00321

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

22

FACULTAD DE CIENCIAS

SOBRE EL PROBLEMA Y2K

T E S I S

QUE PARA OBTENER EL TITULO DE

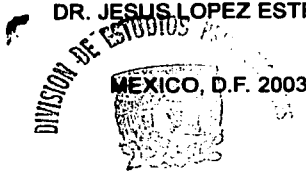
ACTUARIO

PRESENTA:

JESUS FLORES VICARIO

DIRECTOR DE TESIS:

DR. JESUS LOPEZ ESTRADA



MEXICO, D.F. 2003

PAGINACIÓN DISCONTINUA

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Autorizo a la Dirección General de Bibliotecas de la UNAM a difundir en formato electrónico e impreso el contenido de mi trabajo recepcional.
NOMBRE: Jesús Flores Vicario
FECHA: 24 de Octubre del 2003
FIRMA:

DRA. MARÍA DE LOURDES ESTEVA PERALTA
Jefa de la División de Estudios Profesionales de la Facultad de Ciencias
Presente

Comunicamos a usted que hemos revisado el trabajo escrito:

"Sobre el Problema Y2K"

realizado por **Jesús Flores Vicario**

con número de cuenta **08422540-5** , quien cubrió los créditos de la carrera de: **Actuaría**

Dicho trabajo cuenta con nuestro voto aprobatorio.

Atentamente

Director de Tesis Propietario	Dr. Jesús López Estrada
Propietario	Dra. Amparo López Gaona
Propietario	Mat. Margarita Elvira Chávez Cano
Suplente	M. en C. Wilfrido Martínez Torres
Suplente	M. en C. José Antonio Flores Díaz

Consejo Departamental de Matemáticas

M. en C. José Antonio Flores Díaz

CONSEJO DEPARTAMENTAL DE MATEMÁTICAS

Agradecimientos

Gracias al Padre Celestial y dador de vida por permitirme realizar este trabajo que en Él me inspire y regocijé.

Gracias padres mios Alfredo y Yolanda, por su apoyo y gran amor, por ser amigos y compañeros mios en las buenas y en las malas de toda mi vida.

Gracias hermanos: Margarita, María Isabel, Alfredo, Mónica Yolanda, Norma Angélica, Ericka e Isela, que me quieren tanto y apoyan en todo.

Gracias a mis sobrinos que son la esperanza y futuro del mundo.

Gracias Facultad de Ciencia por haber hecho de mi lo que soy hoy.

Gracias a todos mis maestros y amigos que siempre me ayudaron y confiaron en mi.

Gracias a mi director y asesor de tesis Dr. Jesús López Estrada por su gran paciencia y empeño, ya que sin él, no hubiera realizado este trabajo.

Gracias a mis sinodales M.en C. Wilfrido Martínez Torres, Mat. Margarita Elvira Chávez Cano, Dra. Amparo López Gaona y M. en C. José Antonio Flores Díaz, por sus comentarios y sugerencias que ayudaron a mejorar este trabajo.

Prólogo

En esta tesis profesional se aborda el famoso problema Y2K que puso al mundo de cabeza hacia finales del siglo XX. Se habla de su historia y algunas de sus soluciones propuestas; pero antes, para poder tener un panorama general de las cosas se habla de la historia de las computadoras y de como nació este dilema.

Actualmente las computadoras se utilizan ampliamente en casi todas las áreas de la actividad humana: negocios, industria, ciencia, educación, cine, televisión, diversiones, etc. Se han desarrollado a pasos agigantados respondiendo cada vez mejor a las necesidades humanas para realizar trabajos y cálculos más rápidos y precisos.

Es sorprendente cómo Arquímides, sin contar con la notación posicional indoarábiga, pudiera con instrumentos primitivos de la aritmética, lograr una muy buena aproximación del famoso número π [B05].

Una de las primeras herramientas mecánicas de cálculo fué el ábaco, el cuál se compone de un marco atravesado por alambres y en cada uno se deslizan una serie de argollas. Tiempo después aparece la notación posicional para la escritura de los números reales facilitando, la ejecución de las operaciones aritmeticas básicas y haciendo obsoletos "los abacos". Más después, Nepper agiliza las multiplicaciones convirtiéndolas en sumas. Proceso hoy conocido como logaritmos, los cuales facilitan enormemente las evaluaciones de exponenciación y el cálculo de raices de orden alto.

En 1642, Blaise Pascal, desarrolló una calculadora de ruedas engranadas giratorias, antecedente primitivo de la calculadora de escritorio modernas, que, sólo podia *sumar y restar*, se le llamó la "Calculadora Pascal".

En 1671 Gottfried Leibnitz, construyó la calculadora sucesora a la de Pascal, la cual podía efectuar las cuatro operaciones aritméticas Charles Babbage, matemático e ingeniero inglés que es considerado el padre de la computadora actual, construyó en 1822 la máquina de diferencias, la cuál se basaba en el principio de una rueda giratoria que era operada por medio de una simple manivela. Después ésta máquina fué sustituida por otra que podía ser *"programada"* para evaluar un amplio intervalo de funciones diferentes la cual, se conoció como "Máquina Analítica de Charles Babbage".

Años después, aparece Herman Hollerith, quien, en 1880, inventó las máquinas perforadoras de tarjetas, inspiradas en el telar de Jacquard. La finalidad de la máquina de Hollerith, era acumular y clasificar la información. Con esta máquina se realizó *el primer censo*, guardando la información

en una máquina, ya que antes se procesaban los censos en forma manual. Hollerith fué el iniciador de la gran compañía **IBM**.

En 1884, Dr. Eugene Felt, construyó la primera máquina práctica que incluía teclas e impresora, llamada "Comptómetro o Calculadora con Impresora". Konrad Zuse, hizo su calculadora electromecánica Z1, que ya emplea un sistema binario y un programa indicado en cinta perforada, ésta fué una máquina de tipo mecánico considerada como la primera computadora construida, debido a que manejaba el concepto de programa e incluía unidad aritmética y otra de memoria.

Howard Aiken junto con IBM construyó (1937), la computadora MARK 1, en donde la información se procesaba por medio de tarjetas perforadoras, con esta máquina se podían resolver problemas de ingeniería y física, así como problemas aritméticos y lógicos. Después aparecen la MARK II, MARK III Y MARK IV. Con estas calculadoras se alcanza la automatización de los procesos.

John Von Neumann, construyó la EDVAC en 1952, la cuál utilizaba el sistema binario e introducía el concepto de programa almacenado. La primera aplicación que se le dió a la máquina fué para el diseño y construcción de la bomba H.

Posteriormente se hizo la ABC, computadora construida por John Vincent Atanastoff, la cuál contenía bulbos, y que es considerada como la primera computadora electrónica. De ahí en adelante se subdivide el desarrollo de las computadoras en generaciones.

El concepto de primera generación se asocia a las computadoras de bulbos y al programa almacenado. En estas máquinas también aparecieron los dispositivos de almacenamiento secundario. Ejemplo de ellas fueron las máquinas: UNIVAC y MARK I. Como comentario adicional la UNIVAC fue la primera máquina digital producida comercialmente.

La segunda generación se basó en el transistor, con ello se redujo el tamaño a milímetros en comparación a la de los bulbos que ocupaban centímetros. En ellas aparecieron los dispositivos de memoria, los discos magnéticos fijos, las unidades de discos y los monitores. Ejemplo de ellas son la GE 210, IBM 7090, IBM 1401, NCR 304, entre otras.

La tercera generación se caracteriza por la aparición de circuitos integrados llamados chips, con el cuál se reducía notablemente el tamaño de todas las máquinas. En esta generación también aparece el *software* [1] portátil. La

[1] Los términos en ingles se ponen en itálicas a lo largo del presente trabajo.

computadora de esta generación fué la IBM 360.

La cuarta generación mejora a la anterior, teniendo como característica trascendental al microprocesador, el cuál permite la introducción de más transistores en un solo chip, el reconocimiento de voz y formas gráficas. La utilización de *software* para aplicaciones específicas, entre otras inovaciones. Dentro de esta generación de procesadores se encuentran el 8080, 8086, 8088, 80286, 80386, 486 y el Pentium.

En la quinta generación se emplearán microcircuitos con inteligencia, en donde las computadoras tendrán la capacidad de aprender, asociar, deducir y tomar decisiones para la resolución de un problema. Es llamada "La generación de inteligencia artificial"

Dentro de todo este esquema histórico nace desde sus fundamentos un problema que es conocido como el *"Problema del Año 2000 (Y2K)"*, de cuyo impacto aún no se sabe hasta donde llegará. La presente tesis describe su historia, su impacto, sus dificultades de resolución y algunas de sus "soluciones". Se presenta un esquema de lo que puede ser una solución completa que permita avanzar las fronteras del *software* más allá de lo actual. Dada la importancia y relevancia de las computadoras en la vida moderna la búsqueda de una solución ha este problema es interesante, polémico, urgente y apasionante.

Este trabajo nació además dentro de un contexto laboral y de investigación, motivado muchas veces por la propia experiencia. Lo consideramos por tanto como un resumen de nuestra larga labor tanto docente como de trabajo que ha hecho contribuciones al desarrollo del pais.

El trabajo ha sido diseñado de la siguiente manera: En el capítulo primero se plantean los antecedentes históricos de este problema, las soluciones que se han implementado junto con sus defectuosos sistemas de implementación, se abordará el problema del calendario y de algunas reglas mínimas que se han usado, se recalca en este capítulo que lo que se ha hecho es un parche y *no una solución definitiva.* Sin ser alarmistas se mensionarán los costos implicados por este problema, hasta ahora publicados.

El capítulo segundo presenta una variante mejorada de la solución que se denominara *"Windowing* Dinámico". Esta tampoco es una solución para n cuando n tiende a infinito, pero dá la garantía de que al menos se puede aplicar desde ahora una solución para un n mucho más grande. Se discutirá el método y se hablará de cómo implementarlo. Se sugiere también una solución de forma hipotética para cuando n tiende a infinito. Se plantea además el dilema de la representación del tiempo cuando éste se hace demasiado grande.

El capítulo tercero presenta un enfoque general de ver ya no sólo el impacto del año 2000, sino una generalización del mismo que se llamará *problema sistemológico*. Se discuten algunos ejemplos de cómo puede ser llevada a la práctica la quinta generación que por ahora se sueña, al final de este mismo capítulo se presentá una discusión de su impacto en la sociedad, retomando como ejemplo la axiomática que se ha heredado de Euclides.

El capítulo cuarto presenta un análisis comparativo de las soluciones que se han implementado y de la que aquí se propone. Se ve claramente que si se hacen bien las cosas ahora, se puede simplificar mucho lo que el futuro depara.

Se presentan las conclusiones como un resumen de la presente tesis, recalcando que todavía queda mucho trabajo por delante. Se presenta un glosario de terminología técnica y la bibliografía que se uso en esta investigación, con el propósito de hacer más entendible este trabajo y para clarificar la exposición del presente.

Jesús Flores Vicario
Cd. Universitaria Octubre del 2003

Índice General

Capítulo 1

Generalidades

1.1 Antecendentes

Se dice que cuando la computadora hizo su aparición en el mundo, fue considerada el mayor invento desde que el hombre aprendió a usar la electricidad. Hoy, cuando han pasado varias décadas, muchos se preguntan como vivía la gente sin tales máquinas. Este trabajo que se leé se preparó con ayuda de ellas. Estas pueden retener la información almacenada en su memoria y acceder a ella "instantaneamente". Las computadoras, ¡Qué maravillosas son! ¿Qué haría el mundo sin ellas?

En las regiones de vida moderna del mundo influye en casi todo aspecto cotidiano. Las pensiones de jubilación y de invalidéz, las devoluciones de impuestos, los reembolsos de las compañías de seguros y también los depósitos de nómina son transmitidos por computadora. Y lo mismo sucede con el dinero que se deposita en los bancos y los intereses que estos pagan por los saldos mantenidos en ella, aunque a veces mejor dicho son cobrados. Un sin número de aparatos modernos, como los que generan electricidad o los que purifican el agua, son controlados por computadoras. Médicos, clínicas y hospitales se valen de ellas para diagnosticar problemas de salud y salvar la vida de personas. También se usan para pronosticar el estado del tiempo y prevenir accidentes aéreos.

A pesar de la tan basta capacidad que tienen no son más inteligentes que los seres humanos que las programan. Resuelven problemas o los plantean únicamente como se les ha programado o configurado. Cuando desempeñan bien sus funciones, es al hombre a quien se atribuye el mérito. Es cierto que

pueden realizar tareas más de prisa que un hombre, pero no son capaces de ofrecer soluciones sin que se involucre a alguien y les facilite el método para hallarlo.

Por ejemplo, cuando se empezó a programar computadoras en los años sesenta, no hubo suficiente previsión. Puesto que en aquel momento la memoria era muy cara, los programadores buscaron formas de ahorrarla; por ello se ideó el método abreviado de considerar al año por sus últimos dos dígitos.

No se pensó en el abatimiento de los costos de memoria cuando se empezó a programar las primeras computadoras ¿Qué hubiera pasado si los precios no hubieran sido abatidos?. El costo de resolver este problema sería ciertamenta más alto de lo que se estima actualmente.

Esto se le ha llamado el "Virus del Milenio ó el Problema del año 2000" es una de las fuerzas de mayor potencia paralizante que conozca la industria moderna de la informática. Así para las computadoras el año 1998 es simplemente "98" ¿Cuál es el problema en esto? La consecuencia es que el 1 de enero del año 2000, el 90% del soporte físico y lógico de las computadoras "pensarán" que es el primer día del año 1900.

Ya se han cometido errores. Se reporta que "En una prisión estatal el 'virus' hizo que las computadoras calcularan mal las sentencias y varios reclusos fueron puestos en libertad", informó la revista Newsweek.

"En tiendas y restaurantes se han rechazado tarjetas de crédito porque las computadoras han confundido el "00" de la fecha de expiración". Además en varios estados en EU se han cancelado las licencias interestatales de algunos camioneros porque las computadoras han sido incapaces de evaluar las solicitudes de renovación con fechas posterior al milenio.

A las empresas de todo el mundo les costará aproximadamente **600 mil millones de dólares ($6x10^{11}$ USD)** cambiar los códigos de fechas, y esperaban hacerlo en los pocos años que restan antes de que el "Virus del Milenio", se lleve cautivas a más víctimas.

"Aunque el microprocesar nos ha traido una revolución industrial que compite con la invensión de la electricidad" escribió el periódico The Toronto Star, "también nos ha hecho más vulnerables de lo que sus inventores pudieran imaginar" Y añadió "En todo el mundo hay sistemas informáticos y microprocesadores que no pueden distinguir entre el año 1900 y el 2000. A menos que se identifiquen y se cambien los códigos, este problema podría producir un caos económico mundial".

1.1.1 Predicciones de los Expertos

"Todo el mundo, incluido yo, están vaticinando lo grave que será la situación", dijo el Senador estadounidense Robert Bennett de Utah. "Y nadie lo sabrá sino hasta el primer día del 2000 o una semana o dos después". De hecho, hay base para decir que habrá consecuencias sumamente graves para la economía y para los pueblos.

"Estamos conmocionados por los problemas de las redes de suministro de energía eléctrica, las telecomunicaciones y los servicios bancarios" dijo el portavoz de la Agencia Central de Inteligencia de Estados Unidos, CIA.

Como se ha citado, se han reportado errores debido a esto. El sector salud quizás sea uno de los que más preocupación tienen, pues puede en algunos casos afectar incluso a los aparatos biomédicos, entre los que controlan a los pacientes. Si ha esto se auna que en muchos sectores se ignoran los riesgos del año 2000 las consecuencias pueden ser verdaderamente caóticas.

A modo de ejemplo, en el mes de agosto de 1999 se publicó un anuncio en la revista *Discovery en Español* haciendo el siguiente mensaje: "La llegada del 2000 es inamovible. El que necesita moverse es usted. Queda poco tiempo para la llegada del 2000: arregle a tiempo la fecha en sus equipos de cómputo y dispositivos electrónicos y asegure así su buen funcionamiento. Sr. Empresario comuniquese al 01 800 490 4200 para recibir *sin costo* un video explicativo. Y conozca los apoyos preparados para la conversión de su empresa".

Por ello se estimó que habría la quiebra de pequeñas empresas y la retirada masiva de haberes de las instituciones bancarias por parte de los asustados clientes. En Estados Unidos, el Vicepresidente de Defensa llamó a este error informático mundial como el equivalente electrónico del El Niño y comentó: "Me atrevo a decir que no vamos a librarnos de sorpresas desagradables".

El periódico Bangkok Post dirige también la atención al problema informático de Tailandia: "Según el servicio de Información de las Naciones Unidas, las oficinas de estadísticas nacionales de esta región se encaran a un doble desafío relacionado con el milenio: atajar el problema del año 2000, denominado en la comunidad informatica como **Y2K**, en sus sistemas informáticos y prepararse para una nueva serie de censos de la población". Australia, China, Hong Kong, Inglaterra, Irlanda, Japón y Nueva Zelanda se encarán al mismo dilema.

Como se vé este es un problema mundial que clama por una solución. Algunos expertos han calculado cifras astronómicas para su costo. La oficina

de Administración y Presupuesto de Estados Unidos, por ejemplo, estima que depurar tan solo las computadoras del sector federal costará **4,700 millones de dólares ($4.7x10^9$ USD).**

Un grupo de expertos dice que una cifra más realista sería la de **30,000 millones de dólares. ($3x10^{10}$ USD)** y ¿cuánto costará hacer esto en todo el mundo? La friolenta cifra de **600,000 millones de dólares ($6x10^{11}$ USD)**, y un **billón (10^{12} USD)** [1] para afrontar los inevitables pleitos cuando algunos de los arreglos fallen. Otros grupos estiman que por litigios, reparación y negocios fállidos prodría ascender a los **cuatro billones de dólares. ($4x10^{12}$ USD).**

Como se vé el problema del año 2000 resultó ser uno de los más caros que la historia humana ha registrado. Es sin duda, uno de los proyectos más grande, de alto costo y más arriesgado que el hombre ha afrontado.

¿Qué efecto tendrá en usted toda esta situación? Dependiendo del lugar donde viva y de las medidas que estén tomando las ínstituciones, puede que no le afecte mucho, que le resulte un tanto molesto o definitivamente que le ocasionen dificultades. Especialmente durante la primera semana del año 2000. Si le preocupa porque utiliza aparatos especiales, pregunte a la institución donde lo adquirió, que problemas se le presentarán en el año 2000.

"¿Piensa qué sus problemas del año 2000 están solucionados? ¡Tal vez se lleve una sorpresa! Cada versión de programa que usa, aún de los principales proveedores debe ser revisada. Su red también está en peligro debido a que algunos chips del BIOS de su PC no cumple los requisitos técnicos, aplicaciones personalizadas y que decir de los usuarios que lanzan programas no autorizados. Cualquiera de estas causas pueden originar datos de error, decisiones comerciales poco adecuados y seria responsabilidad legal". Comentó Express 2000.

Se habló mucho en los últimos años sobre el problema del año 2000. Hay quienes lo consideran grave, otros replican que son pronósticos exagerados. Unos afirman que los servicios bancarios fallarán, aunque algunos entendidos de la banca alegan que para el 2000 casi todas sus complicaciones están arregladas. Bancomer, para citar tan solo un ejemplo, ha hecho propaganda al respecto asegurando que ellos no tendrán problemas.

Muchas instituciones ya están realizándo pruebas de simulación en laboratorios especialmente diseñados para la conversión del año 2000 y evitarse así muchos contratiempos. Más sin embargo es lamentable que no se están

[1] Billon en el termino Latino no Norte Americano.

implementando mecanismos que garanticen que no se subirán programas sin conversión del 2000. Se han subido[2], a modo de prueba, varios procesos, avisando de dicha problemática a las áreas pertinentes sin que haya tenido respuesta favorable. Por ello el mundo tendrá que esperar para ver el verdadero impacto del **Problema del Año 2000 - Y2K** [B02].

1.2 Problemas que se Afrontaron en el 2000

Ahora en el 2003, cabe la pregunta ¿Qué pasó con las vatinaciones debidas al problema Y2K? El usuario final del sector financiero afrontó pocas dificultades, pero esto debido a que las áreas de sistemas tuvieron que trabajar horario extraordinario, durante las primeras semanas de año 2000. Esto con el fin de corregir los muchos errores que se presentaron, a la hora de ejecutarse por primera vez los programas de conversión del Y2K.

En el 2002 hubo una nota periodística que llamó la atención, dos aviones en Alemania en pleno vuelo chocaron, [3] y una línea probable de investigación que hubo, fué precisamente a que hubiera sido un problema del Y2K. Sobre este aspecto no se dió mucha información, debido al pánico que ocasionan este tipo de comentarios.

1.3 Soluciones Propuestas

Las soluciones que se presentan en este trabajo, tienen que ver con el sector financiero, esto debido a que el autor estuvo involucrado en los desarrollos del Y2K tanto nacionales como en el extranjero.

Existen diferentes técnicas de conversión que se aplican para modificar el código y prepararlo para soportar el año 2000. Las técnicas que se emplean en varios Sistemas Computacionales son una combinación de las siguientes:

Expansión
Ventaneo ó *Windowing*

[2]Término técnico que hace alución al proceso de vida y desarrollo de un programa que nace en un ambiente de desarrollo, se prueba en un ambiente especial de *test* y finalmente llega a su destino final *producción*, por lo que subir se refiere a mover un programa a producción.

[3]Véase apéndices con notas periodisticas.

Puenteo ó *Bridging*

La elección de cualquiera de las técnicas antes mencionadas depende de las necesidades de cada cliente y de la situación de cada sub-aplicación, ya que como ejemplo los módulos contables son tratados de forma distinta a los módulos de créditos o de mesa de dinero. Se deben tomar en cuenta factores como volumen de información y *performs*, vigencias de información por auditorías, respaldos en bóveda y microfilmación, entre algunos detalles.

1.3.1 Técnica de Expansión

Con esta técnica, el campo **fecha** con dos dígitos en el año se expande a cuatro, añadiendo la información correspondiente al siglo. Para lo cuál los componentes de una aplicación y los campos fecha deben ser identificados.

Se debe tomar en cuenta que la mayoría de los sistemas de información pueden requerir de más capacidad de almacenamiento para manejar los campos fecha expandidos, ya que ésta es una expansión física.

Al utilizar esta técnica se requieren, además, que todos los programas, *copys*, tablas, mapas, archivos y otros elementos de trabajo, que hagan una referencia o uso de los campos expandidos, sean cambiados simultáneamente. Cabe mencionar que esta técnica es la más laboriosa de todas, ya que todos los cambios deberán efectuarse de manera simultánea.

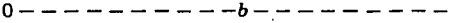
Como dato adicional, cabe decir que *el subir tan solo un sólo programa*, puede involucrar semanas de protocolo y pruebas. Pedir que suba todo el sistema al mismo tiempo, es realmente imposible. Para instalaciones grandes como los *Mainframes* puede no ser tan viable y seguro.

1.3.2 Técnica de Ventaneo ó Windowing

La filosofía de esta técnica consiste en emplear cambios de procesamiento lógico sin expandir las definiciones de los campos fecha físicamente. La asignación del siglo, se hace a partir de un **año bisagra** definido por el programador.

$$f(x) = \begin{cases} \text{Siglo } XX, & \text{si } x \geq b \\ \text{Siglo } XXI, & \text{si } x < b \end{cases} \quad \text{con } b \in [0, 99].$$

Por ejemplo: Si el año bisagra es 50, entonces todas las fechas menores al año 50, pertenecerán al siglo XXI (2000 a 2049) y las fechas mayores o iguales al año 50, pertenecerán al siglo XX (1950 a 1999).



Esta técnica aunque soluciona el problema del año 2000, dá lugar a que se deban hacer modificaciones posteriores, a otro costo adicional al que ya se está pagando. Llevando el año bisagra hasta el extremo 99, nos dá todo un siglo para buscar una alternativa mejor.

1.3.3 Técnica de Puenteo ó Bridging

Esta técnica permite hacer conversiones virtuales de datos, al hacer llamadas de los campos de fechas que permanecen con su formato original, son expandidas dinámicamente cuando son recuperadas para usarse en un programa, es decir: los archivos de datos permanecen igual, sólo los programas son convertidos.

Esta técnica incorpora *Windowing*, para establecer la información sobre el siglo. Se considera a los programas a ser modificados por la técnica de Expansión separadamente de la modificación de los archivos de datos actuales e históricos.

La ventaja de esta técnica es que permite que el código renovado interactue con el código y datos que no han sido renovados, la desventaja de esta técnica es que adolece de las mismas dificultades que se menciona para el *Windowing*.

1.4 Técnica Utilizada en Centros de Cómputo

Cabe mencionar que en muchos centros de cómputo se están utilizando variantes de estas técnicas como son el ventaneo fijo ó *Windowing*. Para ello han preparado todo un conjunto de rutinas y *copys* para facilitar su implementación. Pero volvemos a recalcar que *esto sólo pospone el problema del año 2000 al futuro, esto es un remedio y no una solución definitiva.*

1.4.1 Implementación de Políticas y Diseño

Como todo en la vida, para una mejor convivencia y armonía en los grupos de trabajo es necesario definir políticas de acción, que garanticen el buen funcionamiento de su implementación. Para realizar los cambios relacionados con el año 2000, es necesario establecer reglas de diseño y codificación de fechas, que permitan unificar los criterios utilizados para las interfaces y el código de aplicaciones, logrando con esto la estandarización para la solución del problema de comparación y ordenamiento de fechas, así como la identificación única del año bisiesto [4], a través de una función que lo determine y no de un cálculo explícito en el programa.

Con respecto al calendario existe el problema de la exactitud de la duración año, que la comunidad astronómica debe de plantear y resolver.

1.4.2 Reglas de Diseño de Interfaces

En una *interfaz* cualquier *variable* asociada con archivos, tablas, pantallas, reportes, etc., no es de uso exclusivo para ella. Estas *variables* se pasan de una

[4] La definición de año bisiesto fué introducida por el Papa Gregorio XIII en 1582 para reemplazar al calendario Juliano, que tenía una duración de 365.25 días mientras que el año verdadero es de 365.24219 días, una diferencia de tan solo 0.00781 días por año, casi ocho por milenio [B11].

Cuando el Papa Gregorio, decidió, en 1582 , cambiar las bases del calendario introduciendo el calendario gregoriano, el desfase empezó a ser serio. Gregorio resolvío acortar diez días del año 1582, de tal modo que el día 15 de octubre le siguió el 4. El resultado fueron disturbios graves en muchos lugares porque la gente pensó que se le habian quitado diez días de vida. El Papa también ordenó que los años divisibles por 400 serían bisiestos. Por ello 1800, 1900 y 2100 no son bisiestos, en tanto que 1600 y 2000 sí lo son. Definición que no todos conocen y suponen que ser bisiesto, es tan solo ser divisible entre cuatro, es decir pensamiento juliano.

De hecho, Gregorio se equivocó también en dos cosas, aunque algo menores. En primer lugar, en vez de restar diez días hacia falta restar trece para corregir bien el calendario. De hecho, este error sigue vigente y, cuando los ingleses aceptaron el calendario Gregoriano en 1752, se mantuvo este error de tres días al restar once días del año y no los catorce necesarios.

El segundo error es mucho más pequeño y Gregorio no tenía porqué darse cuenta de ello, consiste en que la duración del año Gregoriano es de 365.2425 muy próximo a su verdadero valor de 365.24219, con lo que se resta un día cada 3,226 años. Para ser más explícitos haría falta que los años múltiplos de mil no sean bisiestos si son divisibles por 4,000. En pocas palabras, el año 2000, *no debería ser bisiesto*, si es que queremos tener un calendario próximo a la realidad.

aplicación a otra, o son compartidas por diferentes aplicaciones, considerando la mecánica de manejo del *buffer*, que pretenden reducir los accesos I/O para optimizar el tiempo de *performas*.

Para controlar la variable **fecha** en la interfaz, se recomienda seguir las siguientes reglas de diseño:

> R1. *Evitar representaciones especiales de fechas*: El uso de esquemas especiales de código, tales como, 2 dígitos en el año o representaciones hexadecimales, hacen que el código sea menos mantenible. Si se va a efectuar un cambio en los campos fecha, lo más conveniente es ampliarlos a 4 dígitos en el año.
>
> R2. *Evitar cambiar variables fecha de interfaz*: Para decidir cuando cambiar y cuando no cambiar una variable fecha de interface se deben tomar en cuenta los siguientes factores:
>
> R.2.1 *Layout de pantallas y reportes:* La posición de los campos y de los *fillers* tiene que ser ajustada. El *layout* requerirá de movimientos significativos, tales cambios pueden afectar la lógica del programa, por lo que muchas veces se prefiere dejar el mismo. Incluso el formato de la variable al desplegarse en consultas y reportes. El formato deberá seguir siendo el mismo.
>
> R.2.2 *Entrada de datos:* Teclear 4 dígitos en el año, toma más tiempo al usuario que teclear 2 dígitos. La expansión requiere de una capacitación adicional para los usuarios, debido al cambio en las pantallas de captura. No es recomendable expandir los campos de entrada de datos.

1.4.3 Desarrollo de un Programa de Conversión.

Si el campo fecha es expandido, se utilizarán los *esqueletos* de programas conversores de expansión/compresión de fechas creados para tal fin. Los *esqueletos* se deben de proporcionar según las necesidades de cada aplicación.

1.4.4 Desarrollo de Programas Puente

Los problemas para que interactúe, el código anterior, el código convertido al año 2000 y la convivencia entre los datos, se solucionan con la implementación

de *programas puentes* [5].

Estos *programas puente* serán generados e integrados a las aplicaciones según su conversión.

Instalación. Cada cambio de expansión agrega costo a la conversión del archivo e incrementa el espacio de almacenamiento, así como la dificultad de convivencia entre programas y datos, por lo que se debe revisar con las áreas de producción la conversión de éstos, o convertir variables fecha a formatos estándar. Convertir las variables fecha existentes a formatos estándar, resulta ser muy complejo.

Reglas de diseño de nuevas aplicaciones. Para el desarrollo del nuevo código de programas se deberá hacer uso de las siguientes reglas de diseño:

Dígitos en el año. Deberán ser de 4 dígitos todas aquellas variables fecha que hagan referencia al año, tanto en la estructura de archivos como en la lógica de programación. Todas las declaraciones y formatos que hagan referencia al año, deberán hacerse con 4 dígitos.

La única excepción son los campos de entrada a través de pantallas. En caso de utilizar la estructura de un archivo existente, que no cuente con 4 dígitos en el año, deberá hacer uso de las rutinas estándar de fechas, para derivar el siglo.

1.4.5 Uso de Formato Estándar CCYY-MM-DD.

Todas las variables fechas, deberán usar formato estándar. La mayor parte de los sistemas soportan campos tipo fecha. En DB2 [6] la función escalar que hace referencia al año de 4 dígitos es DATE, teniendo el formato CCYY-MM-DD, donde CC es el siglo, YY el año, MM el mes y DD el día, mismo que se utilizará como formato estándar.

1.4.6 Uso de Rutinas Estándar.

Se deberán de utilizar rutinas estándar que realicen la manipulación de fechas, evitando así, hacer cálculos internos y derivaciones de siglo en cada programa. Que como ya se comentó conlleva a una definición compleja en el caso del calendario real.

[5] También se llaman programas de interfases.

[6] Veáse apéndice A para una mayor explicación del término, además apartir de aquí todas las palabras señalas con * la primera vez, aparecen ahí.

1.4.7 Palabra Clave de Reconocimiento de Variables Fecha.

Se debe de utilizar un calificativo único (FECH, DATE a cualquier identificativo propio) como palabra clave para la identificación de variables fecha, esto es con el objeto de lograr identificar el uso de variables de fecha en los programas. Por ejemplo:

WS-FECH-NAC
REG-FECH-HOY

En una mejor notación se deberían de combinar caracteres en mayúsculas y minúsculas, que permitan interpretar un procesos, con tan sólo oir su nombre, como CobraCheque, AsignaDatos, etc. En este aspecto existe y no es única, una notación muy usada y es conocida como "notación húngara", en honor Charles Simony, un legendario programador de *Microsoft.* En pocas palabras, la notación húngara empieza con una letra o letras minúsculas, que indican el tipo de dato de la variable, por ejemplo el prefijo **sz** en *szCmdLine* significa que es una "cadena de caracteres con terminación en valor cero" (en inglés, *string with zero*). El prefijo **h** en *hInstace* y *hPrevIntance* indican que son un *"handle"*. El prefijo **i** en *iCmdShow* indican que es entero."[B12]

La notación húngara, ayuda a evitar error de programación, antes de que se conviertan en fallas del proceso en sí, es menos probable que se cometan errores, si se tiene una estructura flexible de definición de tipos. Se dá una lista de las más comunes:

c	*Char*
by	*BYTE*
is	*Short*
i	*Int*
x,y	*Int* como coordenadas
cx,cy	*Int* como contador
b	*BOOL*
f	*Flag*
w	*Word*
l	*Long*
fn	*Fuction*
s	Cadena *string*
sz	Cadena con terminación en cero
h	*Handle*
p	Apuntador

1.4.8 Que no Hacer

El uso de determinaciones lógicas de siglo, es decir: ventaneo en código duro, deberá de ser cambiado por la utilización de rutinas estándares ya definidas. Es decir todo código que realice un ventaneo, lo deberá realizar por llamado a las coorespondientes rutinas. Una declaración en código duro de lógica que no debe hacerse de fechas en COBOL, sería por ejemplo:

```
01 WS-FECHA
 05 WS-SIGLO  PIC X(2) VALUE '19'.
 05 WS-ANIO   PIC X(2).
 05 WS-MES    PIC X(2).
 05 WS-DIA    PIC X(2).
```

No se debe hacer el cálculo explícito en los programas, ésto para saber si un año es bisiesto, es decir tratar de hacer el cálculo en el programa, mas bien que realizar un llamado a una rutina que sea establecida como estándar. Esto con el fin de que si en algún futuro se desea actualizar el calendario real,

sean hecho los cambios en un sólo programa y no de **n** cálculos aislados [7].

Otro error común en los sistemas bancarios es el uso de *ordenamientos internos.* Así, en caso de requerirse una modificación en el programa, que afecte a las variables de fechas, que son a dos dígitos, implica que las adecuaciones sean muy delicadas, haciendose estas muchas veces mal, afectando al sistema y generando mantenimientos costosos, con un alto riesgo de error.

El siguiente es un ejemplo de ordenamiento interno en COBOL:

```
SORT SORTFILE ASCENDING KEY REGSORT-FECHA
INPUT PROCEDURE 10000-SELECT-RECORDS
OUTPUT PROCEDURE 20000-PROCESS-SORTFILE
```

Siempre es mejor realizar los ordenamientos en los pasos previos a la ejecución del programa, reduciéndose así los altos costos de mantenimientos de los sistemas.

1.4.9 Modularización de la Conversión

Dentro de la etapa de conversión para el proyecto Año 2000, se deberá de delimitar cada sub-aplicación del sistema. El cuál puede estar constituido por muchos módulos clientes, operaciones, créditos, fondos de inversión, etc.

Esto requerirá una calendarización de las conversiones, planeación de pruebas y obtención de los Vo.Bo. de las áreas involucradas, para con ello liberar en producción un cambio *garantizado sin errores.* Este punto es muy delicado ya que hay programas que son sumamente inestables y cualquier cambio en ello resulta en *abend* de las corridas, cálculos con error y muchas horas de reproceso para corregir los cambios, hacer *rollback* a las modificaciones, aún teniendo herramientas que automatizan estos procesos.

Subir un sólo programa en producción es *sumamente costoso:* en preparación de ambiente, tiempo de pruebas, aprobaciones, validaciones, autorizaciones, métricas de calidad, protocolo de liberación. Todo ello por escritó, ó mediante un Fax, ó un e-mail [B03].

[7] En los sistemas bancarios es frecuente la copia de cálculos necesarios en tantos programas como se requiera y no mediante un acceso a una función con lo que se tiene un alto riesgo cuando este varía.

Capítulo 2

Windowing Dinámico

2.1 Una Generalización Mejor

La variante que a continuación se define se denominará *Windowing Dinámico (WD)*, y se basa en la idea de que el ventaneo no se realice en forma de código duro, si no mediante un proceso funcional.

La desventaja principal del *Windowing Estático (WE)*, es que no se ve afectado por la corriente del tiempo; se considera fija, la representación del siglo en él. Además, para cada tiempo t se requerirá de un tiempo de auditoría l para poder reconstruir los eventos de forma adecuada dentro de dicho intervalo, es decir: siempre se deberá reconstruir el intervalo $[t-l, t]$ para cualquier t y l dados. Este intervalo es el que se necesitaría de manera completa para poder representar eventos dentro de nuestro ventaneo.

Esta suposición impone la siguiente restricción sobre el año bisagra que se define en el *WD:* $l < b < 100 - l$.

Para el caso que se está analizando se supone que "el lapso de auditoria" cumple la condición anterior.

Recordando la definición de la función *WE* con el año bisagra b:

$$f_{WE}(x) = \begin{cases} \text{Siglo } XX, & \text{si } x \geq b \\ \text{Siglo } XXI, & \text{si } x < b \end{cases} \quad \text{con } b \in [0, 99].$$

Se tiene que la funcion de WD queda igual que la anterior, salvo que el parámetro b satisface la restricción $l < b < 100 - l$, con $l \in [0, 50]$.

Se denotará con $D[n]$ al conjunto de las representaciones a **n** dígitos de un entero. De esta manera $D[2]$ sería el conjunto de las representaciones a

dos dígitos.

$$D[2] = \{x_1x_2 : 0 \leq x \leq 9; i = 1, 2\}$$

Con esto ya se puede definir la función *Windoging* Dínamico Extendido (WDE):

$$f_{WDE,i} : D[2] \rightarrow D[2]$$

con el parámetro $i \in D[2]$ llamado *índice*, dada por

$$f_{WDE,i}(x) = \begin{cases} \text{Siglo i} & \text{si } x \geq b \\ \text{Siglo i+1} & \text{si } x < b \end{cases}$$

A su vez, de manera natural se tiene a la función WDE del año:

$$F_{WDE,i} : D[2] \rightarrow D[4]$$

dada por

$$F_{WDE,i}(x) \equiv f_{WDE,i}(x) \curlywedge x$$

donde el símbolo $\curlywedge$ indica concatenación de "argumentos".

Nótese que el WDE del año:
(a) Depende explícitamente del índice i, el lapso l y el año bisagra b.
(b) Se reconstruye WE para i=19; y
(c) Se extiende hasta su valor máximo del índice i=98.

De esta manera se puede construir la siguiente tabla de resultados de años ventaja con respecto al año 2000

Dígitos	Límite del índice	Años de ventaja
2	98	7,999
3	998	97,999
4	9,998	997,999

En general con n dígitos se tienen $\sum_{k=0}^{n+2} 9(10^k) - 2000$ años de ventaja. En tanto que para un *WE* solamente habrian b años de ventaja con respecto al año 2000.

Sin duda este simple ejercicio nos convence de la potencia que tiene un *WDE*.

De esta manera se seguirá trabajando a dos dígitos, pero sin perder el significado correcto del año.

2.2 Razón del WD

La razón de peso que hay de no expandir a cuatro dígitos, fué **que hay muchos sistemas involucrados, tanto propios como ajenos,** que interactuan y que toman la fecha a dos dígitos y hace las conversiones de WE necesarias. Ejemplo de ello son los sistemas bancarios que interactúan con sistemas internacionales como *American Express, Visa y Master Card* que tienen presencia en muchos paises y que necesitan de estar sincronizados.

Hacer todo este trabajo en los pocos años antes del 2000 era en la práctica imposible; por lo que ahora, los sistemas se ven obligados a vivir con estas dificultades, e interactuar con esas limitantes, que sólo les da unos cuantos años de vida.

Si lo vemos friamente, WDE es una mejor alternativa, ya que con las mismas limitantes a dos dígitos se expande en un margen mucho mejor de uso y vida de los sistemas. Pero la premura de terminarlo antes y rápido, ocasionó que aún se tenga que vivir con este problema, en el mundo presente de la informática.

2.3 Importancia de las Épocas

Si se piensa el calendario es una función monótona y creciente del tiempo, es fácil imaginar el día en el que el sólo poner el año en que se vive, no alcance ni con un libro para ponerla. Por ello es importante imponer al calendario ciclos, a los que denominamos épocas, con ello se volvería a empezar a contar en algún punto futuro.

Este planteamiento lo deben de evaluar las próximas generaciones, ya que es primordial definirlo y así tener un calendario que nos pueda alcanzar con tan sólo las implementaciones de dos dígitos, como se ha evaluado en esta versión modificada del *Windowin.* Si hacemos eso, aunque sea de forma hipotética, vemos que nuestra implementación se extiende por una larga, época en el futuro, pero el problema inicial permanece sin resolver.

Es fácil ver los calendarios de las culturas anteriores a las nuestras, como la maya que veían la necesidades de ciclos y ciclos para la definición del calendario y la medida del tiempo. La naturaleza nos enseña que las cosas son de forma cíclica. Las estaciones empiezan y acaban de esta forma. Por consiguiente parece más natural al pensar esto de forma cíclica, incluso nuestro Sol viaja a lo largo de la Vía Láctea, formando así un ciclo de forma natural.

La única razón de ser del calendario de forma lineal y monótona creciente está en la mente de un Papa que ya no esta en escena por eso ¿Por qué no pensar en los ciclos? si esto nos abre la posibilidad de seguir usando nuestro *software*, por más años de los que ahora vivimos, pero como siempre, la excusa es que no lo viviremos, que no nos correponde. ¿Pero es esa razón suficiente para implementar sistemas tan costosos y con vidas útiles demasiado pequeñas?. Si ese fuera nuestro pensar ¿Por qué entonces los matemáticos invierten tanto tiempo en pensar en procesos y funciones que se extienden en el infinito? ¿Para qué sirve entonces pensar en los comportamientos asintóticos si no es de interés y de minuciosa investigación para la mente creativa de un matemático, que pone con ello, potentes procedimientos y herramientas al alcance de la ciencia?.

En una plática con un Gerente Bancario sobre este problema, se hizo la siguiente observación: Antes para tapar un hoyo se hacia otro y ese era la forma de mantener a la gente ocupada y laborando en algo que era orden de un Rey. Pero ahora es ese mismo principio el que nos hace hacer un hoyo para tapar otro. ¿No habrá maneras más inteligentes de hacer las cosas sin que se vean afectados nuestros sistemas económicos, sociales y bancarios los cuáles viven crisis?. ¿La riqueza se seguirá generando de esta manera poco práctica y util?

Las épocas, como lo ha demostrado la naturaleza, la historia, la propia matemática y un sin fin de procesos, son la forma eficiente de definir las cosas. Es hora de sentase y ver cuantas épocas deben pasar para empezar de nuevo. Esto es, en principio, algo sencillo de hacer, al menos técnicamente; pero no así en la práctica, baste con mencionar que la sociedad moderna usa y tira, pero nunca recicla, ni el papel que necesita para imprimir sus periódicos, los ciclos y el reciclaje son cosas, que nuestra sociedad reclama, ya que los recursos del planeta son limitados y no es ni justo, ni bueno, seguir este derroche.

Pero la verdad, es que vivimos en una sociedad derrochista, que no ve el impacto de su voraz actuar sobre el planeta y sus recursos, y que por no planificar las cosas bien desde un principio se enfrenta a este tipo de dilemas.

Nuestro calendario empezó en el nacimiento de Cristo, ahora nos toca a nosotros empezar de nuevo en algún punto del futuro y así poder tener sistemas que tengan vidas útiles, que se extiendan por decenas de años en el futuro, para el bien de nuestros hijos, y de la sociedad, que algún día nos tocará dejar.

2.4 Cómo Implementarlo

En los sistemas Bancarios que cuentan con *software* la implementación más sencilla, es con un controlador de tareas como Control-M o algún otro paquete parecido, donde se pasarían como parámetros de entrada a la función *WDE* el índice (se recordará que este fué un parámetro introducido en la función WDE), en que se encuentra el proceso.

El programa que deberá hacer el ventaneo, se implementaría como una función del sistema, asegurándose que los ciclos o épocas se procesan bien, y así nuestro sistema estaría haciendo precisamente la teoría antes expuesta.

Si no se contara con el *software* antes indicado se podría generar un archivo o una tabla con la información del índice, y haríamos lo mismo. Leeríamos de él y sabríamos en que época va, aumentando los valores del índice una vez cada siglo.

Los sistemas que quedan con referencias "funcionales" son más dinámicos, ya que los procesos delicados se hacen una sola vez, y todo el sistema queda correctamente referenciado, por ello una referencia funcional del WDE es mejor que una referencia estática.

El implementar estructuras "funcionales", que permitan hacer las cosas una sola vez bien y con la garantía de que todo el sistema sabrá interpretarse bien, es mucho mejor, por lo que se recomienda hacerlo un hábito de programación. Aunque esta es una excelente solución, muchos sistemas no lo hacen así, por lo que los costos de mantenimiento son sumamente altos, debido a los malos diseños y planeaciones desde un principio.

Un ejemplo clásico de diseño "funcional", es en la programación de *Windows*, donde esa es la principal forma de diseño de programas; y es por eso que se pueden definir tamaños, color y diseño de letras tomándolas de un menú específico y logrando con ello tener un sistema referenciado y perfectamente funcional.

Como ya se mencionó los programas en COBOL para *Mainframe y UNIX* no son así, y además tienen muchas ambigüedades. Lo que se debería empezar a exigir es que los sistemas tengan esta técnica básica, y hacer mejores sistemas ahora que empezamos un nuevo milenio, para tener sistemas dinámicos como la sociedad, en la que se está comprometido a tener gran utilidad.

Capítulo 3

Sistemología

3.1 ¿Qué es la Sistemología?

Como se vió, las "soluciones" hasta ahora dadas al problema del Y2K, conllevan ha trasladar el problema a un futuro, con la *esperanza* de que se encuentre una solución real, que nos libere por completo de él. La solución que se propone a continuación es *hipotética*, como tal se harán a un lado algunos detalles reales en su implementación. Por ello se hablará de una propuesta técnica alternativa, y cuando sea el caso, se mencionarán sus limitaciones.

Para empezar se elaborará una comparación de nuestra técnica con la Astronomía y la Cosmología. En primer lugar el astrónomo estudia los distintos objetos que componen el Universo. Estos incluyen al Sol y los planetas, los diferentes tipos de estrellas, las galaxias y el material interestelar. En contraste, el cosmólogo, está menos implicado en el mobiliario cósmico específico y más con la arquitectura completa del Universo.

La Cosmología dá cuenta del Universo, como un todo, cómo comenzó a existir y cómo acabará [B13]. Por "el Universo" los cosmólogos entienden toda la palabra física completa de espacio-tiempo y materia. La Cosmología difiere de otras ciencias en que su sujeto de estudio es único, sólo hay un Universo que observar y aúnque los cosmólogos se refieren algunas veces a otros "universos", están hablando de diferentes posibilidades matemáticas, las cuáles como el universo en rotación de Gödel, que puede quizás tener alguna relación con el Universo real [B14].

De manera análoga, haremos una diferencia entre el estudio de un *sistema* y de su *sistemología*. Nuestro estudio será al sistema como un todo:

procesos *batch, on line, Off Line*, interfaces, dependencias, vinculaciones, comunicaciones, *backup*, procesos *CICS*, etc.

Cuando se hable de la solución de un problema, se verá como implicaciones de un cambio *total del sistema* y no como una parte aislada de un proceso. Por ello un "cambio" puede llegar a implicar un cambio en todos los procesos, *como un todo*. Queremos aprender como interactua un sistema en *su arquitectura básica y completa*. Con ello podremos aprender a preveer, e incluso anticipar situaciones de cambio total de un sistema dado.

Esto obliga a pedir como requisito de entrada que todo el sistema tenga solamente llamados dinámicos a rutinas y no estáticos. Esto con el fin de evitar dependencias innecesarias entre recursos, lo cuál llevaría a tener una *versión* errónea.

Por versión errónea se entenderá todo aquel cambio, que no satisface las necesidades del cliente, y/o sus servicios prestados a la comunidad, en el cuál se ha comprometido con dar atención y servicio de *calidad*.

Básicamente no deberá ser primordial el lenguaje en que se desarrolló la aplicación, es más, debe ser ajena a ella, pero para dar un panorama de cómo están las cosas, se hablará muy a menudo de COBOL, lenguaje muy utilizado en empresas grandes, que manejan *Mainframes* como base de su plataforma, donde el manejo extenso de datos es el pan de cada día.

En este lenguaje las *llamadas estáticas* a una rutina, se realizan mediante un llamado directo a la misma, lo que genera que al código ejecutable se le inserte el código involucrado, es decir, ésta es resuelta en *tiempo de compilación*, pero esto conlleva a tener réplicas innecesarias de un programa, ya que si la rutina sufre alguna modificación, ésta ya no podrá arrojar resultados adecuados, a menos que se re-compilen todos los programas que la invoquen.

En tanto las *llamadas dinámicas* conllevan a definir la rutina a través de una variable la cuál es resuelta en *tiempo de ejecución*. Esto hace al sistema más lento, pero sin réplicas.

Hacer esto, no debería ser ningún problema, ya que cada día salen mejores versiones de equipos que son más rápidos y sistemas operativos mejor diseñados.

Por ello antes de realizar cualquier cambio, se debe hacer un *análisis de impacto*, que bien puede resumirse en un reporte especial, que diagnostique como se verán afectados los módulos al implementar un cambio.

Por ejemplo: si una rutina sufre una alteración en los datos de entrada y/o salida, entonces será requerida una adecuación y compilación de todos los programas que hagan acceso a la misma.

Estos detalles en el modelo que se va a desarrollar, se resolverán en *tiempo de compilación*, a través de un mecanismo que paulatinamente iremos desarrollando.

3.2 Diccionario Sistemológico

Una parte fundamental del presente análisis será la definición y uso del *diccionario sistemológico*. Los sistemas actúan sobre una base de datos, que se tomará como dada a priori y será denominada simplemente como *datos físicos del sistema*. Los *datos virtuales* serán variables que use el sistema y que no requieren de un espacio físico para guardarse.

Denotaremos por D *al conjunto de datos* tanto virtuales como físicos. que usaremos en el *diccionario sistemológico*, es decir, se tomará como la base de datos que se use, aunada a las variables que se definan.

Definición 3.2.1 *Los posibles Atributos de Campo de un dato $d \in D$ de un "diccionario" (denotado simplemente A) son: las diferentes formas en que se puede representar (S), sus valores válidos (V), sus cálculos visibles u ocultos (C), su nivel de seguridad (E) y sus autorizaciones para poder ser modificado (M).*

De la definición anterior se sigue que el de atributo de campo es un vector $\alpha = (v, s, c, e, m)$ con $v \in V$, $s \in S$, $c \in C$, $e \in E$ y $m \in M$

Definición 3.2.2 *Por un diccionario sistemológico se entenderá una función $f : D \rightarrow A$ (i.e. de los datos a los atributos de campo). Por un espacio de diccionarios sistemológicos se entenderá un cierto conjunto de diccionarios sistemológicos $F = \{f : D \rightarrow A\}$*

Definición 3.2.3 *Se dice que los atributos son extendidos cuando se aumenta su representación a una expresión superior a la original.*

Definición 3.2.4 *Un mapeo de cambio de atributos extendido es una función $\varphi_{A,A_e} : A \rightarrow A_e$ de los atributos originales A a los atributos extendidos A_e.*

De forma análoga, se definen los conceptos de contraido, es decir, cuando este cambia a una expresión inferior a la original. También de manera análoga a la anterior, se define el espacio de mapeos de cambio de atributo extendidos.

$$\Phi = \{\varphi_{(A,A_e)} : A \rightarrow A_e\}$$

donde $A, A_e \in \Delta$. Aquí Δ denota al conjunto de todos aquellos atributos A, cuya representación tiene una longitud en bits acotada por arriba y por abajo, de antemano.

De manera natural se inducen los datos extendidos D_e, se denotará por $E : D \rightarrow D_e$ a dicha extensión, y por $\tau : D_e \rightarrow D$ a la contracción natural por truncamiento asociada a $E : D \rightarrow D_e$.

Ejemplo de ello es cuando se expande la representación del año de 38 a 1938 (función E) y de 1938 a 38 (función τ).

Así se tiene que, dada una función de cambio de atributo $\varphi_{A,A_e} \in \Phi$, se induce un operador Ψ sobre F, llamado operador de cambio de atributos, dado por

$$((f, D, A), \varphi_{A,A_e}) \xrightarrow{\Psi} (f_e, D_e, A_e)$$

de manera que

$$f_e \circ E = \varphi_{A,A_e} \circ f$$

Lo que dice que el diagrama siguiente es conmutativo.

$$\begin{array}{ccccc} & & f & & \\ & D & \rightarrow & A & \\ \tau & \uparrow\downarrow & E \searrow & \downarrow & \varphi_{A,A_e} \\ & D_e & \rightarrow & A_e & \\ & & f_e & & \end{array}$$

Esto es, a su vez, dá un criterio de verificación parcial de correctes del pre-compilador que contiene a una función de cambio de atributo φ_{A,A_e}, mediante la aplicación de rectracción (por truncamiento) $\tau : D_e \rightarrow D$, que debe reproducir los datos originales.

3.3 Pre Compilador

Dado un diccionario, estamos interesados en las implementaciones que se pueden hacer, a través de un lenguaje formal £ de las aplicaciones que se desarrollan en éste.

Definición 3.3.1 *Un programa P en un lenguaje formal £ es un algoritmo que se resuelve en sentencias del lenguaje £ y se denotara como* $P \blacktriangleright £$.

Definición 3.3.2 *Una aplicación* Λ *es un conjunto finito de programas P del lenguaje £.*

Así

$$\Lambda = \{P_i : P_i \blacktriangleright £, 0 < i < n; n \in N\}$$

Definición 3.3.3 *Una sentencia extendida de un lenguaje £ es una sentencia formal que no pertence a £.*

Definición 3.3.4 *Un compilador extendido en £, es un compilador que resuelve sentencias extendidas de £ y que a las sentencias propias de £ las deja fijas.*

Así el compilador extendido se entiende como:

$$\Pi_{\mathcal{L}}(x) = \begin{cases} \text{x} & \text{si } x \in £ \\ y \in £ & \text{si } x \notin £ \end{cases}$$

El compilador permitirá adicionalmente realizar un "sobre cargamiento" en la definición de los operadores. Así, sí tenemos una variable de tipo fecha y una de tipo numérico la operación x+1 definirá para cada caso, un resultado adecuado para cada tipo de dato, bajo el cuál fué definida la variable x, y se dice entonces que se sobre cargo el operador "+". Po ejemplo, si x es definida dentro del alfabeto y "x = g" entonces "x + 1 = h".

También se permitirán polimorfismos, que es lo mismo que un sobre cargamiento de operador, pero que este actua sobre código. Con ello, se permitirá que, código compartido se ajuste a circustancias específicas, dando reutilizacion al código fuente en tiempo de compilación.

Con ello, se pueden probar partes pequeñas del código para obtener los resultados deseados, garantizado su eficiencia para cuando éste se ha ensamblado de forma completa.

La importancia de los atributos de un diccionario, radica principalmente en que en ellos se encuentran las definiciones básicas del sistema. Los atributos de campos y las reglas básicas en las que se estructura todo la base interna del sistema. Para hacer una referencia a un tipo de campo, usaremos el lenguaje extendido en nuestro compilador de la sentencia como "like tabla.campo variable"; esta sentencia, lo único que hará a la hora de compilación, sera hacer una sustitución, ir a ver el tipo que se tiene definido en el diccionario, y con eso bastará para saber el tamaño, y atributo del campo. Así, si se usa la sentencia:

like *cust.nombre* NOMBRE

y en el diccionario se encuentra la definición de "PIC X(04)", la sustitución de esta sentencia que hará el pre-compilador será

77 NOMBRE PIC X(04).

Si luego se cambia la definición a, por ejemplo "PIC X(06)", no habría que hacer nada a los programas, ya que solamente se cambiaría en la longitud de los atributos del diccionario y con eso bastaría para hacer las modificaciones en todo el sistema y el pre-compilador harían las sustituciones adecuadamente. Para este ejemplo, por

77 NOMBRE PIC X(06).

con esto, se quiere hacer notar la ventaja de la sistemología, sobre la programación tradicional, ya que desde este punto de vista, el problema Y2K es sólo una sustitución en los atributos de campo, y la compilación de la aplicación haría tener un sistema totalmente preparado con sólo hacer una sustitución de variable. Obviamente las bases de datos sufrirían alteración y el sistema avisaría a los procesos auxiliares para que se dispararan (i.e. poner en funcionamiento) y con ello, se convirtiera toda la base de datos, ciertamente es una ventaja ver al sistema como un todo y no como un sólo programa.

También los procesos que generen archivos externos sufrirían una modificación se generaría un reporte con todos los archivos modificados para avisar a los sistemas externos del cambio implicado.

De esta manera la sistemología del Y2K, es equivalente a realizar la técnica de conversión, que se dejo de lado por su complejidad.

A continuación daremos eventos importantes dentro de la sistemología, con lo cuál los sistemas se vuelven más eficientes, completos y de fácil mantenimiento.

3.4 Kernel Sistemológico

Es usual que en las aplicaciones un mismo *proceso* [1] se repita **n** veces, debido a que ha sido copiado y adecuado para facilitar la generación de un *proceso nuevo*, muy similar al original; pero sucede, que cuando se detecta un error en los programas padres, con probabilidad casi uno, no son corregidos en los programas hijos, y al no tener un control de los programas que forman estas familias seudo isomórficas, los programas empiezen a diverger con respecto a la versión original, y esta anomalía se expande como la cardinalidad de la familia $\forall x, P \in K,\ P \rightarrow P_x \Longrightarrow Err(P) \rightarrow Err(P_x)$.

Una solución a dicha problemática es generar un *Kernel de Programas*, que serán el subconjunto original de programas, que se tomó como base para la copia, y que sirven de base central del desarrollo subsecuente, generándose así las familias seudo isomórficas.

Reconocer las estructuras seudo isomórficas, ayuda en gran manera, para poder definir las familas de kernels básicos. Si nuestro sistema se basa en familias de programas, sería interesante implementar mecanismos que permitan heredar propiedades y bondades de ciertas partes del código.

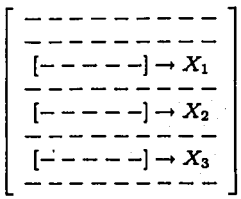
Por ejemplo, supongamos que tenemos un programa que lee un archivo en forma secuencial, el algoritmo quedaría como sigue:

```
Perform Until EOF
  Read xxxx into yyy
    at end
        set EOF to True
    Not at-end
        perform ForEach
  end-read
End-Perform
```

Si a su vez tomamos regiones consecutivas de código fuente, como por ejemplo, la declaración del *layout* de entrada (X_1), las regiones auxiliares de

[1] Subconjunto de programas de una aplicación.

memoria (X_2) y definición de la función *ForEach* (X_3), el programa podria quedar con los siguiente apuntadores de código:



Una vez dividido el programa de la forma esquemática anterior ¿Qué se puede hacer?, se puede por ejemplo, modificar los registros, cambiarlos de orden sobre el mismo layout, generar un reporte, entre otros procesos, esto con sólo cambiar las definiciones de código X_i. El ensamblanje de las diferentes regiones se haría en *tiempo de compilación generándose un programa* $\tilde{P}$ *diferente del original* P.

Con ello se quiere recalcar, la potencia de generar nuevos programas, teniendo uno como base un kernel, generándose así unas estructuras seudo isomórficas.

Estos conceptos llevados a la práctica, generan robustez en los programas, eficacia y rapidez en la codificación; y por ello, la elaboración de códigos cada vez más confiables, ya que si se detecta algún fallo en algún módulo kernel, éste se sustituiría por uno mejor, corriguiendo la anomalía, beneficiado así no solamente un programa sino a toda una familia seudo isomórfica. Esta sería una solución que como se ha dicho, es en tiempo de compilación, a la cual llamaremos *kernel estático*.

El realizar los mismos cambios pero en *tiempo de ejecución*, ciertamente sería más complejo de implementar, pero si se ha aprendido a restructurar las cosas de forma adecuada, estos mismos elementos serían implementados dentro de un programa. Una manera sería implementando dentro del programa un *seudo lenguaje*, que sea capaz de interpretarse y ejecutarse, al cuál llamaremos *Seudo Lenguaje Interno*. Como todo lenguaje formal tendría sus reglas de semántica y sintáxis, a este tipo de *kernel* más complejos los llamaremos *Kernel Dinámico*.

Además, si se identifican las partes estáticas de las dinámicas, se define de forma natural una partición dentro de la arquitectura interna de la sistemología que se esté estudiando.

3.5 Macros Referenciales

Cuando se tiene un equipo grande de personas desarrollando un sistema, algunos programadores tienen más experiencia que otros, lo ideal de un buen equipo de trabajo es que todos compartan su experiencia, y de alguna manera hacer extensivo el uso de patrones de diseño, que garanticen una mejor implementación. Estos si se basan en la experiencia entonces darán el mejor camino para el desarrollo final del sistema. Una manera de conservar y usar, sin tantos contratiempos, los buenos hábitos de programación son mediante las *Macros Referenciales*, que definen ciertas tareas de una manera precisa, concisa y eficaz.

Esta parte es medular en el desarrollo de la sistemología, ya que de ésta se desprenderá la base del código que el pre-compilador usará, para ir resolviendo el programa en el código extendido, al fuente que se pueda resolver sin problemas en el lenguaje al que se este haciendo referencia.

Una forma de identificar los programas extendidos sería por su extensión, ya que por ejemplo para COBOL los tradicionales irían como *.cbl en tanto que los extendidos irían como *.pre, éste se compilaría con un programa especial, denominado pre-compilador, el cuál dejaría el tradicional *.cbl y de ahí a la compilación normal. Con ello se puede programar en código reducido.

El comportamiento de uno u otro código así generados va a depender completamente de la implementación de la macro referencial, la cuál siempre estaría codificada para garantizar el mejor algoritmo conocido. Si posteriormente se encontrara otro algoritmo más eficiente, solo se sustituirían los módulos generadores, y compilando el sistema, éste quedaría completamente convertido garantizando siempre que se va ha hacer de la mejor manera.

Cada programador tiene su estilo propio, y eso está bien, pero a la hora de implementar algoritmos, lo recomendable es que todos hagan siempre las mismas cosas, sin olvidarse de los mismos detalles que a veces provocan errores en la ejecución.

Por ejemplo hacer un programa con rearranque [2], conlleva a tener en

[2]Programa que realiza actualización física de la base de datos y que por fallos necesita empezar desde algún punto anterior al error.

mente un cierto estilo de programar, hacerlo puede ser engorroso y hasta peligroso si no sé tienen los cuidados pertinentes. Por ejemplo, dentro de estos detalles estaría saber si leerá de un sólo archivo, o dos o más a la vez, lo cuál conlleva a guardar en las áreas de memoria unos ciertos valores, los cuáles se deben de recuperar de manera correctamente, y hacer COMMIT adecuadamente cada determinado intervalo de tiempo, etc. Todo esto quedaría reducido con la siguiente macro y todo el trabajo sería del compilador.

$$PGMREA\ FIREAD < 1 > ... < n >\ COMMMIT\ IN\ < m >$$

Un programador, quedaría capacitado sólo con saber que sentencias nuevas se encuentran dentro de la biblioteca de compilación, el resto del trabajo sería para el compilador. Si el programador trabaja de este modo, se ganaría siempre ventaja competitiva, ya que los programas serían más eficientes, y con buenos diseños, acoplándose a los dinámicos cambios que requiera el cliente y que muchas veces no está dispuesto a perder tiempo, ya que eso implica pérdida para él y para su negocio.

Mientras más rico y completo sea el arsenal de los programas así generados, darán más versatilidad y mejores resultados al menor tiempo. Otra utilidad sería cuando se cambira de arquitectura, [3] si trabajamos en *UNIX* hay ciertos estilos, presupuestos muy diferentes sí cambiamos a un *MAINFRAME*, las cadenas de ejecución cambian respetando los nombres de los ficheros y sus longitudes, etc. Trabajo engorroso y muy mecánico al convertir, programas y cadenas, pero ello se haría con una Macro.

SHECOBOL WITH (UNIX] MAINFRAME)
PROCOBOL WITH (UNIX] MAINFRAME)

Es cierto, es difícil empezar, pero si se hacen bien las cosas se puede dar a los clientes, un producto versátil, dinámico y que se adecúe con sólo una compilación a sus necesidades.

Terminamos esta parte con un ejemplo más interesante, un XSEARCH con dos algoritmos diseñados para obtener respuesta, uno con acceso directo y el otro con acceso secuencial, para mejorar los tiempos de respuesta, teniendo

[3] Plataforma en la que se desarrolla una apliación.

que leer demasiados registros de una tabla DB2. La determinación de usar uno u otro código depende del parámetro de método de acceso. Así, sí por ejemplo se uso un metodo directo y con el tiempo se ve que la respuesta es demasiado alta, con solo cambiar un parámetro y compilando el sistema se tendría una implementación mejor y efectiva sin demasiados riesgos al cambiar los códigos de los programas.

XSEARCH c1..cn FROM t (DIRECT] READ-SEQ)

Donde:
c1..cn Condiciones de la búsqueda.
t Tabla sobre la que se hace la petición.
(DIRECT] READ-SEQ) Parámetro opcional de método.

Regresando:
FOUND-TABLE Registro encontrado y su posición
NOT-FOUND-TABLE Registro no encontrado sin su posición.

Se busca de forma secuencial mediante una petición a una tabla de DB2.

```
XSEARCH   W124CODENT = W09CODENT  AND
          W124PAN    = W09PAN
     FROM TST09
   DIRECT
```

Implementado:

```
          INITIALIZE ATTARJET
          MOVE W124CODENT TO W09CODENT
          MOVE W124PAN    TO W09PAN
          MPVE 01 TO COD-OPE
          MOVE 01 TO IND-SELECCION-DB2
          MOVE ATTARJET TO ATCXXXXX
          MOVE SPACES   TO ATRXXXXX
          CALL CTE-PSTE009 USING ATCPINTM ATCXXXX  ATRXXXX
          MOVE ATCXXXXX TO ATTAJET
          EVALUATE SQL-STATUS
          WHEN ACCESO-CORRECTO-DB
                SET FOUND-TABLE  TO TRUE
          WHEN ACCESO-NO-FOUND
                SET NOT-FOUND-TABLE TO TRUE
          WHEN OTHER
                MOVE ERROR-SQL TO MENSAJE-PROGRAMA
                MOVE SQL-STATUS   TO MENSAJE-PROGRAMA (15:10)
                PERFORM 3-FINAL
                   THRU 3-FINAL-FIN
          END-EVALUATE
```

Se busca en un archivo previamente ordenado.

```
XSEARCH    W124CODENT = W09CODENT  AND
           W124PAN    = W09PAN
     FROM  TST09
 READ-SEQ
```

Implementado:

```
           SET NOT FOUND-TABLE  TO TRUE
           IF EOF-TABLE
                 SET NOT FOUND-TABLE  TO TRUE
           ELSE
                   IF W09CODENT < W124CODENT AND
                      NOT EOF-TABLE
                      PERFORM LEER-TABLA
                         THRU LEER-TABLA-FIN
                        UNTIL W09CODENT >= W124CODENT
                           OR EOF-TABLE
                   END-IF
                   IF W09CODENT = W124CODENT AND
                       IF W09PAN < W124PAN AND
                             NOT EOF-TABLE
                             PERFORM LEER-TABLA
                                THRU LEER-TABLA-FIN
                               UNTIL W09PAN >= W124PAN
                                  OR W09CODENT NOT EQUAL
           W124CODENT
                                  OR EOF-TABLE
                       END-IF
                   END-IF
                   IF W09CODENT = W124CODENT AND
                       W09PAN    = W124PAN
                          AND NOT EOF-TABLE
                      SET FOUND-TABLE TO TRUE
                   END-IF
           END-IF
```

3.6 Mapeos de Memoria y Procesos

En los programas en COBOL se definen de forma precisa las áreas de declaración de archivos, memoria y procesos. Para simplificar las cosas se supondrá que sólo existen estas áreas, que un programa debe de cumplir para poder ejecutarse.

A estas áreas se les definira un orden lógico ya que así lo necesita el programa final de conversión, para compilar:

001nnnnnnn Definición de Archivos
002nnnnnnn Memoria
003nnnnnnn Procesos

Los recursos que se necesiten, se irán concatenando en las áreas correspondientes, y cuando se genere el programa final se les dará una ordenación, para colocar en sus respectivas áreas los recursos generados, así cuando un proceso necesite la definición de un archivo de entrada, sera en el área de definición de ficheros donde se deberá generar y utilizar. También esta codificación algorítmica presupone que los archivos de entrada sean ordenados en base a las llaves, para que su localización interna sea la más eficiente posible.

Con esta partición de las áreas de los programas, las macros referenciales se vuelven dinámicas, y son posibles definir diferentes estructuras, y sus variables se generarán y serán correctamente definidas. Este punto es escencial, ya que la correcta compilación del programa, depende de que se hayan colocado adecuadamente los diferentes componentes del programa. Esta es una tarea laboriosa ya que el pre-compilador deberá ser tan inteligente que reconozca este tipo de estructuras y sea capaz de interpretarlas de forma correcta.

La práctica y elaboración de las macros referenciales en este esquema, nos enseñan que es una parte difícil de codificar, ya que una serie de líneas mal puestas, pueden hacer que el programa no compile, y que por ende, no se vea la dinámica y principal propósito de este proceso inteligente, de hacer y manipular los programas.

Una recomendación sería en este punto: se debe definir claramente la sintaxis que usarán las macros referenciales, y que nos ayudarán a definir las variables y archivos que se usarán a la hora de pre-compilar.

3.7 El Problema de la Ambigüedad

La experiencia ha demostrado, que uno de los principales problemas a los que se enfrenta un compilador es la ambigüedad. Si se definine bien la sintaxis desde un principio del seudo lenguaje formal, ella sera de gran ayuda a la hora de diferenciar los distintos algoritmos que éste puede interpretar, al cambiar sus parámetros, como el ejemplo que se expuso de *XSEARCH*, poder ubicar problemas de ambigüedad, antes de codificar es esencial para poder dar solución a ellos desde el diseño.

Por ejemplo, el precompilador pudiera tener la necesidad de compilar dos ó más veces, para reconocer estructuras ambigüas y darles la solución adecuada.

Unas funciones que ayudan mucho en el diseño de compiladores, son las de lectura de código, para poder vértir el seudo comando o macro dentro de la estructura que se puedan interpretar. Hablar de compiladores de más de una pasada es hablar de sistemas que quieren saber interpretar comandos en base al contexto.

Por ejemplo, PCTex, permite compilar más de una vez y así tener el resultado esperado quitando las referencias "cruzadas". Pero hablar de este tipo de compiladores, en un ambiente como el bancario, ciertamente debería de estar bien justificado. La justificación debería hacer ver, porque ciertos comandos requieren conocer el contexto para poder interpretarse, generando el código esperado cuando este sea sometido al proceso de compilación.

3.8 Del Lenguaje Universal al Formal

Desde el Génesis, que considera la diversidad de las lenguas como un castigo divino que impide la cooperación entre los hombres, hasta Voltaire que la califica como "una de las mayores plagas que asolan a la humanidad", muchos han lamentado la inmensa barrera que para la intercomunicación humana supone la multiplicidad de las lenguas.

Si el vulgo espeso y municipal estaba condenado a no traspasar nunca el agujero de su propia etnicidad, al menos la comunidad occidental de los sabios y eruditos tenían su propio instrumento de comunicación universal: el latín. Durante la Edad Media, el Renacimiento y el Barroco, el latín era la lengua franca de las universidades, del derecho, de la teología, la ciencia y la filosofía.

Desde Tomás de Aquino hasta Spinoza, desde Vesalio hasta Newton, casi todos los textos se escribían en latín y todas las clases se daban en latín. Todavía en el siglo XIX el matemático Gauss escribía sus obras en latín, y en latín se presentaban la mayoría de las tesis doctórales en Alemania y Francia.

Pero el latín era una lengua complicada y difícil, demasiado llena de idiosincracias e irregularidades como para permitir su uso generalizado como lengua moderna auxiliar. Por eso entre los que buscaron su resurgimiento propusieron su simplificación y regularización entre las que destacó el Latín sine flexione del lógico Peano.

Los filósofos del siglo XVII, buenos conocedores del latín, eran concientes de que esa lengua, además de ser difícil, presentaba todo tipo de defectos y ambigüedades, como cualquier lengua natural, defectos que sólo podrían ser superados con el uso de una lengua artificial.

Según Leibniz, todas las ideas complejas son combinaciones de ideas simples, lo mismo que todo los números naturales son producto de los números primos. El programa leibniziano era ambicioso: habría que analizar todas las ideas del espíritu humano, hasta reducirlo a sus presuntas expresiones simples. Era tratar a las expresiones como números naturales que puedan descomponerse en sus expresiones primas. Este trabajo nunca se concluyó y de hecho se anticipó al los de Gödel.

Pero todos estos trabajos abrieron el camino para el surgimiento de lenguajes formales del tipo artificial con reglas bien definidas de sintaxis y gramática, los cuáles pueden ser interpretadas por aparatos mecánicos. Todo ello al surgir la lógica moderna, esta asaña fué inaugurada por Frege quien es considerado como su fundador. El fué el que introdujo por primera vez los cuantificadores y las variables ligadas, con lo que pudo desarrollar la primera teoría coherente de la cuantificación múltiple. Estas trabajos permitieron analizar de un modo satisfactorio las estructuras lógicas de enunciados compuestos [B10].

Todo esto sirvió para formalizar lenguajes como COBOL, FORTRAN, C, PASCAL, por mencionar unos cuantos. La idea central de nuestro seudo compilador es tomar como base un lenguaje en particular y darle vida más amplia, con comandos extendidos que se puedan interpretar, por así decirlo, en un nivel superior de compilación. Con ello, se ve al lenguaje de forma extendida; el cuál requiere una compilación de más para poder resolver sus expresiones. Esto sin duda dá los procesos sistemológicos una ventaja sobre el original, ya que se tendrán expresiones de más alto nivel garantizando además el mejor algoritmo de solución.

El tener así a nuestro lenguaje, en un nivel superior de interpretación nos dará ventaja competitiva, ya que podremos capacitar mejor a nuestros programadores, logrando cambios significativos con un menor número de horas de trabajo. En ese sentido los lenguajes se vuelven una versión plus y propia de cada cliente, adecuada a sus necesidades particulares.

3.9 El Principal Enemigo de la Sistemología

En nuestro análisis, un factor que no puede hacerse a un lado es el *humano*, de hecho es el más limitante de todos, por ello se contrastará con un ejemplo histórico. *El sistema axiomático deductivo* conocido por el hombre, que se halla en la monumental obra de Euclides. **Los Elementos**, presenta una teoría deductiva importante e influyente, jamás concebida.

Euclides organizó el trabajo de los matemáticos que lo precedieron de una unidad bien estructurada, usando la lógica aristotélica y mediante un modelo deductivo, que por dos mil años se creyó perfecto e influenció en la forma de pensar humana, la manera de enseñar las matemáticas, entre otras cosas por todo el mundo.

Hoy se sabe que los axiomas de Euclides, ni siquiera permiten demostrar su primer teorema, pero también es cierto que la idea de rigor en una demostración a cambiado, ésta ha evolucionado influenciada por la cultura matemática del momento.

Lo que más preocupó a los matemáticos después de Euclides, fué su quinto axioma, el de las paralelas; ya que como estas verdades evidentes por sí mismas, resultaba éste axioma carente de la sencillez de los demás, ciertamente este era el 'patito feo' de una tan bella teoría; pero, ¿Qué dice tal axioma?, dentro de algunas variantes dice lo siguiente: "Si dos rectas son cortadas por una transversal de tal forma que sus ángulos internos sean menores a dos rectos, las rectas se cortarán en algún punto por el lado en que sean menor".

Generaciones enteras de matemáticos de todas las culturas trataron de demostrar ese axioma a partir de los otros básicos y por ende, convertirlo en un teorema, nunca se vieron glorificados; uno de los intentos más famosos e interesantes, fué el de Gerolamo Saccheri, quien supuso falso el quinto axioma y trató de llegar a una contradicción, no llegó a ninguna, pero los resultados que obtuvo, eran tan extraños, que concluyó que eran contrarios a la naturaleza de la línea recta y que por lo tanto Euclides quedaba disculpado de cualquier error. Saccheri nunca supo que sus resultados son una serie de

teoremas válidos de las geometrías no Euclidianas [B08].

Muchos matemáticos comprendieron que las conclusiones de Saccheri no eran una contradicción; y por ello, trataron de resolver el problema, uno de ellos fué Gauss quien escribió a un amigo: "Los teoremas de esta geometría parecen paradójicos y para los no iniciados, absurdos, pero un análisis sereno y metódico revela que no contiene nada que no sea posible". Pero ¿Por qué no publicó Gauss estos resultados? "Temía el griterio de los beocios".

Pero la solución de este problema estaba a punto de resolverse Bolyai y Lobachesky, llegaron a los mismos resultados de Gauss. Bolyai llamó a su geometría "Absoluta", en tanto Lobachesky "Imaginaria"; pero el nombre que hoy subsiste "Geometría no Euclidiana" es debida a Gauss.

Estos descubrimientos transtornaron la manera del pensamiento de los matemáticos y pasó mucho tiempo antes de ser aceptadas. Georg Cantor, habló de la ley de la conservación de la ignorancia: "Cuando se llega a una conclusión falsa y ésta es aceptada extensamente, no es fácil renunciar a ella y cuanto menos se entienda, más tenazmente se defiende"

Max Planck fundador de la mecánica cuántica dijo: "Las nuevas verdades no triunfan porque convenzan a los oponentes y hacerles ver la luz, sino porque estos mueren y las nuevas generaciones crecen con las nuevas ideas" [B06].

Por ello, retomando las palabras de Gauss la gritería de los beoicos, ciertamente hace de este planteamiento una gran limitante. ¿Que centro de cómputo es capás de afrontar estos cambios tan dinámicos, en los que, el solo hecho de hacer mal las cosas una vez, puede acarrear la desaparición completa del sistema y de los datos?

Capítulo 4

Sistemología Aplicada

4.1 Ventajas de Nuestro Modelo

Como ya se discutió, la ventaja que presenta el *Windowing* Dinámico es arrolladora, ya que nos permite desde ahora tener una ventaja sustancial de años para la conversión del problema Y2K. Lo cierto que las implementaciones costosas y laboriosas que se hicieron son deficientes. De nuevo en el futuro se oirán las voces de alerta por su mala planificación. ¡Si este costó una verdadera fortuna! ¿Cuánto nos costará el que viene?.

Ya se mencionó lo que dijo un Gerente Bancario: "Lo de hacer un hoyo para tapar otro". Este modelo de riqueza ciertamente no es ni el que mejor conviene, ni el que a la larga, dará lo mejor al planeta. ¿Cuánto se invierte en Salud, Educación y otros menesteres urgentes de nuestra sociedad?. Ciertamente los gastos son enormes, y esta sociedad no puede ignorar en que se despilfarra su dinero. Es hora de sentarse y planificar mejores sistemas que reditúen verdaderas riquezas a nuestros hijos y sobre todo que cuiden la inversión, no a unos pocos y fríos años, sino, a verdaderas y auténticas soluciones. ¿Cómo quedará justificado este gasto a generaciones futuras? No se ve ningun justificante, las cosas se podieron haber hecho mejor, desde un principio, si tan sólo pensáramos en lo que dejamos a los demas y no solo en nosotros.

Este análisis, hace ver que no se está pensando en verdaderos futuros, sino, en cercanas y costosas malas planificaciones. Algún día alguien leerá esta tesis y se dará cuenta del valor de lo que aquí se expone, y sobre todo, las ganas de dejar en claro que las cosas se pueden hacer mejor. Este trabajo lo

debe evaluar la sociedad en la que vivimos, que exija sus derechos y no se le time con sistemas que urgen arreglar, pero que por falta de previción, a la larga les costará más caro.

No se quiere plantear ésto en frías cifras, pero bien puede verse que es cierto lo que una vez se nos dijo: "Tu haz lo mejor para tí y para los demás, si no se toma en cuenta que no sea eso lo que te anime a escribir esta tesis sino que sea el deseo de ayudar a esas generaciones que vienen,...Al menos hubo alguien que pensó en los cambios de valores que muchas veces son necesarios y que pocos se atreven a hacer".

Cuando me senté ha escribir este trabajo, pensé en mis sobrinos que ahora son pequeños, pero sé que algún día crecerán y con gusto les explicaré los lugares oscuros de este trabajo, que sé, son de gran valor, a mí me ayudaron y han hecho de mi un buen programador, he trabajado en diferentes plataformas, pero en todas ellas he visto que si no tenemos disciplina, nuestro trabajo no trasciende.

Comprometámonos siempre a trabajar de la mejor manera, seremos personas felices, porque nuestro trabajo es para los demás, y no sólo para nuestro egocéntrico ser.

4.2 Abatimiento de Costos y Futuro

Cuando se preguntó por la razón de que se implementara tan mala solución al problema Y2K, se encontró siempre con lo mismo: "No nos tocará vivir", "Se cambiará por otro sistema mejor". Si los costos de la memoria no se hubieran abatido los costos de solución ciertamente serían enormes, y más al depender tanto el mundo de los servicios computacionales. Implementar un sistema nuevo es carísimo, por eso los viejos sistemas sobrevivieron por mucho más años de lo estimado.

Cuando a algunas personas se les informa que aún se trabaja en COBOL, piensan que es un lenguaje tan viejo, que ya no se utiliza. Tremendo error, pues en realidad, es el lenguaje por excelencia de los sistemas bancarios, de Internet, de gobierno, de agencias militares, en fin en todo lo que maneje volumen de información. Si hasta en las empresas de teléfonos celulares lo usan.

Como se ve los sistemas viejos, sobreviven más años de los que se les especuló en un principio. El futuro depende tanto de si aparece o no una nueva versión de COBOL. Es un lenguaje con el que se está verdaderamente

casado, y no nos podremos liberar de él tan fácilmente.

Por eso en este trabajo se ha hablado de él, es lo que se usa en el mundo real, en la sociedad en la que vivimos. Este es el paraíso de COBOL y se quiere ser realmente futurista, se puede asegurar que será el lenguaje por excelencia de los sistemas de ahora, y de muchos que están por aparecer en el mercado.

La primera vez que me topé con él fué en la Facultad de Ciencias de la UNAM, en la materia de OIPA, y que tonto fuí en esa ocasión, pensé que no me serviría para nada. Y sin embargo, ha sido el principal lenguaje en el que he desarrollado aplicaciones complejas e interesantes en Internet para servicios Bancarios.

Las computadoras van siendo cada vez mejores, más rápidas y eficientes, pero un lenguaje de *Mainframe*, que maneje volumen como es el caso de COBOL, no tiene rival, y le auguro un futuro bastante prometedor.

Hacer la sistemología del sector Finaciero es una tarea interesante, por lo que creo debería existir un mecanismo de inversión en la Universidad, Pública, Gratuita, Laica y Popular, para que esta la desarrolle como parte de un trabajo académico, así nuestra sociedad valorará la necesidad de la UNAM y de tantas otras Universidades Públicas, Gratuitas, Laicas y Populares, para trabajar en equipo y ayudar al desarrollo verdadero y sustentado del país.

4.3 Modelo Práctico

Por último quiero agregar que la sistemología es de un gran valor, ya que permite de manera fácil, ver, este tipo de problemas, como sólo un cambio de definición en el diccionario, y dejar todo el trabajo al compilador. La implementación ciertamente es interesante, de hecho se ha trabajado en algunas partes de él y se tienen algunos ejemplos prácticos que han ayudado a desarrollar programas bajo este esquema.

La sistemología del sector Financiero ciertamente es un gran reto, ya que dada su complejidad y eficiencia debería hacerse como un trabajo docente, en las que colaboren tanto la Universidad como las instituciones bancarias. Lo mismo que necesita un Banco, lo requieren las industrias que prestan servicios tecnológicos de alta calidad y servicio.

En esta época, los sistemas van siendo cada vez más complejos, y de hecho ya no hay quien los comprenda al cien por ciento; de hecho se dividen en varias áreas y muchas veces hay ignorancia en lo que a las demás implica.

Los sistemas complejos nos permiten tener vidas más cómodas, disfrutar de la vida gracias a los avances tecnológicos y si ellos sufrieran la transformación a un sistema sistemológico, problemas como el Y2K se hubieran solucionado de manera efectiva, rápida y económica. Estos cambios que a veces no se pueden implementar, hacen que los programadores hagan, uso de su ingenio y hagan implementaciones, que después se vuelven verdaderos dolores de cabeza. Así uno no se ve obligado a vivir con implementaciones defectuosas, y todo por no tener una sistemología en su sistema.

Hacer la sistemología ciertamente es un problema interesante, ya que nos permite conocer a fondo las estructuras básicas de los sistemas y poder hacer cambios y adecuaciones con mayor velocidad. El hacerlo no es ninguna pérdida de tiempo, es más bien una inversión a futuro para que las próximas generaciones vean robustez y funcionalidades en los sistemas que hacemos ahora, dejando a un lado la excusa de que "eso no lo viviremos", "ya les tocará a otros". Pretextos que ya no tienen razón de ser. Es hora de hacer algo mejor por el mundo en que vivimos, desarrollando sociedades verdaderamente justas y humanas. Que tratan a todo lo del planeta como algo que apreciar, que se valore todo hasta el papel que se tira que se invierta en donde realmente hace falta y no en arreglar unos sistemas que por falta de previsión, análisis y dirección que no se dió.

4.4 Cambios y Sistemas Óptimos

Como se ha analizado, la sistemología es ciertamente un caso general del muy recurrente error de programación como en el caso que dió lugar al problema Y2K. Cuando se diseñan sistemas, muchas veces no se cumplen las expectaciones del cliente, y hay grandes necesidades de cambiar algún dato del modelo diseñado. Estos cambios son sencillos desde nuestra óptica, mas no es así en la realidad. Para implementar dichos cambios muchas veces se tienen que pagar grandes sumas de dinero para poner los sistemas a punto y que funcionen como el cliente lo desea.

El período de tiempo que pasa entre que se compra un sistema y se deja en funcionamiento, conocido como adaptación, puede ser de meses o años, ya que no se cuenta con los mecanismos de cambio sistemológico, aumentando grandemente su costo, que finalmente pagará el usuario final con altas cuotas y reduciendo la competitividad en el mercado del cliente.

Los sistemas que se compran pagan cifras astronómicas, para adecuarlos

a las necesidades del cliente, y esto no va acorde con lo que finalmente se entrega. Por falta de dirección no se hacen buenos ánalisis, ocasionando que los programadores y analistas le den la vuelta a un problema de expansión, para evitarse modificar tablas y datos que requieren de conversiones.

Estos gastos superfluos, pero necesarios se pagan subiendo cuotas a los usuarios, con lo que pierde ventaja competitiva con respecto a otros proveedores del mismo servicio.

La tecnología está para usarla de la mejor manera, pero muchas veces no se percatan de lo que se pone en sus manos, hay que ser sinceros y reconocer que las innovaciones tecnológicas no van de la mano con el *software* del que se dispone para explotarla.

Hacer los cambios sistemológicos nos dan ventaja y posición en el mercado, cada vez más competitivo y agresivo en el que se vive, y muchos negocios son comprados por las grandes compañías que si tienen el poder económico para absorber los gastos del mantenimiento y funcionalidad de los sistemas. ¿Qué le pasó a nuestra Banca? Simplemente fué absorbida por las grandes corporaciones extranjeras, ya que las nacionales simplemente no pudieron absorber los gastos de mantenimiento y funcionalidad de que requerían los sistemas.

Capítulo 5

Conclusiones

Concluimos que *en rigor el problema Y2K no está resuelto*, por lo menos para las computadoras digitales de hoy en día. En esta tesis sólo se presenta una, aún cuando no definitiva, muy buena solución desde un punto de vista práctico, la cual resulta económica, de gran durabilidad, al alcance de las necesidades actuales, pero no definitiva. Se puede ampliar la solución por más años, añadiendo más ciclos y épocas, una infinidad de veces bajo el supuesto de que la humanidad vivirá eternamente; por ello, es una lástima ver como sectores tan importantes como el financiero, no busquen mejoras perdurables, que a la postre den robustez y larga vida, dentro la ya tan compleja red de programas que involucran sus sistemas. Por eso recalcamos en *rigor el problema Y2K* ***permanece abierto***, al menos como aquí se ha planteado.

El problema Y2K y su generalización lógica de expanción de campos de una base de datos, es abordado con base en la Sistemología que consiste en hacer un planteamiento global, el cuál a través de un diccionario de datos, genera las referencias correctas del tipo de variable que son incluidos en los sistemas a través de macros referenciales en los programas. Los cuáles necesitan de un pre-compilador que resuelva las sentencias de las macro referenciales, del lenguaje extendido al lenguaje normal, dejando así todo el trabajo tedioso de definición de variables al compilador. Con esto damos trabajos de calidad, por el bien de la sociedad en la que se vive y en la que dá servicio de calidad.

Tan sólo la generalización más sencilla que se dió, el WD, nos permite ver a nuestros sistemas en funcionamiento por más años, y no unos pocos y escasos años de un siglo. Por ello, se pagará en el futuro nuevamente por re-

querimientos del viejo Y2K una cantidad inimaginable de dinero, pudiénsose haber evitado, si se hubiéra planificado bien y analizado mejor; realizando buenas implementaciones, y no sólo lo que fué más rápida al momento, pero que a la postre resultan ser lo más caro.

Este problema nos hizó ver que la *mala planificación, organización y programación* hace que se monten o liberen sistemas defectuosos, cuyos costos como se han publicado son astronómicos, y no es justo que se time a una sociedad que debe mejor invertir en menesteres mas útiles. Si vivimos en una sociedad en la que es necesario que se invierta en salud, agricultura y sobre todo en educación, hace de éste un gasto injustificado.

Si tan sólo se diseñáran mecanismos de inversión entre la Universidad y el sector financiero, es claro que se harían mejor las cosas, a bajo costos. Y sobre todo, se vería la utilidad y necesidad de nuestras instituciones públicas de educación superior, para el desarrollo del país, que en estos tiempos de crisis que se necesitan con premura y verdadera urgencia.

La vida moderna depende mucho de la tecnología, pero es una realidad que muchas personas que dirigen el uso y desarrollo de ella, no están completamente capacitadas, enteradas de los problemas de fondo, conllevando a una mala organición directiva, y por ende a hacer gastos superfluos e inesesarios.

El problema Y2K ya pasó y se está pagando, con altos costos de los servicios bancarios. Costos que en el futuro se incrementarán y harán poco competitivas a la Banca, viéndose, por tanto, con insumos insuficientes para hacer frente al agresivo mercado moderno.

Si eso le paso a la Banca. ¿Dejaremos que le pase a la industria y demás instancias necesarias para el desarrollo moderno y sustentando de nuestro país?

El problema Y2K es un caso recurrente de un problema general, que se tiene cuando se expanden las variables de un campo. Cambio fácil sistemológicamente, pero difícil de hacer en la práctica.

A medida que los sistemas se van haciendo más globales, él no tener mecanismos también globlales de solución, como es el sistemológico, nos hace ver la gran debilidad estructural que sufren estos sistemas, que si no son corregidos, crearán grandes incomodidades por no satisfacer las demandas que día a día, se ve sometida la alta tecnología que presta servicios e insumos a todos.

Por último, nuestros sistemas económicos, políticos y sociales deberían abrirse, para ayudar al prójimo y no tratar de estar por encima de él, creando los mecanismos de inversión que en esta tesis se plantearon.

Gracias por habernos acompañado en este trabajo profecional, que espero, sirva muchos para abrir nuevas fronteras del *sofware* y donde parezca que todo se resuelve con una compilación. Vivir en un mundo en el que trabajen las máquinas y descanse el hombre.

Capítulo 6

Epílogo

"Se supone a menudo que la tesis de alguién deberá ser algo de lo mejor que la persona pueda hacer y dará la justa medida de lo que es. Sin embargo, la tesis no es más que un trabajo específico con el cuál el obrero que ha terminado su aprendizaje acredita el dominio de su oficio, y si no puede superar esto en el curso de su carrera, en verdad es muy ineficiente. Sé que muchos creen que la tesis debe destacar por años, pero esto es ignorado muy frecuentemente en la práctica. Es sólo cuando la persona deja atrás su tesis y ya no es molestado adelante con nuevos requesitos formales, que puede hacer su mejor trabajo como persona libre, con sus propias tareas y metas, y no como algo espurio de una cierta posicion académica o social. La tesis debe ser buena, pero si en su trabajo no se supera pronto el valor de la tesis, el candidato se convertirá en un fósil viviente de los que abundan en las facultades universitarias de nuestros días" [B07].

Norbert Wiener.

Como se ha visto este trabajo se ha pulido a veces de forma fácil, a veces de forma difícil pero es un trabajo por el cuál se luchó y se apuró para que lo leyeran ustedes mis sinodales y también mi familia. Si llegara a tener más público me sentiría muy satisfecho, si no, esto sería una gran lástima. Es sólo una tesis que pretende ser un trabajo original, como siempre se pretendió durante mis estudios en la Facultad de Ciencias de la UNAM. Este trabajo lo saqué de mi desempeño diario, haciendo de él una metodología en mi vida y que me ha permitido llegar lejos, tan lejos como España y Alemania.

Sólo quisiera que las aprovechen ustedes y sean dignos profesionistas que como yo, me enorgullece ser universitario UNAM.

Bibliografía

[B01] Federico Gimeno, *Año 2000 Sus repercusiones en el área de los Sistemas de Información*, Ra-Ma, México D.F. 1999, ISBN 970-15-0370-8.

[B02] W. Michael Fletcher, *La Crisis de las Computadoras del año 2000*, Diana, México D.F. 1998, ISBN 968-13-3110-9.

[B03] Sunny Easwaran, *Year 2000 Mainframe Survival Guide*, Prentice Hall PTR, USA 2000, ISBN 0-13-010481-7.

[B04] Curtis Garvin y Steve Eckols, *DB2 for the COBOL Programmer*, Mike Murach Associates,Inc. USA 1999 , ISBN 1-890774-02-2.

[B05] James R. Newman, *Sigma, El mundo de las matemáticas*, Grijalbo, Barcelona 1997, ISBN 84-253-0250-1.

[B06] Richard Mankiewiscz, *Del Cálculo al Coas, Historia de las Matemáticas*, Paidós, Barcelona 2000, ISBN 84-493-0951-4.

[B07] Norbert Wiener, *Ex Prodigio mi Infancia y Juventud*, Consejo Nacional de Ciencia y Tecnología, México D.F. 1982, ISBN 968-823-109-6.

[B08] N.V. Efímov, *Geometría Superior*, MIR, URSS 1984

[B09] Alan Freedman, *Glosario de Computación*, ISBN 968-451-575-8

[B10] Jesús Mosterín, *Los Lógicos*, Espasa, Madrid España 2000, ISBN 84-239-9755-3

[B11] *Universo Astronomía y Astronáutica.* Abril 1998, página 48.

[B12] Charles Petzold, *Programación en Windows 95*, Mc. Graw Hill, España 1996, ISBN 84-481-0723-3.

[B13] Paul Davies, *Los Mitos de la Materia*, Mc. Graw Hill, España 1994 ISBN 84-481-1754-9.

[B14] Kurt Gödel, *Obras Completas*, Alianza Universidad, Madrid España 1989 ISBN 84-206-2286-9.

[B15] Roberth Johnson, *MVS Manual para programadores*, Mc. Graw Hill, Madrid España 1993 ISBN 84-481-0092-1.

Apéndice A

Glosario Técnico

ABEND ABnormal ENDs.

Sinónimo de aborto, terminación inesperada de un programa, causado por una anomalía en la ejecución del proceso, debido a que no reconoce una operación; por ejemplo división por cero ó por violar una limitación del sistema, lo cúal hace que este decida terminarlo, debido a que la computadora se ve imposibilitada ha ejecutar una instrucción ó procesar cierta información.

El proceso también puede ser interrumpido por el operador, si este decide que el programa ha caído en un ciclo, o sí ha excedido del tiempo usual asignado por el proceso. En este punto puede que se requiera apoyo de un área especial de seguimiento que autorize dicha cancelación ya que se debe considerar que se hará en caso de volver a relanzar la cadena. Esto puede implicar tiempo de reproceso, duplicidad de información o pérdida de la misma.

Esta operación también la puede tomar la computadora y su Sistema Operativo, a un aborto se le llama también una caídá del programa, un *Crach o Bomb.* Al considerar la complejidad de una computadora ¿Cómo es posible que no sea tan común una caida de esta? tan solo la memoria puede contener unos 288 millones de celdas de almacenamiento en uno de sus dos estados activo o inactivo I-O, al trascurrir un segundo, millones de estas celdas cambian de estado, si la COMPUTADORA no dispone de mecanismos internos de Detección y Correccion de Errores, el cambio accidental del estado de una sola celda en 288 millones podría ocacionar un arruinamiento de toda la actividad del equipo, por ello las áreas de Soporte Técnico monitorean y de forma periódica se ven en la nececidad de Apagar todo el equipo o dar IPL.

ALGORITMO.
Método utilizado en la resolución de un problema. Se puede ver como un conjunto detallado de acciones lógicas de solución, esto puede incluir todo el proceso de cálculo de alguna proposición, enunciado, relación o problema.

BACKUP Respaldo.
Respaldo para casos de emergencia, para ello se realiza una copia de seguridad ya sea de un archivo o de una base de datos. La experiencia ha demostrado que una mala gestión de los mismos causa grandes dolores de cabeza ya que no es tan fácil recuperarla y más cuando hablamos de millones de datos. Los usuarios deben de seguir los lineamientos, las políticas de respaldo y vigencia de archivos; ya que actualmente se vive una crisis en cuanto a la gestión de archivos, su almacenamiento y recuperación, ya que lamentablemente cuando se requieren resultan de muy poco valor práctico, pero eso si, de un alto costo de almacenaje y gestión.

COBOL COmmon Business Oriented Language.
Lenguaje para negocios comunes, lenguaje de programación utilizado principalmente en aplicaciones de negocios. El COBOL es un lenguaje de compilador, fué uno de los primeros lenguajes de alto nivel desarrollado. Surgió a mediados de la década de 1950, apartir de un lenguaje llamado *Flowmatic*; pero fué adaptado formalmente hasta 1960. COBOL ha sido desde entonces el lenguaje por excelencia de los sistemas *Mainframes,* es decir para aplicaciones de negocios con grandes volúmen de información a procesar. Además de tener amplio uso en las microcomputadoras.
COBOL permite normalizar la documentación, como consecuencia de esto resulta fácil de entender ya que las palabras son comunes referentes al proceso que se realiza. Un programa COBOL se estructura en cuatro divisiones:

- IDENTIFICATION DIVISION identificá al programa.
- ENVIRONMENT DIVISION define la computadora fuente y objeto.
- DATA DIVISION, define el área de memoria, constantes y demas áreas de trabajo.
- PROCEDURE DIVION, describe la lógica propia del programa.

CICS Customer Information Control System.

Sistema de control de información sobre clientes. Es el sistema desarollado por IBM para la operación en línea de las aplicaciones para las terminales tontas. Este sistema permite la creación en linea de un medio ambiente, multiusuario, dentro de los sistemas operativos de procesamientos por lotes de IBM. El CICS tiene una gran variedad de comandos y características propias; el personal que desarrolla aplicaciones bajo CICS se le denomina "programador CICS o en línea", es sistema CICS se puede considerar un Monitor de comunicación remota.

DB2 Data Base 2.

En 1970 Dr. E.F. Codd desarrollo un modelo de base de datos relacional. Esta primera versión fué elimida por diversos problemas que tuvo, de esta primera versión se eliminaron las redundancias hasta hacerlo versatil y efectivo dando paso al nuevo DB2.

DB2 es el manejador de Base de Datos por excelencia de las aplicaciones que manejan un volumen alto de datos. Sus estandares (SQL) y modos de acceso se pueden aplicar en distintas plataformas de desarrollo como son *UNIX y los Mainframes.*

DB2 es el principal administrado de Base de Datos de sistemas (DBMS) para *Mainframes* de IBM que operan bajo el sistema operativo MVS.

HANDLE Manejador.

Cuando un archivo es abierto de forma física se asigna una dirección de memoria llamada *Handle*, para poder hacer referencia lógica a los archivos, de esta forma se direccionan las funciones básicas de escritura y lectura de ellos.

POLIMORFISMO Sobrecarga.

Caracteristíca de una sola función que se puede ejecutar mediante argumentos de una variedad de tipos. Identificador de procedemiento o método que denota más de un procedimiento.

ROLLBACK Regresar.

Expresión técnica que hace alución a deshacer un proceso, para volver a un punto de inicio, cuando se a cometido un error y garantizar, que no se procese información de forma errornea.

Apéndice B

Sistemología y Sociedad

Este apéndice está tomado tal cual de la página de Internet www.ing.ula.ve, debido a Roldan Tomasz Suárez. Y se trascribe por su gran vinculación con el tema de este trabajo y con quien comparto su postura.

B.1 Sistemología Interpretativa

"La Sistemología Interpretativa no consiste en un conjunto fijo de proposiciones engranadas lógicamente entre sí, sino en un camino de indagación impulsado por el afán de comprender una cierta problemática. Usted, en estos momentos, está por abordar ese camino.

Imagínese parado al principio del mismo. El camino se abre, se aleja en suaves meandros hacia el fondo y se hunde, finalmente, en un horizonte lejano y enigmático. Al principio del camino se ve el planteamiento de una cierta problemática. Un poco más allá, la comprensión de su unidad. Finalmente, en el límite del campo de visibilidad, se vislumbra la definición de un destino.

La problemática de la Sistemología Interpretativa aparece en la confluencia de dos preguntas aparentemente inconexas: o ¿Qué hace posible la trascendencia holística de los fenómenos? o ¿Qué hace posible la situación de subdesarrollo en Latinoamérica?

Lo distintivo de esta perspectiva inicial es que es una prospectiva: una mirada dirigida hacia aquello que aún no ha sido caminado. La prospectiva dibuja el camino desde afuera (o desde antes) del camino. Y sólo muestra con claridad lo que se halla al principio, dejando en la oscuridad lo que hay más adelante, más hacia el fondo. Veamos, con mayor detalle, en qué consiste la

problematicidad de estas dos cuestiones".

B.2 La Pregunta por la Trascendencia Holística

"El enfoque de sistemas nace en este siglo como el clamor por concebir y estudiar los fenómenos como "sistemas" y no como "meros agregados de partes". Este clamor se opone al método reduccionista de la ciencia moderna, según el cuál los fenómenos son estudiados mediante su descomposición analítica en elementos simples, y son explicados como la suma de las propiedades de tales elementos.

Este tema fué inaugurado por Ludwig Von Bertalanffy (1901-1972), considerado como el padre del enfoque de sistemas.

Rene Descartes (1596-1650) es considerado como el precursor del método de la ciencia moderna. En oposición a esto, el enfoque de sistemas muestra que los fenómenos presentan una característica especial de la que carece la mera reunión de sus partes. Esta característica especial es la unidad (el carácter sistémico o sentido holístico) de cada fenómeno, es decir, el hecho de que sus partes conforman 'una' cosa. Dado que tal unidad va "más allá" de la mera reunión de partes, se puede decir que los fenómenos manifiestan trascendencia holística.

Esto no sólo significa que los fenómenos no pueden ser reducidos a la suma de sus partes sino que, además, tal reducción pierde de vista lo más importante del fenómeno: su unidad, lo que lo hace ser lo que es.

Cuando vemos una silla, ésta se presenta inmediatamente como "una" y como "silla".

Pero si viéramos sus partes desconectadas unas de otras, apiladas desordenadamente en un rincón, no veríamos ni 'una' ni 'silla'.

No es pues, la simple reunión de las partes lo que dá la unidad. ¿Qué es lo que dá unidad, entonces?

Pero el movimiento de sistemas, sin antes medir el alcance de su propia crítica a la ciencia moderna, se apresura a conceptualizar la trascendencia holística en términos de la Teoría de la Propiedad Emergente. Según esta teoría, el sentido holístico de los fenómenos es producto de las relaciones que establecen entre sí sus partes. En otras palabras, un conjunto de elementos se presentará o no como unidad dependiendo de su organización interna. Esta

teoría, sin embargo, presenta una serie de deficiencias.

El ejemplo anterior parece indicar que las partes de la silla se presentarán o no como "una silla" dependiendo de si están o no organizadas (u ordenadas) de un modo particular.

Según esto, la unidad "emerge" de las relaciones entre las partes.

Un primer problema es que la "emergencia" misma no logra ser explicada satisfactoriamente. ¿Por qué y cómo una cierta organización particular de elementos tendría la facultad de "crear" unidad?

Por otra parte, dado que las partes de los fenómenos también son unidades, es de suponer que ellas emergen, a su vez, de la organización de sus partes (y así sucesivamente). Siendo así, o bien hay que suponer que los fenómenos se componen de unidades elementales e indivisibles, o bien que son puras relaciones. Si son lo primero, ¿Cómo pueden ser estas unidades sin partes? Si son lo segundo, ¿Cómo puede haber relaciones sin unidades que se relacionen?

Finalmente, esta teoría contradice la elemental intuición de que la unidad de los fenómenos, lo que los hace ser lo que son, depende del sentido que ellos tienen para nosotros. Personas de distintas culturas verán distintas cosas en un "mismo" lugar. La unidad de una cosa depende, pues, de la perspectiva particular desde la cuál mira el observador y de los fines y valores asociados a ella. Esta perspectiva sitúa lo que aparece en un contexto, dándole o no sentido dentro del mismo. Y tal perspectiva, a su vez, es constituídá por el cúmulo de experiencias previas que el observador ha tenido.

El que una silla se presente como "una" y como "silla", parece estar muy asociado al hecho de poseer una "función" en nuestra vida.

Pero una "función" es una construcción cultural. Alguien criado en una cultura en la cuál esta "función" no existe, probablemente no verá ni "una", ni "silla" donde nosotros sí la vemos.

Tanto la noción de "contexto" como la de "cúmulo de experiencias previas" apuntan hacia algo difuso y "externo" a los fenómenos que les brinda unidad. Según esto, el sentido holístico de los fenómenos no radica en nada "interno" a ellos (sus partes o su organización), sino en su "encajar" con una especie de trasfondo. Siendo así , la Teoría de la Propiedad Emergente, sigue atrapada en el reduccionismo, pues piensa la trascendencia holística como producto de algo "interno" a los fenómenos.

¿Hasta qué punto, entonces, habría que cuestionar las bases conceptuales del reduccionismo para darle pleno sentido al clamor del enfoque de sistemas? ¿Cómo fundamentar de manera rigurosa las intuiciones anteriores referentes a la trascendencia holística de los fenómenos?"

B.3 La Pregunta por el Subdesarrollo

"Desde hace ya mucho tiempo nos hemos acostumbrado a decir que los países latinoamericanos son "subdesarrollados" o que están "en vías de desarrollo". Mediante esta expresión nos comprendemos a nosotros mismos como naciones que van a la zaga de un proceso histórico esencial a la humanidad y que consiste en el progresivo mejoramiento de sus condiciones de vida. Gracias a esta imagen de "atraso" nos resulta natural pensar que el progreso, en nuestras latitudes, debe darse por vía de la importación de los "avances" de los países "más desarrollados".

Esta idea moderna de progreso (junto con la imagen que ella instaura de la relación entre los países desarrollados y los subdesarrollados), ha nutrido, desde el siglo XIX, los proyectos políticos de Latinoamérica. En ese sentido, nuestras instituciones públicas son producto del intento por transplantar a nuestro suelo las instituciones de la Europa moderna y su propósito formal es el de contribuir con el desarrollo (o "modernización") de nuestras sociedades.

Sin embargo, este proceso, lejos de avanzar con pasos cada vez más firmes y producir los frutos esperados, en muchos aspectos parece haberse vuelto problemático.

Por una parte, el comportamiento de nuestras organizaciones públicas no se ajusta a los propósitos formalmente establecidos. Una observación más atenta revela incluso que, en muchos casos, estas mismas organizaciones contribuyen con la perpetuación de las condiciones de subdesarrollo que ellas están llamadas a eliminar.

Extrañamente, esta "esquizofrenia institucional" suele pasar desapercibida en la mayoría de los casos. Y en las pocas ocasiones en las que surge como problema es conceptualizada en términos de una simple "falla administrativa" (perdiéndose de vista los aspectos culturales asociados a los intentos de transplante de prácticas sociales).

Un claro ejemplo de "esquizofrenia institucional" lo proporcionan nuestras instituciones penitenciarias latinoamericanas.

Lejos de cumplir con el papel formal de "rehabilitar" al delincuente, la permanencia en estas instituciones tiende a reforzar actitudes anti-sociales en los internos.

Lo anterior sugiere que el rotundo fracaso del proceso de modernización en nuestras sociedades está asociado a nuestra propia incapacidad para comprender el sentido de este proceso y la naturaleza de los obstáculos que encuentra a su camino. Tal incapacidad se manifiesta tanto en nuestra sis-

temática ceguera ante la esquizofrenia institucional, como en nuestro modo simplista de abordarla, una vez que la vemos. Nos topamos pues, en el fondo de este problema, con la ausencia de un modo de pensamiento que nos permita, como sociedad, hacer sentido de nuestra propia condición. He aquí algunos autores que han pensado más a fondo en este sentido: Mariano Picón Salas, Mario Briceño Iragorry, Enrique Bernardo Nuñez, José Manuel Briceño Guerrero, Luis Britto García, Leopoldo Zea, Eduardo Galeano, Octavio Paz.

Pero a esto se le añade una complicación adicional: en las sociedades "avanzadas" también parece haberse desvirtuado el ímpetu modernizador original.

La idea de progreso fué articulada por la cultura europea naciente en el siglo XVIII como pieza clave de la cosmovisión propia de la Modernidad. En aquel entonces se constituyó una nueva constelación de referencias (tales como progreso, racionalidad, libertad, derechos, etc.) que servía como macrocontexto para comprender globalmente el sentido de cualquier cosa que se presentara, incluyendo las acciones humanas. La ciencia y la tecnología modernas también cobraban pleno sentido en referencia a aquellos ideales.

Una comprensión es "global" si intenta dar cuenta del lugar que algo ocupa en la totalidad del ocurrir (o sea, si busca entender la unidad del algo). Es "reducida" si dá cuenta de ese algo sólo en términos de un fragmento del ocurrir (busca sólo un aspecto del algo).

Como forma culturalmente dominante de pensamiento, esta constelación ejerció su poder en Europa a lo largo de varios siglos, e intentó reproducirse en otras regiones geográficas. Sin embargo, el diagnóstico de muchos autores contemporáneos indica que actualmente esta constelación ha entrado en crisis y que sus ideales están dejando de ejercer poder en la cultura occidental.

Lo que va dejando la Modernidad en su retiro es un modo de pensamiento que renuncia a comprender globalmente. Desaparece toda discusión seria acerca de cuál es el bien o los bienes al que tributa todo lo que ocurre. Y la ciencia, lejos de oponerse a tal situación, la legítima y se pone al servicio de un desarrollo tecnológico movido por un afán meramente instrumental.

He aquí algunos autores que han pensado más a fondo el tema de la Crisis de la Modernidad: Jurgen Habermas, Martin Heidegger, Michel Foucault, Leo Strauss, Luc Ferry, Charles Taylor, Alasdair MacIntyre, Anthony Giddens.

De este modo descubrimos que la tarea de comprender la situación de "subdesarrollo" de nuestras sociedades, resulta mucho más compleja de lo

que pensamos normalmente. Tal tarea parece exigir que seamos capaces de comprender globalmente nuestra propia situación histórica. Pero, ¿Cómo podemos comprender globalmente nuestra situación si vivimos un momento de renuncia a las comprensiones globales?"

B.4 La unidad de la Problemática

"Los dos problemas que acabamos de resumir encuentran su raíz común en el modo de pensamiento propio de la ciencia moderna. Ya hemos visto cómo la comprensión que tiene la ciencia de los fenómenos en general no logra dar cuenta satisfactoriamente de la trascendencia holística. Veamos ahora, con más detalle, cómo esa misma comprensión legitima la ausencia de una discusión seria acerca del sentido global de los fenómenos.

La ciencia moderna divide al mundo en dos dominios: lo objetivo y lo subjetivo. Lo objetivo es todo aquello que se refiere a los hechos, a las realidades tales como ellas son en sí mismas, independientemente del significado subjetivo que tengan para alguien. Lo subjetivo, por su parte, es todo aquello en lo que "creemos" pero que no encuentra un sustento firme en la realidad objetiva. Es una suerte de proyección que lanzamos sobre la realidad tiñéndola de significados y valores que ella no posee por sí misma.

En ese sentido, el mundo subjetivo es un dominio de fantasías e ilusiones engañosas que nos impiden mirar la realidad de manera plenamente objetiva. Note que "lo objetivo" sólo puede definirse por oposición a "lo subjetivo" (y viceversa). Esto constituye, desde el punto de vista lógico, una paradoja auto-referencial: para afirmar la existencia de cualquiera de los lados de esta relación hay que presuponer su existencia de antemano.

Esto indica que el dualismo sujeto-objeto es una distinción a priori cuya validez no puede ser demostrada con base en el "método científico". Bajo este esquema dualista, todo juicio acerca del sentido que tiene una acción o una cosa, es subjetivo. Su veracidad no puede ni debe ser discutida pues, por definición, tal juicio carece de objetividad. Por otra parte, al pensar la existencia de los fenómenos como independiente de toda subjetividad, éstos se convierten en potenciales instrumentos de una variedad de fines posibles. Así , el dualismo de la ciencia moderna legitima tanto la desaparición del afán por comprender globalmente, como la aparición de un pensamiento meramente instrumental. Cuando pronunciamos juicios tales como "X es bueno" o "X es hermoso" o "X es triste", intentamos dar cuenta de un posible sig-

nificado de X.

Pero, de acuerdo con la ciencia moderna, no tiene "objeto" debatir la veracidad de tales "juicios de valor". Ahora bien; el método reduccionista y el dualismo de la ciencia moderna están íntimamente vinculados. En efecto, ambos no son sino dos aspectos de una misma suposición de fondo: que todo lo que es, es en sí, es decir, existe de manera independiente de cualquier cosa "externa" a ello. Es esta suposición la que justifica, por una parte, estudiar los fenómenos mediante su descomposición, y, por la otra, pensarlos como objetos independientes de los significados que tienen para nosotros".

B.5 La Vocación del Enfoque de Sistemas

"La discusión anterior permite entender con mayor profundidad la situación en la que se encuentra el enfoque sistemas. Esta situación está signada por el predominio cultural (constituido históricamente) de un pensamiento reduccionista que tiende a destruir el afán por hacer sentido global (u holístico) de lo que se presenta. Tal predominio explica una serie de fenómenos más visibles en nuestras sociedades: la desaparición del debate político, la esquizofrenia institucional, la atomización de la sociedad, la consolidación del mercado como autoridad máxima o la desorientación que muchos experimentamos en nuestras vidas privadas. La forma de conocimiento que consideramos más legítima en nuestra cultura –la ciencia moderna– se revela como absolutamente impertinente, para dar cuenta de los aspectos más vitales de nuestra existencia".

B.6 ¿Cómo Hemos Podido Llegar a Este Punto?

"Frente a esta comprensión de la situación actual, el enfoque de sistemas se reconoce como llamado a recuperar la posibilidad de pensar sistémicamente nuestras propias vidas sociales y privadas. Esta tarea exige ante todo, sentar las bases teóricas de una perspectiva desde la cuál tenga sentido comprender globalmente el sentido holístico de los fenómenos.

Tales bases teóricas, como ya hemos visto, deben trascender la concepción del ser como ser-en-sí y deben dar cuenta del sentido holístico de los fenómenos a partir de la perspectiva, contexto o trasfondo desde el cuál éste

aparece. Esto en otra palabras, significa romper con el dualismo y establecer una unidad entre el ámbito (llamado) objetivo y el (llamado) subjetivo. Aquí la Fenomenología y los métodos hermenéuticos o interpretivistas parecen un buen punto de partida. La Fenomenología, creada por Edmund Husserl (1859-1938), es una corriente filosófica que aspira a examinar los fenómenos permaneciendo fiel al modo como ellos se presentan: siempre en la circuns-tancialidad de un aquí y ahora.

Además, para ser consistentes consigo mismas, estas bases teóricas deben formar una unidad con la comprensión que ellas permiten de la actualidad. En ese sentido, las bases teóricas deben impulsar estudios de fenómenos particulares que arrojen nueva luz sobre las raíces de nuestro presente antisistémico. Tal comprensión debe contribuir a su vez, con el desarrollo de esas mismas bases. Esto, en pocas palabras, significa que el enfoque de sistemas debe construirse a símismo mediante una sostenida crítica del reduccionismo.

Finalmente, debido a su interés por recuperar para el presente la posibilidad de que éste se piense a sí mismo sistémicamente, el enfoque de sistemas está esencialmente comprometido con una labor educativa que permita la difusión socio-cultural de su propia perspectiva como antídoto a la actual ausencia de sentido global. Una pequeña parte de esta labor educativa se lleva a cabo a través del Postgrado en Sistemología Interpretativa de la Universidad de los Andes".

Bibliografía

[A1] Fuenmayor, R.L., "The Roots of Reductionism: A Counter-Ontoepistemology for a Systems Approach", *Systems Practice*, Vol. 4, No. 5 (1991) 419-448.

[A2] Fuenmayor, R.L. "The Self-Referential Structure of an Everyday-Living Situation: A Phenomenological Ontology for Interpretive Systemology". Systems Practice, Vol. 4, No. 5, 449-472. 1991. Plenum Press.

[A3] Fuenmayor, R.L. "Truth and Openness: An Epistemology for Interpretive Systemology". Systems Practice, Vol. 4, No. 5, 473-490. 1991. Plenum Press.

[A4] Fuenmayor, R.L. and López-Garay, H. "The Scene for Interpretive Systemology". Systems Practice, Vol. 4, No. 5, 401-418, 1991. Plenum Press.

Apéndice C

"Se Promete Investigación Objetiva de Accidente de Avión"

De la página de Internet
http:\\www.fpspa.peopledaily.com.cn \200207\04\sp20020704\$_5$5772.*html*

"El presidente de Rusia, Vladimir Putin, dijo el miércoles que el canciller alemán Gerhard Schroeder prometió una investigación objetiva sobre el choque en el aire de un avión ruso con un avión de carga, en el que murieron 71 personas, y en su mayoría niños rusos".

"El martes hablé con el canciller alemán y me aseguró que la parte alemana está haciendo todo lo necesario para realizar una investigación imparcial", dijo Putin durante una reunión en el Kremlin con el ministro de Transporte Sergei Frank y con el fiscal general ruso Vladimir Ustinov.

"Investigadores rusos viajaron el martes al sitio del accidente en donde un avión de pasajeros TU-154 chocó contra un Boeing 757 de carga el lunes por la noche.(Xinhua)"

Apéndice D

Declaración para la Prensa

Accidente de avión, 2 de julio, en Alemania. *De la página de Internet* http:\\www.la-reg.dhl.com\about\nes\Accident*july*2*final**es.htm*

"Bruselas - 2 de julio, 2002 - Aproximadamente a las 11:43 p.m. (LT) del día 1 de Julio,

(5:43 p.m. EST), un Boeing 757-200, operado por DHL International que iba en ruta de Bahrain (vía Bergamo, Italia) a Bruselas, Bélgica se vió implicado en un accidente ocurrido sobre la ciudad de Ueberlingen en el Lago Constance al sur de Alemania.

El accidente ocurrió entre una aeronave de carga de DHL y una aeronave Tupolev con pasajeros a bordo, operado por el carrier ruso Bashkirian airlines. La aeronave DHL estaba tripulada por dos pilotos, uno de Gran Bretaña y el otro de Canadá. Ambos residían en Bahrain. Ningún otro pasajero o tripulante viajaba en la aeronave.

El ministro de transporte de la provincia alemana Bundesland Baden-Wuerttemberg, confirmó anoche que un controlador aéreo en Zurich, responsable del espacio aéreo en donde ocurrió el accidente, le pidió en repetidas ocasiones al piloto de la aeronave rusa que cambiara su altitud ya que se encontraba en un nivel de vuelo incorrecto. El piloto ruso no respondió a dichos llamados.

Frente a la alerta del Boeing 757-200's Traffic Collision Avoidance System (TCAS), el piloto de DHL intentó actuar y tomar una medida correctiva para evitar la colisión, pero no pudo lograrlo".