ESTA VERSION


         NO.DE VAR. QUE SALIERON DEL MODELO=0

        VARIABLES QUE TIENEN ERROR EN EL MODELO


         NO.DE VAR. CON ERROR EN EL MODELO=0
 FIN DE COMPILACION  OK
```

Ejemplo (continuación)

TABLA DE SIMBOLOS

REN	NVM	ETIQUETA	NSBD	NBD	NEM	TIPO DE VARIABLE	REZAGO
027	4	C	14	0		GENERADA	0
029	6	S1	100	0		GENERADA	1
030	7	S	10	1	S	IDENTIDAD	1
031	8	S2	200	0		GENERADA	0
032	9	S3	301	0		GENERADA	0
033	10	CT	320	1	CT	COMPORTAMIENTO	1
034	11	L1	358	1	CT	GENERADA	0
035	12	L	154	1	L	IDENTIDAD	1
036	13	L2	400	1	CT	GENERADA	0
037	14	C1	444	1	CT	GENERADA	0
038	15	Y	2	1	Y	IDENTIDAD	0
039	16	IDT	631	1	IDT	COMPORTAMIENTO	1
040	17	IDT1	510	1	IDT	GENERADA	0
041	18	RT	482	1	RT	COMPORTAMIENTO	0
042	19	TT	569	1	IIT	EXOGENA	0
043	20	IRT	650	1	IRT	COMPORTAMIENTO	1
044	21	IRT1	589	1	IRT	GENERADA	0
045	22	IIT	699	1	IIT	COMPORTAMIENTO	1
046	23	IIT1	560	1	IIT	GENERADA	0
047	24	M1	701	1	RT	GENERADA	0
048	25	M	237	1	L	EXOGENA	1
049	26	G	800	1	Y	EXOGENA	0
050	27	TD	758	1	L	EXOGENA	0

Ejemplo 1 (continuación)

122e

NVARM	ETIQUETA								
4	C	DEFINIDA	18						
		REFERENCIAS	43	61	78	95	110		
10	CT	DEFINIDA	43						
		REFERENCIAS	32	41	45	125			
14	C1	DEFINIDA	41						
		REFERENCIAS	43						
26	G	NO DEFINIDA							
		REFERENCIAS	125						
16	IDT	DEFINIDA	61						
		REFERENCIAS	47	59	63	125			
17	IDT1	DEFINIDA	59						
		REFERENCIAS	61						
22	IIT	DEFINIDA	95						
		REFERENCIAS	82	93	97	125	132		
23	IIT1	DEFINIDA	93						
		REFERENCIAS	95						
20	IRT	DEFINIDA	78						
		REFERENCIAS	65	76	80	125			
21	IRT1	DEFINIDA	76						
		REFERENCIAS	78						
1	KMENTA	NO DEFINIDA							
		REFERENCIAS	1	147					
12	L	DEFINIDA	142						
		REFERENCIAS	39	40	136	144			
11	L1	DEFINIDA	39						
		REFERENCIAS	40						
13	L2	DEFINIDA	40						
		REFERENCIAS	43						
25	M	NO DEFINIDA							
		REFERENCIAS	108	110	142				
24	M1	DEFINIDA	108						
		REFERENCIAS	110						
18	RT	DEFINIDA	110						
		REFERENCIAS	61	78	95	99	112		
7	S	DEFINIDA	132						
		REFERENCIAS	19	129	134				
6	S1	DEFINIDA	19						
		REFERENCIAS	20	21					
8	S2	DEFINIDA	20						
		REFERENCIAS	21						
9	S3	DEFINIDA	21						
		REFERENCIAS	61	78	95				
27	TD	NO DEFINIDA							
		REFERENCIAS	142						
19	TT	NO DEFINIDA							
		REFERENCIAS	61	78	95				
15	Y	DEFINIDA	125						
		REFERENCIAS	43	110	120	127	132		

Ejemplo 1 (continuación)

CONCLUSION

Cuando los seres humanos encuentran feo una flor o un animal, tienen siempre la impresión de que es un producto artificial. "Parece un ..." dicen. Esto arroja cierta luz sobre el significado de las palabras "feo y "bello"

Ludwig Wittgenstein, 1951

La metodología presentada y su aplicación en la implantación de un lenguaje para especificación de modelos macroeconométricos mostró su utilidad al permitir llevar el proyecto a su fin en forma ordenada y controlable.

En aplicaciones tradicionales de procesamiento de datos, la necesidad de ofrecer al usuario final herramientas de fácil manejo enfocadas a su problema se ha detectado de tiempo atrás, dando como resultado la aparición de los llamados lenguajes de cuarta generación, la mayoría de estos productos provienen de casas de Software en busca de un nuevo mercado. Este mecanismo disparador en el proceso tecnológico muy conocido en las ciencias de la computación, tanto por su ventaja en términos de la facilidad para buscar la inovación, como por los problemas que genera; entre estos básicamente podemos citar la diversidad de productos en el mercado, la falta de estandarización y la alta probabilidad de que una vez que el usuario ha invertido una fuerte suma en esfuerzos en conversión y capacitación, se enfrente a la frustación de que el producto no era lo que esperaba.

En áreas de aplicación menos comunes o más cerca del estado del arte, es imposible esperar se genere una demanda

abierta lo suficientemente grande para atraer a terceras personas a ofrecer las herramientas adecuadas. Por otro lado el avance de la rama de la ciencia que genera la demanda puede verse fuertemente afectado por la escacez de apoyos adecuados. Es por esto que la necesidad de contar con un grupo especializado en generar herramientas es sumamente importante para las áreas que requieren un apoyo dinámico de la computadora. La labor de estos grupos podría denominarse el enseñar a la computadora el lenguaje de los expertos en un área de la ciencia. Para apoyar a estos grupos se requiere una metodología como la aquí presentada.

El punto más obscuro es la selección del paradigma o conjunto de paradigmas para un lenguaje en particular. Esto gira alrededor de preguntas como ?Qué tan natural debe parecer el lenguaje?; esta pregunta implica que ya se tiene la respuesta a ?Quién es el usuario?; y así mismo que ya conocemos el conjunto mental de los usuarios, o la disciplina en particular. La detección del grado de control de la computadora que se requiere, los recursos con los que se cuenta para desarrollar el proyecto, la calidad de la interfase hombre-máquina etc., son cruciales para la identificación del paradigma más adecuado.

Existen actualmente gran cantidad de herramientas las cuales pueden apoyar el desarrollo de procesadores de lenguaje, la mayoría son sumamente costosas y requieren de personal

altamente calificado para su empleo, algunos de estos productos se pueden conseguir más económicamente como parte de las utilerías para desarrollo dentro de sistemas operativos como el UNIX. Así mismo las herramientas para especificación de lenguajes son importantes a fin de establecer ubn mecanismo de comunicación formal. La razón por la que dentro de la metodología no se expuso el empleo explícito de estas herramientas, fue que la metodología se pretende este a la mano de cualquier programador de aplicaciones con conocimientos generales de compiladores, sin requerir que se convierta en un experto en procesadores de lenguaje.

Para mucha gente involucrada en la utilización de las computadoras, la constante aparición de lenguajes de programación plantea un tremendo problema, tanto por los efectos económicos que esto representa en cuanto a el esfuerzo de capacitación en la utilización del lenguaje, como a la incompatibilidad entre los equipos y sistemas desarrollados. Considero este comentario valido, sin embargo si se busca una conclusión a lo tratado en el capítulo de Paradigmas de programación vemos que el problema no emana de que en forma egoista un diseñador invente un lenguaje, sino del hecho de que la evolución normal de las ideas respecto a la utilización de la computadora, nos está llevando rápidamente a la invención de nuevos paradigmas y obviamente estos nuevos paradigmas requieren de lenguajes con características

especiales, los cuales irán madurando poco a poco y serán las fuerzas del mercado quienes decidirán quien quede.

Aquí es interesante observar que tal vez el ya veterano Fortran verá desaparecer en el olvido al por muchos proclamado Pascal.

El contar con una Metodología que ayude en el diseño de Lenguajes de Alto Nivel, no solo nos ayuda a llevar una cierta ruta en el desarrollo del producto final, sino a tener puntos de control para evaluación y posible realimentación a lo largo de la vida del proyecto.

En el futuro no muy lejano la presencia de la Inteligencia Artificial en productos comerciales, activará la necesidad de diseñar lenguajes nuevos que tal vez utilicen nuevos paradigmas y estos deberán contemplar de manera muy atenta la relación hombre máquina, es de esperarse que los puntos tratados por la presente sigan muy activos en la literatura en el futuro cercano.

BIBLIOGRAFIA

[AHO 79] Aho, Alfred V., Principles of Compiler Design, Reading, Massachusetts, U.S.A., Addison-Wesley Publishing Company, 1979.

[ARDE 80] Arden, Bruce W., What Can be Automated?, Cambridge, Massachusetts, U.S.A., Harvard University Press, 1980.

[ALEX 64] Alexander, C., Notes on the Synthesis of Form, Cambridge, Massachusetts, U.S.A., Harvard University Press, 1964.

[BOYD 78] Boyd, D.L. and Pizzarello, A., "An Introduction to the Well Made Design Methodology", IEEE Transactions on Software Engineering, TSE-4, 4 July 1978, 276-282.

[CALI 79] Calingaert Peter, Assemblers, Compilers, And Program Translation, Potomac, Maryland, U.S.A., Computer Science Press, Inc., 1979.

[CELL 82] Cellier, Francois E., Progress in Modelling and Simulation, London, England, 1982.

[CUFF 80] Cuff, Rodney N. "On Casual Users", International Journal Machine Studies, 1980, 12, 163-187.

[DASG 84] Dasgupta, Subrata, The Design and Description of Computer Architectures, New York, U.S.A., Jonh Wiley and Sons, Inc., 1984.

[DIJK 76] Dijkstra, Edsger , A Discipline of Programming, Englewood Cliffs, New Jersey, U.S.A., Prentice-Hall, Inc., 1976.

[FREE 80] Freeman Herbert and Lewis II, Philip M., Software Engineering, New York, U.S.A., Academic Press, 1980.

[GHEZ 82] Ghezzi Carlo and Jazayeri Mehdi, Programming Language Concepts, New York, U.S.A., John Wiley & Sons, Inc. 1982.

[HARD 82] Hardy, I. Trotter, Jr., "The Syntax of Interactive Command Language: A Framework for Design", Software Practice and Experience, Vol. 12, 1982, 67-75.

[HART 77] Hartmann Alfred C., Lecture Notes in Computer Science, New York, U.S.A., Springer-Verlag, 1977.

[INTR 78] Intriligator, M. (1978), Econometric Models, Techniques and Applications, Prentice-Hall, Inc., Englewood Cliffs, U.S.A.

[KLEI 80] Klein, L. & Young, R. (1980), An Introduction to Econometric Forecasting and Forecasting Models, Lexington Books, D.C. Heath & Company, Massachusetts.

[KMEN 71] Kmenta, J. (1971), Elements of Econometrics, MacMillan Publishing Company, Inc., Nueva York, U.S.A.

[KMEN 80] Kmenta, J. & Ramsey, J. (1980), Large-Scale Macroeconometric Models, North-Holland Publishing Company, Amsterdam.

[KUHN 62] Kuhn Thomas, S., The Structure of Scientific Revolutions, Chicago, Illinois, U.S.A., University of Chicago Press, 1962.

[KUHN 77] Kuhn Thomas, S., The Essential Tension, Chicago, Illinois, U.S.A., The University of Chicago Press, 1977.

[LEND 81] Lengard Henry and Marcotty Michael, The Programming Landscape, U.S.A., Science Research Associates, Inc., 1981.

[MADD 77] Maddala, G. (1977), Econometrics, McGraw-Hill Book Company, U.S.A.

[MALH 80] Malhotra Ashok, Thomas John C., Carroll John M & Miller Lance A., "Cognocitive Processes in Design", International Journal Man-Machine Studies, 1980, 12, 119-140.

[ORTE 70] Ortega, J.M. & Reinbolt W.C., Interative Solution of Nonlinear Equations in Several Variables, New York, U.S.A., Academic Press, 1970.

[PAGA 81] Pagan, Frank G., Formal Specification of Programming Languages. A Panoramic Primer, Englewood Cliffs, New Jersey, U.S.A., Prentice-Hall, Inc., 1981.

[PIND 81] Pindyck, R. & Rubinfeld, D. (1981), Econometric Models and Economic Forecasts, McGraw-Hill Book Company, segunda edición.

[RZEV 82] Rzevski G., "Systematic Design of Simulation Software", in Cellier, Francois E., Progress in Modelling and Simulation, London, England, 1982.

[SHAW 80] Shaw M. & Wulf W.A., "Toward Relaxing Assumptions in Languages and Their Implementation", U.S.A., Sigplan Notices, 15 (3), March 1980, 45-61.

[SPIE 83] Spiegler Israel, "Modelling Man-Machine Interface in a Data Base Enviroment", International Journal Man-Machine Studies, 1983, 18, 55-70.

[STAN 82] Standridge, C.R. & Pritsker, A.A.B., "Using Database Capabilities in Simulation", in Cellier, Francois E., Progress in Modeling and Simulation, London, England, 1982.

[TENN 81] Tennent, R.D., Principles of Programming Languages, Englewood, Cliffs, New Jersey, U.S.A., Prentice-Hall International, Inc., 1981.

[URBA 84] Urban Joseph E., "Computer Languages", in Vick, Charles R & Ramamoorthy, C.V., Handbook of Software Engineering Van Nostand Reinhold Co., New York, U.S.A., 1984.

[VICK 84] Vick, Charles R. & Ramamoorthy, C.V., Handbook of Software Engineering, Van Nostrand Reinhold Co., New York, U.S.A., 1984.

[WALK 82] Walker, Michael G., Managing Software Reliability, The Paradigmatic Approach, North Holland, Amsterdam, 1982.

[WIRT 76] Wirth Niklaus, Algorithms + Data Structures = Programs, Englewood Cliffs, New Jersey, U.S.A., Prentice-Hall, Inc., 1976.

[WRIG 82] Wright P. & Bason G., "Detour Routes to Usability: a Comparison of Alternative Approaches to Multipurpose Software Design", International Journal Man-Machine Studies, 1982, 18, 391-400.

[WULF 80] Wulf, W.A., "Trends in the Design, an Implementation of Programming Languages", IEEE Computer, January, 1980, 13 (1), 14-25.

[WULF 81] Wulf, W.A., Shaw M., Flon L., and Hilfinger P.N., Fundamental Structures of Computer Science, Reading Massachusetts, U.S.A., Addison Wesley Publishing Company, 1981.