UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

S I S D O C

SISTEMA PARA LA DOCUMENTACION DE PROGRAMAS

T E S I S

QUE PARA OBTENER EL TITULO DE:

M A T E M A T I C O

P R E S E N T A:

VICENTE GUERRERO ROJO

FALLA DE ORIGEN

México, D.F. 1990

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

DERECHOS RESERVADOS © PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis está protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

TABLA DE CONTENIDO

INTRODUCCION

Dentro de una empresa, es tradicional que en los centros de Informática los Sistemas que son desarrollados pertenezcan a otros centros de la organización. ¿Qué hace la gente de un centro de Informática para si mismos?. La respuesta es sencilla, casi nada. La razón más común de esto es "El centro de Informática es de servicio, y sus actividades están dirigidas al auxilio de los demás centros". Las actividades propias deben ser resueltas en los tiempos muertos o en proyectos de baja prioridad.

Como consecuencia de esto la gente de informática de los centros de cómputo no cuenta con herramientas de calidad para auxiliar el desarrollo de sus actividades.

Para darnos cuenta de esto podemos observar algunas de las actividades más frecuentes que son llevadas a cabo en varias organizaciones, durante el mantenimiento de Sistemas, al aparecer algún error en un Sistema:

- El centro usuario del Sistema da aviso del error.

- El encargado del mantenimiento de ese Sistema solicita la mayor información posible por parte de los operadores del Sistema para tratar de deducir cual es la causa del error. Los errores pueden ser de dos tipo: de operación y del Sistema mismo (alguna de las partes que lo forman).

- Se determina si fue error de operación. Para esto se requiere conocer perfectamente los procedimientos que lleva a cabo el operador del Sistema así como las entradas y las salidas de cada uno de ellos para poder concluir cual fue la secuencia de actividades del operador que generaron la falla. En caso de que no conozca, o no recuerde dichas actividades, debe tener a la mano el manual de operación respectivo.

- Se determina en caso contrario cual es la parte que falló del Sistema. En esta situación es necesario encontrar exactamente si lo que falló fue un programa, archivo, etc. Para esto es necesario conocer como está formado el Sistema a todos niveles, ya que la falla puede estar en cualquier lado. Aun cuando se conozca el Sistema es necesario tener a la mano el manual técnico del Sistema, o parte de él, y en algunas ocasiones el de operación, para determinar con exactitud cuales deben ser los pasos a seguir para reconstruir la situación del error, cuando esto sea posible.

- Una vez detectado el error y la razón de éste, se inicia el procedimiento de corrección el cual puede

implicar el tener que recurrir a las tareas o programas fuente del Sistema o a la descripción de los archivos o bases de datos. Para esto es necesario consultar las carpetas con los listados respectivos.

- Se efectúa la corrección y se prueba cada una de las partes modificadas del Sistema.

- Se actualiza la documentación técnica y de operación respectivas.

Al analizar la secuencia de pasos seguidos para efectuar una corrección podemos observar la necesidad de las consultas a la documentación técnica y de operación del Sistema. Cuando sólo se cuenta con documentación impresa, las actividades de búsqueda y actualización sobre dicha documentación se llevan buena parte del tiempo de atención al problema. Esto si se considera que la creación y la actualización de la documentación es parte de la solución al problema.

Pensando en este tipo de situaciones y con la idea de que al centro de informática no se le dedican los recursos necesarios, se decidió iniciar un proyecto cuyo objetivo fuese el de crear un Sistema que resolviera, en parte, el problema del mantenimiento de otros Sistemas. Tal proyecto consistió en crear un Sistema capaz de almacenar y controlar toda la información necesaria para el mantenimiento de Sistemas. Dentro de sus características se plantearon las siguientes:

- Contener la descripción de cada una de las partes del Sistema a controlar.

- Proporcionar facilidades para el manejo de textos que permitieran manejar la documentación del Sistema.

- Mantener el control de las diferentes versiones de las partes que forman los Sistemas, es decir, control sobre las actividades en el tiempo.

- Proporcionar facilidades para la diagramación.

- Permitir el control de bitácoras de operación, de protecciones, de errores y otras.

- Dar facilidades para entender el Sistema.

- Capacidad de almacenamiento y control de datos de prueba.

Al ir analizando actividad por actividad, se llegó a la conclusión de que era un proyecto ambicioso, por lo que se decidió recortar el alcance y satisfacer sólo las necesidades

más apremiantes. Por esto, lo primero que tenía que ser resuelto sería el control de la documentación, ya que es una de sus partes más importantes. Posteriormente, ya con los resultados de la primera parte, sería reconsiderado el proyecto. Así, se decidió crear una herramienta que permitiera agilizar la consulta y actualización de los manuales, entre otras cosas. Las actividades para desarrollar tal herramienta fueron:

- Revisar cual es el papel de la documentación de Sistemas durante el ciclo de vida de los mismos y principalmente durante su mantenimiento.

- Abstraer desde el punto de vista del mantenimiento lo que es un Sistema.

- Identificar el tipo de preguntas más comunes realizadas al efectuar el mantenimiento de un Sistema.

- Crear un modelo que permitiera almacenar la información del Sistema necesaria para contestar las preguntas antes mencionadas y actualizar dicha información.

- Desarrollar la implantación del modelo y un manejador del mismo.

Para cumplir con estos objetivos se revisaron varios Sistemas desarrollados en COBOL con un ambiente UNISYS y se resumió la experiencia acumulada durante su desarrollo y mantenimiento. El capítulo 1 del presente trabajo tiene por objeto presentar una descripción de los Sistemas tipo más comunes cubriendo su ciclo de vida, su mantenimiento, sus costos y problemas.

Se revisaron cuales son las preguntas más comunes durante el mantenimiento de los Sistemas mencionados. Estas fueron agrupadas y concentradas para obtener sus patrones y el tipo de información necesaria para contestarlas.

Tomando en cuenta el resultado de estos análisis se propuso un modelo de documentación y un manejador para manipular esta información. El modelo y el manejador sufrieron constantes ajustes hasta llegar a lo descrito en el segundo capítulo.

La implantación del modelo y su manejador estuvo apoyada en la utilización de bases de datos y ambientes deductivos. Conforme el diseño de tal herramienta fue avanzando, los alcances de proyecto fueron siendo cada vez menos ambiciosos y más realistas, hasta lograr una implantación final del modelo y su manejador, en lo que llamamos SISDOC. Este se describe en el último capítulo del presente trabajo.

Los anexos tienen por objetivo mostrar el listado fuente de SISDOC, su compilación, las ayudas en línea y por último, una muestra real de la información almacenada por SISDOC.

CAPITULO 1

SISTEMAS

1 Introducción

El objetivo del presente capítulo es presentar una breve descripción de lo que entendemos por un Sistema, las etapas que lo generan y sus principales características. Se hará hincapié en la etapa de mantenimiento dado que es la parte en donde se ha observado se consume la mayor cantidad de recursos y hacia donde está dirigido el objetivo de este trabajo.

2 Sistemas

Algunos diccionarios definen, de manera muy abstracta, a un Sistema como sigue:

- Combinación de partes reunidas para obtener un resultado o formar un conjunto.

- Un conjunto organizado de doctrinas, ideas o principios, usualmente dedicados a explicar el trabajo de un todo.

Como podemos ver en estas dos definiciones, hay diferentes tipos de Sistemas; en realidad casi todo puede considerarse como un Sistema o componente de un Sistema (o ambos). Como también se desprende de las definiciones, algunos de los Sistemas son creados, organizados o mantenidos por los humanos. Ejemplos de tipos de Sistemas pueden ser:

- Los Sociales: organizaciones de leyes, doctrinas, costumbres, etc.

- La Transportación: carreteras, canales, aviación aérea, etc.

- La Manufactura: fábricas, líneas de armado, etc.

- Los Financieros: contabilidades, nóminas, inventarios, etc.

La mayoría de estos Sistemas utilizan las computadoras y es más, algunos no podrían existir sin ellas. Sin embargo, es importante notar que los Sistemas ya existían aun cuando las computadoras no habían aparecido. Hay algunos Sistemas que no están computarizados y aún siguen funcionando y lo harán por décadas. Otros contienen una computadora como uno de sus componentes, al igual que contienen otros componentes no computarizados.

Podemos decir que los Sistemas computarizados son Sistemas creados por el hombre, los cuales interactúan con o son controlados por una o varias computadoras. No hay duda que cada uno de nosotros ha estado en contacto directa o indirectamente con alguno de estos Sistemas y que es posible distinguir varios tipos de Sistemas computarizados.

Aun cuando hay diferentes tipos de Sistemas computarizados, todos ellos tienen componentes comunes:

- Un hardware. Procesadores, unidades de disco, videos, impresoras, etc.

- Un software. Sistemas operativos, bases de datos, comunicaciones, programas de aplicación, etc.

- Datos: la información que los Sistemas manejan.

- Procedimientos: políticas formales e instrucciones para operar los Sistemas.

- Personal: aquellos que diseñan, programan, utilizan y operan los Sistemas.

Una forma de categorizar a los Sistemas puede ser por su aplicación: de manufactura, administrativos, militares, etc, y otra por sus características generales, como pueden ser:

- Operacionales

- De apoyo en la toma de decisiones

- Para la planeación estratégica

- Basados en conocimientos

A continuación describiremos de manera breve que entendemos por Sistemas de estos tipos.

2.1 Sistemas operacionales

Las principales características de estos Sistemas está dada por la forma en que entra y sale la información y el tipo de cálculos que hacen. Estos son relativamente triviales, mientras que los datos (especialmente su estructura y organización) son complejos. Estos Sistemas tienen como objetivo interactuar con las personas, ya sean operadores o usuarios.

La mayor parte de los Sistema computarizados construidos en los últimos 30 años en todo el mundo son Sistemas operacionales, para llevar a cabo la operación cotidiana de las organizaciones. Dentro de estos Sistemas tenemos que

los más comunes son: las nóminas, las contabilidades, los inventarios, etc. Estos Sistemas han sido desarrollados en la mayoria de las empresas, lentamente y a un gran costo, por lo cual en estos momentos representan grandes inversiones.

2.2 Sistemas de apoyo en la toma de decisiones

En la actualidad varias organizaciones están dirigiendo sus baterías a la construcción de Sistemas de apoyo a la toma de decisiones que a diferencia de los operacionales, permiten en forma más directa ayudar a los administradores a tomar decisiones acerca de la organización.

La principal característica de estos Sistemas es la habilidad que tienen para captar información, efectuar cálculos matemáticos sobre ella y presentarla en una variedad de formatos (gráficas, diagramas, reportes, etc). Dentro de este tipo de Sistemas podemos mencionar: las hojas de cálculo electrónico, los paquetes estadísticos y de pronóstico, los Sistemas de información gerencial y otros.

2.3 Sistemas para la planeación estratégica

Estos Sistemas están diseñados para analizar y evaluar la misión de la organización. En lugar de aconsejar acerca de una decisión de un problema en forma aislada, proporcionan consejos generales acerca de la naturaleza del medio ambiente de la organización.

2.4 Sistemas basados en conocimientos

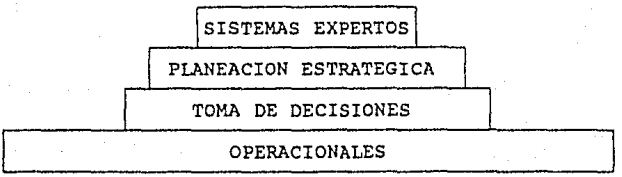
Estos Sistemas, también llamados Sistemas Expertos, son relativamente nuevos en la industria de las computadoras. Tales Sistemas están asociados con el campo de la Inteligencia Artificial y los podemos definir como:

> Programas que tienen los conocimientos y la capacidad para operar a un nivel cercano al de un experto humano. Algunas veces describen las líneas de razonamiento que siguen al tomar una decisión.

Aun cuando los Sistemas expertos están fuera del alcance de este trabajo, es importante mencionar que no está lejano el día en que puedan incorporarse a los Sistemas tradicionales.

La jerarquía que existe entre estos diferentes tipos de Sistemas, la podemos observar en la Figura 2.1, la cual muestra que la base de los Sistemas son los operacionales ya

que son los que proporcionan los datos en forma continua y actualizada a los demás Sistemas.



Tipos de Sistemas

Figura 2.1

Como puede observarse en la Figura 2.1, el tamaño y número de los Sistemas operacionales (medidos en años-hombre, líneas de código, etc.) excede y con mucho al tamaño de los otros. Como ya se había mencionado antes, las organizaciones gastaron gran cantidad de años desarrollando estos Sistemas por lo que ahora están enfocando sus recursos hacia los otros tres niveles. Pero podemos preguntarnos ¿Que se está haciendo por los Sistemas operacionales?. Ellos están allí, produciendo resultados y creando también nuevos problemas, como veremos más adelante.

El objetivo de este trabajo pretende abarcar solamente los Sistemas operacionales, por lo que en adelante los designaremos como los "Sistemas".

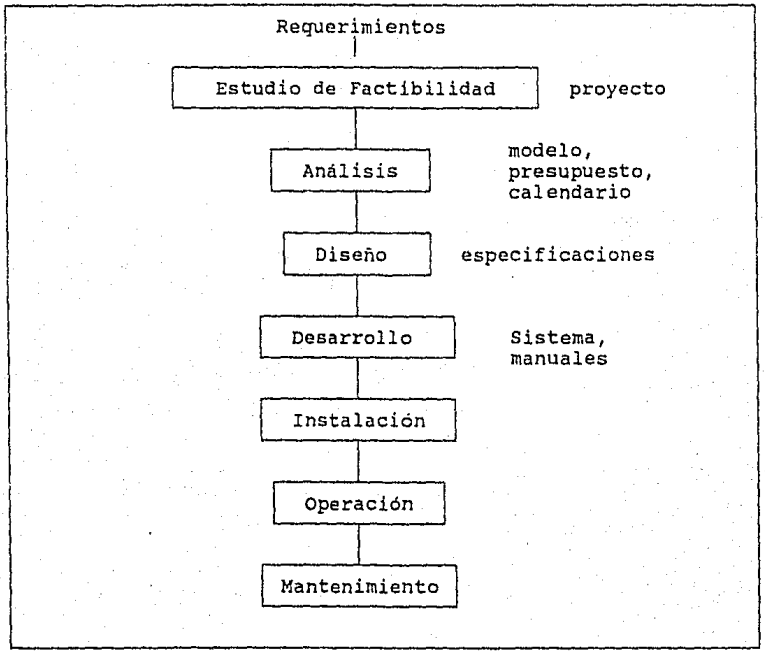
3 Ciclo de vida de los Sistemas

Para la creación de un Sistema es necesario tomar en cuenta las etapas o actividades que son necesarias en su desarrollo. Los términos conjunto de métodos o técnicas, metodologías o ciclo de vida de un Sistema son utilizados para denotar estas actividades. El objetivo principal de estos métodos son:

- Definir las actividades a llevar a cabo en el proyecto de desarrollo.

- Introducir consistencia entre los diferentes proyectos de desarrollo en una organización.

- Proporcionar un medio de control del proyecto a través de actividades bien definidas.



Etapas del Ciclo de vida de un Sistema

Figura 2.2

En la literatura al respecto existen diferentes metodologías: Análisis y Diseño Estructurado, Ingeniería de Software con Flujo de Datos y Estructura de Datos, Prototipos, Diseño Orientado a Objetos. Cada una de ellas cuenta con sus propios métodos y técnicas, sin embargo, podríamos decir que en su forma más simple y clásica, casi todas ellas cuentan con las siguientes etapas:

- Estudio de Factibilidad
- Análisis
- Diseño
- Desarrollo
- Instalación

- Operación

- Mantenimiento

Esquemáticamente estas etapas pueden verse en las Figura 2.2.

3.1 Etapa 1: Estudio de Factibilidad

Dentro de esta etapa se desarrolla un documento llamado Estudio de Factibilidad, el cual permite a los administradores de una organización aceptar o cancelar el desarrollo del proyecto del Sistema. Los principales objetivos de esta actividad son:

- Identificar el alcance del proyecto y sus usuarios.

- Identificar las deficiencias del medio ambiente del usuario.

- Determinar la factibilidad de desarrollar un Sistema, mostrando sus posibles escenarios y un plan de trabajo.

Una vez presentado y aceptado el Estudio de Factibilidad, se inicia el proyecto de creación de un Sistema, con lo cual entra en acción la siguiente etapa, el Análisis.

3.2 Etapa 2: Análisis

El principal objetivo de esta actividad es la creación de un modelo que describa y documente la interacción entre el Sistema y el usuario: se recaban las necesidades del usuario, se determinan las deficiencias en sus procedimientos y se crea el modelo del nuevo Sistema mediante alguna técnica (diagramas de flujo, de flujo de datos, de entidad-relación, de transición de estados, etc.).

Al final de esta actividad se obtiene la estimación precisa del calendario de trabajo, del presupuesto y del costo-beneficio del proyecto.

La mayoría de los autores indican que esta etapa es la de mayor duración en la vida del proyecto. Lamentablemente por gran cantidad de razones, esto no se lleva a cabo así lo que conlleva una serie de consecuencias nefastas durante el mantenimiento.

3.3 Etapa 3: Diseño

El diseño es la etapa encargada de la traducción de los requerimientos de los usuarios y el modelo obtenido en el

análisis en un modelo propio para permitir la especificación de las partes que formarán el Sistema como son: la jerarquía de módulos y sus interrelaciones (Flujos de Información, Entradas y Salidas principalmente), los depósitos de datos (archivos, bases de datos, reportes, etc), las tareas y los programas que intervienen en cada módulo.

3.4 Etapa 4: Desarrollo

Esta etapa tiene por objeto la codificación y la integración de los módulos definidos en la etapa 3, en un esqueleto progresivamente más completo. Los programas ya especificados son codificados y probados por separado. Los módulos son creados mediante la reunión de varias tareas y programas y después son probados también. Al final el Sistema es probado con la participación de los usuarios.

Una parte importante en esta actividad es la terminación de la documentación del Sistema, iniciada en las etapas anteriores; aquí se describe no sólo el Sistema, sino también la forma como interactúan los usuarios con el Sistema. El resultado de esto es la terminación de los manuales de operación y del manual técnico.

3.5 Etapa 5: Instalación

Esta etapa consiste en la liberación del Sistema, instalándolo en el ambiente de los usuarios, haciendo los cambios necesarios para que entre en operación. Dentro de estos cambios está la transformación o carga de la información del usuario al nuevo Sistema; la instalación de nuevos equipos y la presentación de los cambios en los procedimientos de los usuarios. Como producto adicional se incluye la capacitación de los usuarios.

Durante la fase de capacitación de los usuarios (Directivos, personal afectado y operadores) se les instruye en lo referente a las ventajas, limitaciones, entradas, salidas, capacidades, tareas y procesos del Sistema. Para esto son preparados varios documentos (manuales de operación, folletos de divulgación, normas y procedimientos, etc) de los cuales los que más relevancia tienen son el manual de Operación y el manual Técnico. Esto se debe a que un Sistema, o más simplemente un programa, "existe" sólo si tiene documentación [1].

3.6 Etapa 6: Operación

En esta etapa el Sistema es utilizado por los usuarios con los beneficios y las consecuencias de su desarrollo. Aquí es donde aparece uno de los problemas más grandes en la vida

de los Sistemas, el mantenimiento. El mantenimiento es la actividad que permite la continuidad en la operación del Sistema con sus correcciones, adaptaciones o mejoras.

3.7 Etapa 7: Mantenimiento

El mantenimiento de los Sistemas se ha caracterizado como un iceberg, ya que dentro de los centros de informática se conoce la enorme masa de problemas potenciales y costos que esconde bajo la superficie. El costo que implica el mantenimiento de programas ya existentes puede extenderse hasta el 80% [2,3] de todo el esfuerzo empleado en su desarrollo. Este costo se incrementa conforme son desarrollados Sistemas adicionales, y puede llegar a elevarse al grado de suspender, en algunos casos, la creación de nuevos productos. Si consideramos que en las organizaciones siempre hay Sistemas que desarrollar [2], la falta de oportunidad en el desarrollo de esos Sistemas puede provocar serias consecuencias a las organizaciones.

El porcentaje antes citado puede parecer exagerado, pero hay que recordar que los programas siempre están cambiando, tienen problemas que componer, mejoras que agregar, optimizaciones que hacer o simplemente, hay equipos que cambiar. Estos cambios no sólo deben aplicarse a la versión actual, sino también a todas las versiones soportadas.

Además de los problemas ya existentes, están los que las correcciones, adaptaciones o mejoras pueden provocar. Esto simplemente nos indica que los Sistemas no son estáticos, por lo que el mantenimiento se extiende hasta que los Sistemas dejan de operar.

El mantenimiento, como ya se había indicado, es la actividad que permite la continuidad en la operación del Sistema; al corregir y adaptar o mejorar partes de él.

4. Tipos de Mantenimiento

En vista de que SISDOC cobra mayor importancia durante la etapa de mantenimiento, la descripción de esta actividad será extendida para destacar los diferentes tipos de mantenimiento de un Sistema. Por lo tanto, lo podemos definir en forma más precisa como mantenimiento correctivo, adaptativo y perfectivo.

4.1 Mantenimiento Correctivo

El mantenimiento correctivo es el proceso que incluye el diagnóstico y la corrección de los errores. Es la primera actividad del mantenimiento y se inicia al aparecer los

primeros errores en el Sistema. Estos son reportados al personal de mantenimiento, que tiene el deber de corregirlos. Esta actividad es inevitable ya que es casi imposible cubrir todos los posibles errores en los Sistemas antes de liberarlos.

4.2 Mantenimiento Adaptativo

El mantenimiento adaptativo modifica los Sistemas para adaptarlos a su medio ambiente cambiante. Se genera debido a los rápidos cambios que se están dando en los centros de informática. Aparecen nuevas computadoras, nuevos periféricos, nuevas versiones en los compiladores, sistemas operativos y medio ambientes; otros dejan de ser soportados, y dado que la vida útil de un Sistema puede extenderse por varios años éstos pueden quedar fuera del medio ambiente con el cual fueron creados.

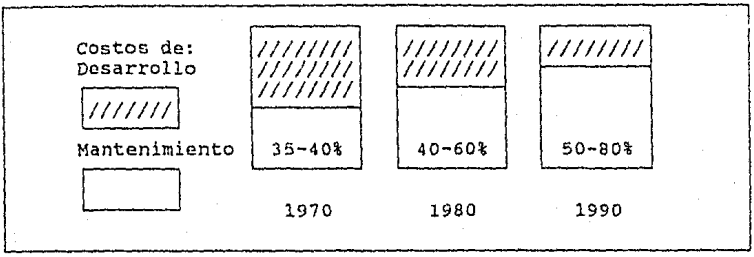
4.3 Mantenimiento Perfectivo

El mantenimiento perfectivo sucede cuando el Sistema tiene "éxito", es decir, funciona. Conforme el Sistema es utilizado surgen las recomendaciones para nuevas funciones, modificaciones a las ya existentes y, en general, mejoras al Sistema, ya sea por propuesta de los usuarios o por cambios en las necesidades.

Estas actividades que conforman el mantenimiento también requieren de algunas de las tareas del ciclo de vida de los Sistemas. Para adaptar o perfeccionar un Sistema se debe determinar los nuevos requerimientos, rediseñar, generar nuevo código, probarlo, documentarlo e instalarlo. Por esto podemos decir que el mantenimiento puede ser dividido en dos actividades: **las productivas** (análisis y evaluación, diseño de modificaciones y codificación) y **las reconstructivas** (entender qué hace el código, interpretar las estructuras de datos, deducir las características de las tareas, etc.).

4.4 Costos de Mantenimiento

El costo de mantenimiento se ha elevado considerablemente durante los pasados 20 años [3]. La Figura 2.3 muestra los porcentajes del costo total del desarrollo y el mantenimiento para un gran número de proyectos. Aunque los promedios son difíciles de obtener, dado el gran número de empresas y personas dedicadas al desarrollo de Sistemas, tenemos que un proyecto típico consume del 50% al 80% de su costo en el mantenimiento correctivo, adaptativo y perfectivo.



Costos de Desarrollo y Mantenimiento

Figura 2.3

Es interesante comentar cual de las actividades del mantenimiento es la que consume más recursos.

El mantenimiento correctivo por su propia naturaleza tiene una de las prioridades más altas cuando se le requiere. Dadas las presiones generadas al aparecer un error en algún Sistema, la corrección es obtenida lo más pronto posible. El esfuerzo empleado puede ser muy grande, pero no así el costo ya que el tiempo dedicado a esta actividad es corto.

El mantenimiento adaptativo puede tener costos muy altos. Sin embargo, dado que se da con intervalos de tiempo grandes (4 o más años), y comparado con los beneficios que producen, sus costos resultan justificados.

Ahora bien, el mantenimiento perfectivo dado que nunca termina, aparece constantemente y por largos periodos, se acumula y por lo tanto implica el mayor costo.

Dentro de los costos mencionados no se toman en cuenta los llamados costos intangibles como son:

- Descontento de los usuarios cuando las peticiones de reparación o modificación no pueden ser atendidas a tiempo.

- Reducción de la calidad de los resultados del Sistema por efecto de los cambios que introducen los errores latentes.

- Atraso o pérdida de oportunidad en el desarrollo de otros Sistemas cuando el personal es dirigido a realizar actividades de mantenimiento.

Ahora bien, dentro de estas actividades de mantenimiento las que más recursos consumen son las reconstructivas y así lo

indican algunos modelos que tratan de cuantificar los esfuerzos en el mantenimiento [3,4,5,6]. Los esfuerzos (y costos) pueden crecer exponencialmente cuando se presentan los problemas del mantenimiento. Algunos de estos problemas se describen a continuación.

5 Problemas del mantenimiento

Los problemas encontrados en el mantenimiento, y que generan grandes esfuerzos, son motivados por varias razones: la metodología utilizada en el desarrollo del Sistema fue deficiente, no se encuentran disponibles las personas que lo desarrollaron o lo mantienen, se carece de una buena documentación. Por lo tanto, los motivos de los problemas que pueden ser encontrados en el mantenimiento se pueden resumir en tres categorías:

- De Metodología
- De Personal
- De Documentación

5.1 Problemas de metodología

La mayoría de los Sistemas no son desarrollados teniendo en mente que serán modificados. Sólo cuando son desarrollados con una metodología que utilice el concepto de modularidad e independencia de los módulos, pueden ser más fáciles de modificar y los problemas de mantenimiento disminuirán.

Cuando la documentación desarrollada en las primeras etapas del ciclo de vida del Sistema, no es integrada a la documentación final del Sistema, gran parte de los costos del mantenimiento serán consumidos en actividades reconstructivas tratando de deducir el análisis y el diseño del Sistema, a través del Sistema mismo. Aun cuando esta información pueda ser deducida, no tendrá la confiabilidad suficiente. Por ello las decisiones que se tomen tendrán siempre cierto grado de incertidumbre e introducirán algunos problemas potenciales en el Sistema.

5.2 Problemas del personal de mantenimiento

Dentro de este problema tenemos algunos como la rotación de personal, la pérdida de experiencia y la aversión a las actividades del mantenimiento.

5.2.1 Rotación de personal

Dentro de la rotación de personal tenemos rotación temporal y rotación definitiva.

La primera sucede con frecuencia cada año y consiste en la ausencia de soporte en los Sistemas cuando el personal de mantenimiento se ausenta de la organización, ya sea por vacaciones o por algún otro motivo. En esta situación la aparición de un problema, cualquiera que este sea, crea un caos por la falta de experiencia en el Sistema, y por lo tanto, el esfuerzo empleado para corregir o reparar el problema es mayor de lo que debería ser. Lo más frecuente en estos casos es que las correcciones aplicadas generarán a su vez nuevos problemas dado que no se conoce bien el Sistema.

La segunda rotación, se da cuando el personal de mantenimiento obtiene una promoción o cambia de organización o área. En este caso, todo el conocimiento y la experiencia del personal se pierde, creando con esto problemas mayores a los comentados en el párrafo anterior.

Esto se minimiza cuando se tiene la suerte de contar con otra persona a quien poder capacitar para sustituir los servicios del personal antes su partida. Otra solución sería contar con una documentación que no sólo cumpla con sus objetivos propios, sino que cuente con características adicionales que le permitan mostrar el Sistema, tanto desde un punto de vista general, como particular a nivel de la operación y del nivel técnico.

5.2.2 Perdida de experiencia

Conforme pasa el tiempo el personal de mantenimiento aprende a conocer el Sistema estudiando sus componentes, pero aprende más conforme atiende los problemas y fallas que éste presenta. De la misma forma, con el tiempo, ese conocimiento se va perdiendo cuando los problemas disminuyen. Si esta experiencia no se plasma en algún lugar, tal conocimiento se perderá; llegando al grado de perder módulos completos del Sistema debido a que ya no se les entiende.

Las otras personas, quiénes lo diseñaron o programaron, posiblemente ya no estén por ahí cuando se les necesite. La movilidad de este tipo de personal es realmente alta, por lo cual, confiar en el conocimiento del diseñador o del programador cuando el mantenimiento es requerido es sumamente arriesgado. Recordemos que algunos errores no aparecen sino hasta pasados algunos años y que otros no son reportados (aproximadamente el 50%), por lo cual no se tiene registro de que existieron y que de alguna forma fueron corregidos.

5.2.3 Aversión al mantenimiento

La aversión al mantenimiento proviene de los grandes niveles de frustración generada durante el mantenimiento, ya que el personal de mantenimiento es sometido a grandes presiones para que corrijan, cambien o reparen los problemas que presentan los Sistemas, sin importar si estos problemas fueron provocados por errores en el diseño, programación o en algunos casos, por la operación misma. Siempre se escucha el siguiente comentario, "Si la operación del Sistema se retrasa, el programador tiene la culpa". Pero si logra la corrección a tiempo, decimos "Ese es su trabajo". Otra de la razones es que generalmente se piensa que el mantenimiento no es creativo por lo cual se abandona esta actividad, provocando consumo de recursos por el constante entrenamiento de nuevo personal y problemas potenciales por su falta de experiencia.

5.3 Problemas de documentación

Cuando la documentación no refleja exactamente el estado actual del Sistema, probablemente es tan desastroso como no tenerla. Esto se debe a que los efectos laterales en mantenimientos posteriores, pueden aparecer cuando otras personas tomen decisiones sobre una documentación desactualizada. También se pueden presentar cuando el usuario encuentre nuevos procesos, entradas o reportes no descritos en su documentación. También cuando encuentre tablas de contenido, índices y textos desactualizados, o mensajes de error no documentados que pueden causarle confusión y desconfianza.

Dado que el objetivo de este trabajo se centra en la documentación de los Sistemas, a continuación se analizarán algunas de las razones por las cuales la documentación, generalmente existente para los Sistemas, no es la adecuada.

La razones las podemos agrupar en falta de documentación, falta de actualización y de referencias cruzadas y el tamaño impráctico de los manuales.

5.3.1 Falta de documentación

Algunas de las razones por las que se presenta esta situación son:

a) En gran cantidad de lugares los programas, y aún más los Sistemas, son desarrollados con un mínimo de especificaciones. A las personas les cuesta trabajo escribir y llevar control de lo que escriben. Las situaciones más frecuentes son:

- El que diseña el Sistema es el mismo que lo desarrolla y no siente que requiere documento alguno. Todod los detalles los tiene en la cabeza.

- No tiene los medios o la experiencia para especificar por escrito las partes del Sistema. Siempre se recurre a "platicar" lo que hará el Sistema y como lo puede hacer.

b) La documentación se inicia sólo al terminar la instalación y prueba del Sistema. También cuando esto sucede ya no hay tiempo para generarla. Los programadores están tan presionados por terminar la programación, pruebas e instalación que la documentación recibe tan baja prioridad que prácticamente es olvidada. Cuando se termina el Sistema, se han olvidado tantos detalles que resulta muy costoso en tiempo y esfuerzo crear o completar la documentación.

c) Por problemas sicológicos o temores. Piensan algunos que al documentar lo que están haciendo podrían prescindir de ellos, o simplemente dejarían de ser importantes.

d) Las especificaciones iniciales no fueron tomadas en cuenta en el programa terminado, lo cual provoca que la documentación se inicie de cero.

e) Falta de procedimientos. También se da el caso que el mismo que desarrolla se quede encargado del mantenimiento por lo se hace innecesario dedicar tiempo a la documentación.

5.3.2 Falta de actualización

Conforme las modificaciones o reparaciones suceden en los Sistemas, los programadores se ven tan presionados en tiempo con correcciones, pruebas e implantación, que resulta difícil la actualización de la documentación. Si a esto se le añade la dificultad de actualizar y obtener otro manual por carecer de herramientas adecuadas para estas actividades, lo más probable será que algunas de las anotaciones que se vayan haciendo queden en cualquier lado menos en donde les corresponde. Con el tiempo se perderán.

5.3.3 Falta de referencias cruzadas

En pocas ocasiones se hacen referencias cruzadas que indiquen:

- Qué programas utilizan qué archivos.

- Qué archivos se utilizan en una tarea determinada.

- Qué campos tiene un archivo y qué almacena cada uno de ellos.

Imaginemos ahora los problemas de estas referencias cruzadas, si existen, cuando se ha ido actualizando el Sistema a través del tiempo y el personal de mantenimiento cree tener la memoria suficiente para acordarse de todos los detalles de actualización. Las referencias pueden provocar decisiones erróneas cuando no están actualizadas.

5.3.4 Tamaño impráctico en los manuales

Muchas veces los manuales son tan grandes que para encontrar el dato que uno busca hay que viajar a través de una masa de datos sin relación. La falta de índices provoca que no se lean los manuales por ser poco prácticos, desperdiciando todo el esfuerzo de documentación. No hay que olvidar que la mayor parte de los manuales son tipo novelas Victorianas, cientos de páginas de texto narrativo sin figuras, diagramas o formatos. Al ser tan voluminosos sólo se reimprimen nuevos manuales hasta que ya no es posible manejar los actuales.

Todos los problemas descritos en esta sección pueden, en parte, ser atribuidos a que se carece de las herramientas adecuadas que permitan minimizar el esfuerzo requerido para documentar Sistemas.

6 Documentación de Sistemas

La documentación de un Sistema es todo aquello que lo describe.

La documentación tiene varios propósitos, primero durante el análisis y diseño, es uno de los principales productos creados por los analistas, diseñadores y usuarios; posteriormente, al instalar el Sistema, es la base para el mantenimiento.

Podemos decir que una buena documentación es aquella que proporciona adecuadas referencias al personal que la utiliza, ya sea cuando requiere información o cuando los problemas se presentan.

La documentación de un Sistema deberá tener por lo menos los siguientes manuales: el de los operadores y el técnico. Cada uno de ellos debe ser creado con el estilo y el lenguaje más

adecuado para los lectores del mismo. Los operadores buscan información sobre el medio ambiente, procedimientos de inicio y de ejecución, así como el calendario de actividades y las salidas a obtener. A los técnicos Analistas y Programadores de mantenimiento, les son más útiles los modelos del Sistema, los listados fuente, las descripciones de archivos, etc.

6.1 Manual de Operación

Una vez liberado el Sistema tendrá que ser operado por el personal adecuado. Este requiere la información de como operarlo en forma cotidiana y como hacerlo en caso de error. Esta información corresponde al llamado manual de operación. Es preparada por los analistas y los diseñadores y completada por los programadores.

El manual de operación en su forma más simple debe contener los siguientes puntos:

- Descripción general del Sistema
- Lista de módulos
- Descripción de cada módulo con:
  - Objetivos y definiciones
  - Calendario y secuencia de ejecución
  - Entradas y salidas
  - Controles y Procedimientos de Emergencia
  - Tareas y su descripción

La descripción general del Sistema debe indicar en forma narrativa el objetivo del Sistema, sus alcances y limitaciones. También debe tener una lista de los módulos que lo forman así como sus interrelaciones.

Por cada módulo se debe indicar sus objetivos y la descripción de las operaciones que atiende; tanto en la parte computarizada, como en la manual. Contiene la definición de los términos del ámbito al que pertenecen las operaciones (contables, financieras, técnicas, etc.).

El calendario especifica las fechas o los periodos en los cuales debe ejecutarse cada módulo. La secuencia indica las actividades que deben ser realizadas, así como las condiciones que deben cumplirse antes de ejecutar cada módulo.

Las entradas y salidas de cada módulo deben ser comentadas mostrando los medios a través de los cuales se obtiene la información (reportes, cintas, discos, archivos, etc.), así como breves descripciones de sus contenidos y de los responsables de entregarlas o de recibirlas conjuntamente con las fechas para realizarlo.

La forma de verificar cuando ha sido terminada la ejecución de un módulo debe ser descrito en la sección de controles y procedimientos de emergencia. Las actividades en caso de error también son mencionadas al igual que los usuarios o áreas afectadas en caso de retraso.

La descripción de cada módulo debe ser complementada con la descripción de cada una de las tareas que forman el módulo. Estas son documentadas al extender cada uno de los puntos siguientes:

- Prerrequisitos y limitaciones
- Procedimientos manuales
- Secuencia de ejecución
- Procedimientos de respaldo
- Entradas y salidas
- Condiciones de error, mensajes y soluciones.

6.2 Manual técnico

El manual técnico tiene un doble objetivo, proporcionar una guía en las primeras etapas del ciclo de vida del Sistema (análisis, diseño y codificación) y posteriormente asistir al mantenimiento. Por su doble objetivo está sujeto a innumerables cambios durante el ciclo de vida del Sistema.

El manual Técnico del Sistema en su forma más simple debe contener los siguientes puntos:

- Alcance
  - Objetivos del Sistema
  - Alcances y limitaciones
  - Medio ambiente
- Documentos de referencia
  - Normas y procedimientos de las actividades del usuario
  - Reglas de operación de las operaciones computarizadas
- Análisis y diseño
  - Estudio de factibilidad
  - Modelos
- Estructuras de datos
  - Descripción
  - Estructura (diccionario, índices, tipos, etc.)
- Módulos
  - Descripción e interrelaciones
  - Implantación
  - Submódulos, supermódulos y tareas
- Tareas
  - Descripción
  - Entradas y salidas
  - Prerrequisitos
  - Programas

- Situaciones de error
- Programas
  - Objetivo
  - Medio Ambiente
  - Entradas y Salidas
  - Funciones y Algoritmos
  - Limitaciones
  - Problemas

La primera sección describe cual es el objetivo del Sistema, quienes son los usuarios del mismo y cuales son las actividades que computariza, así como las que no quedaron incluidas en el Sistema. También describe los equipos, el personal y los insumos que requiere el Sistema para su operación.

La sección de documentos de referencia contiene las citas a los documentos que de alguna forma norman o describen a los usuarios y sus actividades. Por lo general están citados las normas y procedimientos que rigen a los usuarios al igual que las reglas de operación que describen las actividades de los usuarios.

En la sección de análisis y diseño está la referencia al estudio de factibilidad, los diagramas generales y a detalle del análisis del Sistema en donde se indican, entre otros puntos, los siguientes:

- Las personas, organizaciones y Sistemas con los que interactúa el Sistema.

- La información que recibe y la que proporciona el Sistema

- La descripción de los procesos de cada actividad donde se recibe o emite información.

También debe incluir los diagramas del diseño que contengan la estructura del Sistema, sus módulos, las interrelaciones entre ellos y la especificación de los diferentes elementos del Sistema.

La sección de estructuras de datos debe contener por cada depósito de datos (archivos, bases de datos, cintas, etc.) la estructura del depósito, sus índices, métodos de acceso, organización, etc. También, entre otras cosas, debe contener el diccionario de datos que indique la composición de la estructura, tanto a nivel campo como a nivel registro.

En la sección de módulos se tiene su descripción (textual o gráfica), sus interrelaciones y objetivos, así como las partes en que son divididos (submódulos, tareas). Los submódulos son descritos de la misma forma que los módulos. Las tareas se describen en la sección respectiva.

La sección de tareas contiene, por cada una de ellas, sus objetivos, la descripción de como los lleva a cabo, las entradas y las salidas (opciones, archivos, reportes, cintas, etc.) que utiliza o que genera. Menciona también cuales son las condiciones que deben de cumplirse para poder llevar a cabo sus objetivos tales como calendarios de operación, tareas previas, procesos manuales previos, etc. También tiene la descripción de las subtareas que forman las tareas y los programas utilizados en ellas.

La última sección tiene la descripción de los programas del Sistema, haciendo hincapié en los siguientes puntos: su objetivo; medio ambiente (indicando los equipos, compiladores, sistema operativo, etc., necesarios para utilizarlos y modificarlos); entradas y salidas (indicando los archivos, parámetros, subrutinas, reportes, etc., necesarios); la descripción de las funciones, algoritmos o procesos que efectúa para llevar a cabo sus objetivos; al finalizar debe contener las limitaciones y condiciones de error que se manejan.

Observando los problemas antes mencionados, se llegó a la conclusión de que algunos de los problemas del mantenimiento pueden ser minimizados contando con alguna herramienta que facilite la creación, actualización y consulta de la documentación de los Sistemas operacionales. Por lo tanto se decidió crear una herramienta para atacar algunos de estos problemas durante el mantenimiento de los Sistemas. Los problemas a resolver fueron los siguientes:

- Capitalización de la experiencia
- Falta de documentación
- Falta de referencias cruzadas
- Tamaño y número impráctico de los manuales
- Reimpresión y distribución de los manuales

El presente trabajo tiene por objeto proporcionar una herramienta que permita computarizar la creación, actualización y consulta computarizada de la documentación de uno o varios Sistemas de equipos UNISYS 2200 a través de equipos PC-compatibles.

La descripción de la herramienta se deja para el siguiente capitulo.

7 Bibliografia del Capitulo

[1] J.D. ARON
The Program Development Process
Part 1 The Individual Programmer
Addison-Wesley.
Pág. 186-195.

[2] YOURDON EDWARD
Modern Structured Analysis
Yurdon Press Computing Series
Prentice-Hall 1989.

[3] Roger S. Pressman
Software Engineering: a practitioner's approach
McGRAW HILL International Book Company, 1982

[4] Henry C. Lucas Jr.
The Analisis, Design and Implementation of Information Systems
McGRAW HILL International Book Company, 1985

[5] Torguimaraes
Managing Application Program Maintenance Expenditures
Communications of the ACM
Octubre 1983, Vol 26, Num 10
Pag. 739-746

[6] Geral M. Berns
Assessing Software Maintainability
Communications of the ACM
Enero 1984, Vol 27, Num 1
Pag. 14-23

[7] Iris Vessey and Ron Weber
Some Factors Affecting Program Repair Maintenance
Communications of the ACM
Febrero 1983, Vol 26, Num 2
Pag. 128-134

CAPITULO 2

DESCRIPCION DE SISDOC

1 Introducción

El objetivo del presente capítulo es describir el paquete SISDOC, a través de la descripción de un modelo general para los Sistemas computarizados, de la adecuación de la documentación al modelo, así como de su base de datos y su lenguaje de consulta y actualización.

2 Jerarquías

2.1 Jerarquía en los Sistemas

La documentación de un Sistema computacional la podemos describir haciendo una abstracción sobre su estructura jerárquica que contemple varios niveles (Figura 2.1), en los cuales hay una serie de objetos. Los niveles de tal estructura serían: Los Sistemas, Los Módulos, Las Tareas, Los Programas, Los Archivos y sus Estructuras.

- Los sistemas. El primer nivel lo ocupa el Sistema mismo. Encontramos en el la descripción más general del Sistema, sus objetivos, alcances y limitaciones, así como las partes en que está dividido, es decir, los módulos que lo forman.

- Los módulos. En el segundo nivel tenemos a los módulos que forman el Sistema. Son las partes en que se divide un Sistema ya sea por razones de simplificación o propias del problema computarizado. Estos módulos pueden estar formados por actividades manuales, computarizadas (tareas) o combinación de ellas.

- Las tareas. El tercer nivel lo forman las tareas, mejor conocidas como corridas en el ambiente UNISYS, que permiten ejecutar las partes computarizadas de los módulos. Estas corridas son conjuntos de instrucciones en algún lenguaje de control (a veces del Sistema operativo). Las corridas permiten manipular los archivos y ejecutar los programas, entre otras cosas. En este nivel también están las subcorridas las cuales son, a su vez, corridas ejecutadas dentro de otras corridas.

- Los programas. El cuarto nivel lo forman los programas que integran las corridas y son los encargados de procesar la información que reciben las corridas (vía archivos u otros medios) para producir los resultados u

objetivos del módulo al que pertenecen. En este nivel también tenemos a las subrutinas las cuales pueden ser vistas como programas llamados por otros programas.

- Los archivos. En el quinto nivel tenemos los archivos que son los depósitos físicos de la información que manejan las corridas. Son utilizados en forma indirecta por los programas y en forma directa por los comandos del lenguaje de control como pueden ser el "sort", los editores de texto, los copiadores, etc.

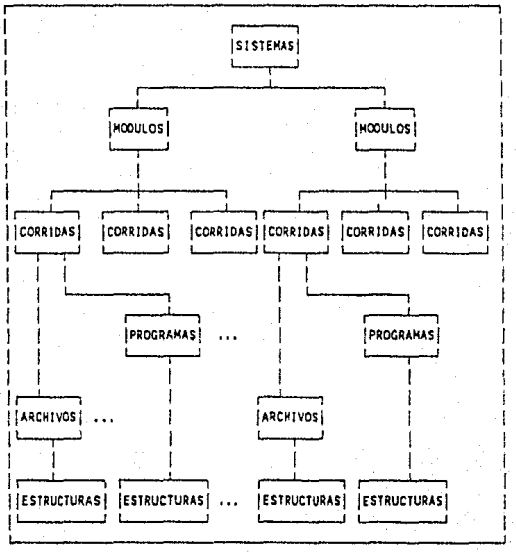
- Las estructuras. En el último nivel están las estructuras que son las descripciones lógicas de los archivos. Cada archivo tiene una y sólo una estructura, mientras que varios archivos pueden tener la misma estructura.

En el ambiente UNISYS los programas utilizan sólo estructuras. El programador es el encargado de asociarle a las estructuras los archivos, de forma tal que un programa puede utilizar, en una misma corrida, diferentes archivos (con la misma estructura) sin necesidad de recompilación.

Por ejemplo, imaginemos un Sistema S1 formado por dos módulos M1 y M2, los cuales están formados por las corridas C1, C2 y C3, C4 respectivamente. A su vez, las corridas utilizan los archivos A1, A2, A3, A4 y los programas P1, P2, P3 y P4. Estos últimos utilizan las estructuras E1 y E2. Este Sistema lo podemos representar en forma esquemática mediante la jerarquía de la Figura 2.2.

2.2 Jerarquía de la Documentación

La documentación de un Sistema la podemos abstraer como un conjunto de documentos aislados donde cada uno contiene la descripción de uno de los objetos en la jerarquía. La asociación que se efectúa la podemos observar en la Figura 2.3, para el manual de operación y la Figura 2.4, para el manual técnico. Ambas indican cuales son los mapeos que se hacen entre las secciones de los manuales de operación y técnico, con la jerarquía de los Sistemas.



Estructura Jerárquica de un Sistema

Figura 2.1

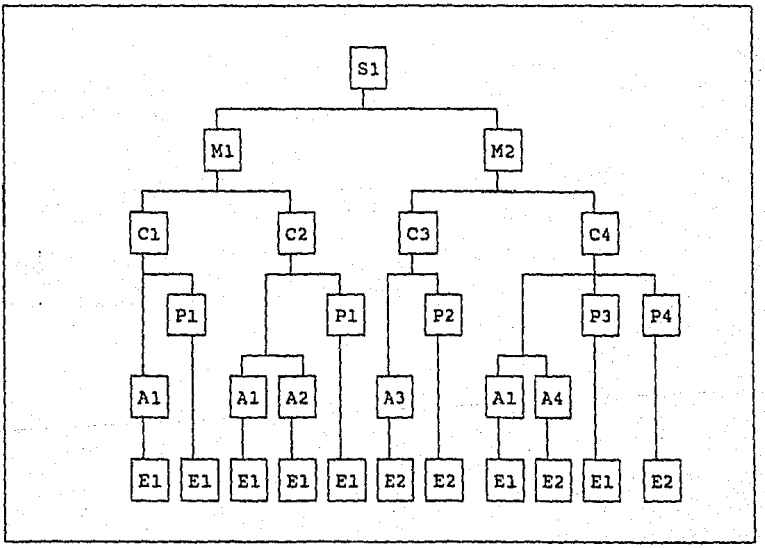
Por lo tanto la documentación (técnica y de operación) de un Sistema la podemos ver como el conjunto de descripciones (documentos) pertenecientes a cada uno de los objetos que existen en los niveles de la jerarquía de los Sistemas, ordenados según esta jerarquía. Por lo tanto, podemos decir que para el Sistema de la Figura 2.2, su documentación consistiría del conjunto de descripciones de los objetos S_i, M_i, C_i, P_i, A_i y E_i, y de la jerarquía respectiva.

La forma de manipular esta abstracción se hará a través de lo que llamaremos el modelo de una documentación.

3 El Modelo

3.1 Modelo de una Documentación

Este modelo está formado por dos tipos de elementos: niveles y objetos. Estos permiten manipular la documentación de uno o varios Sistemas a la vez. El nivel representa la jerarquía de los Sistemas, mientras que los objetos permiten asociar a los elementos de cada nivel el documento que contiene su descripción.



Ejemplo de Sistema

Figura 2.2

OBJETOS A NIVEL	CONTENIDO
sistema	Descripción general del Sistema Lista de módulos
módulo	Objetivos y definiciones Calendario y secuencia de ejecución Entradas y salidas Controles Procedimientos de Emergencia Lista de corridas
corridas	Prerrequisitos y limitaciones Procedimientos manuales Secuencia de ejecución Procedimientos de respaldo Entradas y salidas Condiciones de error, mensajes y Soluciones.

Descripción de la Documentación para Operación

Figura 2.3

3.1.1 Niveles del Sistema

Los niveles permiten describir la jerarquía general de un Sistema e indican a la vez, la naturaleza de los objetos en cada nivel. La descripción de la jerarquía y de los objetos se da a través de atributos. Uno de estos atributos indica cual es el nivel de abajo, mientras que otro representa el documento que es asociado a cada objeto. La sintaxis de un nivel es la siguiente:

$$\text{nivel}(\text{nom-nivel}, \text{Atrib}_1, \text{Atrib}_2, \ldots, \text{Atrib}_n)$$

donde nom-nivel representa el nombre del nivel (sistema, módulo, corrida, etc.), y Atrib_i representa el nombre del atributo que describe a los objetos del nivel nom-nivel.

Los niveles, dado que representan entidades distintas, tienen diferentes atributos. Por lo tanto, el modelo de una documentación requiere de seis tipos de nivel (uno por cada nivel de la jerarquía), con sus propios atributos como se muestra en forma general en la Figura 2.5.

OBJETOS A NIVEL	CONTENIDO
sistema	Alcance Objetivos del Sistema Alcances y limitaciones Medio ambiente Documentos de referencia Normas y procedimientos de los usuario Reglas de operación de las actividades de los usuarios Análisis y diseño Estudio de factibilidad Modelos
módulos	Descripción e interrelaciones Implantación Submódulos, supermódulos y corridas
corridas	Descripción Entradas y salidas Prerrequisitos Programas Situaciones de error Programas
programas	Objetivo Medio Ambiente Entradas y Salidas Funciones y Algoritmos Limitaciones Problemas
archivos	Descripción y Estructura
estructuras	Diccionario, índices, tipos, etc.

Descripción de la Documentación Técnica

Figura 2.4

nivel(sistema, $atrib_{11}$, $atrib_{12}$, .., $atrib_{1n}$)
nivel(módulo, $atrib_{21}$, $atrib_{22}$, .., $atrib_{2m}$)
nivel(corrida, $atrib_{31}$, $atrib_{32}$, .., $atrib_{3c}$)
nivel(programa, $atrib_{41}$, $atrib_{42}$, .., $atrib_{4p}$)
nivel(archivo, $atrib_{51}$, $atrib_{52}$, .., $atrib_{5a}$)
nivel(estructura, $atrib_{61}$, $atrib_{62}$, .., $atrib_{6e}$)

Niveles de un Sistema

Figura 2.5

3.1.2 Los Objetos del Sistema

Los objetos permiten asociar un documento a cada objeto, pero también pueden asociarles más características. Es decir, cada objeto puede estar documentado y descrito por más información que la contenida en un documento. Para describirlos se tienen los valores correspondientes a los atributos del nivel al que pertenecen, como lo muestra su sintaxis:

$$\text{nom-nivel(nom-objeto,valor}_1\text{,valor}_2\text{,..,valor}_m\text{)}$$

donde nom-nivel es el nombre del nivel (sistema, módulo,...) al que pertenece el objeto, nom-objeto es el nombre del objeto descrito y valor_i es el conjunto de valores que lo describen. En su sintaxis se respeta el orden de aparición de los atributos, esto es, el valor_i corresponde al atributo_i. Los objetos están agrupados en lo que llamaremos bases de datos, que reciben el nombre del nivel al que pertenecen los objetos que contienen.

Por lo tanto podemos decir que las bases de datos de una documentación tiene la forma general mostrada en la Figura 2.6.

$\text{Sistema(nomina,valors}_{11}\text{,valors}_{12}\text{,..,valors}_{1n}\text{)}$
$\text{Sistema(bancos,valors}_{21}\text{,valors}_{22}\text{,..,valors}_{2n}\text{)}$
...
$\text{Sistema(finanzas,valors}_{p1}\text{,valors}_{p2}\text{,..,valors}_{pn}\text{)}$

$\text{módulo(m1,,valorm}_{11}\text{,valorm}_{12}\text{,...,valorm}_{1m}\text{)}$
...

$\text{corrida(c3,valorc}_{11}\text{,valorc}_{12}\text{,..,valorc}_{1c}\text{)}$
...

$\text{programa(p7,valorp}_{11}\text{,valorp}_{12}\text{,..,valorp}_{1p}\text{)}$
...

$\text{archivo(a9,valora}_{11}\text{,valora}_{12}\text{,..,valora}_{1a}\text{)}$
...

$\text{estructura(e2,valor}_{e1}\text{,valor}_{e2}\text{,..,valor}_{ee}\text{)}$
...

Objetos en las Bases de Datos

Figura 2.6

Que puede ser interpretada como:

El Sistema nómina tiene las siguientes características:

$atrib_{11}$ = $valors_{11}$
$atrib_{12}$ = $valors_{12}$
...
$atrib_{1n}$ = $valors_{1n}$

El Sistema bancos tiene ...

Para construir el modelo es necesario definir la jerarquia de niveles y dar forma a los objetos de cada nivel. Para esto es necesario seleccionar los atributos que describirán cada nivel asi como definir el tipo y los valores de estos atributos.

3.1.3 Tipos de atributos

Los tipos de atributos que tiene el modelo son dos:

Directos
e
Indirectos

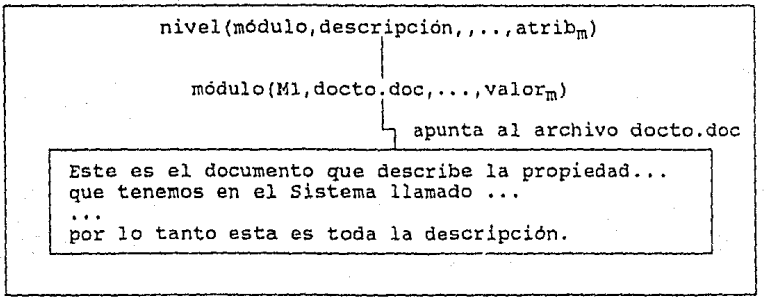
Los directos son los atributos cuyo valor es proporcionado explicitamente. Por ejemplo, para representar que la estructura E1 tiene una longitud de registro igual a 128 y una organización tipo secuencial definimos en el nivel estructura los atributos "longitud" y "organización", y en el objeto E1 correspondiente, colocamos simplemente el "128" y la palabra "secuencial" como se muestra a continuación:

nivel(estructura,longitud,organización,...,$Atrib_n$)

estructura(E1,128,secuencial,...,$valor_n$)

Los indirectos son aquellos atributos cuyo valor, como su nombre lo indica, se obtiene indirectamente, es decir es necesario obtenerlo a través de un proceso de indirección.

El ejemplo tipico lo tenemos con los atributos tipo documento (i.e. descripción textual). En ellos podemos colocar la documentación, es decir, toda la información en texto que deseemos. Para ello necesitamos que el atributo sea definido como indirecto y su valor sea el nombre del archivo donde se encuentra la información. Por ejemplo, si definimos en el nivel módulo el atributo indirecto "descripción", su valor puede ser "docto.doc" que representa el nombre de un archivo donde está realmente el valor (el texto) del atributo (Figura 2.7).



Ejemplo de un atributo indirecto

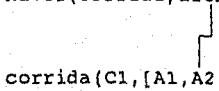
Figura 2.7

Los valores que pueden tomar los atributos de los niveles, en los objetos son de dos tipos:

Monovaluados
o
Multivaluados

Los monovaluados son aquellos que tienen un sólo valor. Por ejemplo, en la estructura E1, antes citada, están los valores "128" y "secuencial".

Los multivaluados corresponden a aquellos atributos que tienen varios valores al mismo tiempo. Son representados como un grupo de monovaluados separados por comas y delimitados por paréntesis cuadrados. Por ejemplo, para representar que la corrida C1 utiliza los archivos A1, A2 y A3 creamos el atributo "archivos" en el nivel corridas, y en el objeto corrida correspondiente a C1, se coloca el valor multivaluado [A1,A2,A3] como se muestra a continuación:



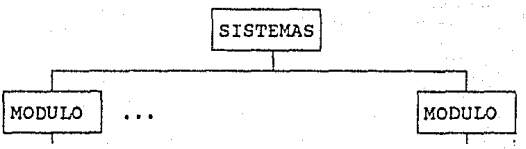
3.1.4 Representación de la Jerarquía

Es importante mencionar la forma en que el modelo representa la jerarquía de los Sistemas. Para esto hay que definir atributos de tipo indirecto en donde el atributo tenga el nombre de un nivel. Por ejemplo, para representar la jerarquía sistema-módulo hay que definir en el nivel Sistema un atributo con el nombre "módulo". Así el modelo queda de la forma siguiente:

nivel(sistema,atrib$_1$,...,módulo,...,atrib$_n$)

nivel(módulo,atrib$_m$,...atrib$_1$)

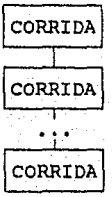
que representa la jerarquía siguiente:



Hay que notar que es válido encontrar dentro de la jerarquía objetos que utilizan objetos que están al mismo nivel. Esto lo tenemos por ejemplo, en las corridas que utilizan otras corridas las cuales son llamadas subcorridas. También lo encontramos en los programas que utilizan otros programas También llamados subrutinas. La forma de representar esto de la misma que con cualquier otra jerarquía. Por ejemplo, el modelo para las corridas quedaría de la forma siguiente:

nivel(corrida,atrib1,...,corrida,...,atribn)

que representa la jerarquía:



3.2 Modelo Mínimo

Una vez observada la forma de definir el modelo de una documentación, podemos decir que la documentación de operación y técnica, típica de un Sistema con la jerarquía mostrada en la Figura 2.1 (sin subcorridas ni subrutinas) quedaría representada mediante el siguiente modelo, al cual llamaremos el modelo mínimo:

```
nivel(sistema,módulo,descripción)
nivel(módulo,corrida,descripción)
nivel(corrida,programa,descripción)
nivel(programa,archivo,descripción)
nivel(archivo,estructura,descripción)
nivel(estructura,descripción)
```

donde "descripción" apunta a los archivos que forman el manual de operación y la nota técnica (separados como se indicó antes). Por lo tanto podemos decir que este modelo coincide y mejora, en cierto sentido los manuales, como se verá más adelante.

3.3 Modelo de SISDOC

Dadas las características del modelo, entre mayor sea el número de atributos que tenga un objeto, mayor será la descripción del mismo, y así, al consultar la documentación el número de preguntas que podrán ser contestadas o relaciones que podrán ser explotadas será mayor.

```
nivel(sistema,módulo,descripción)
nivel(módulo,corrida,descripción)
nivel(corrida,corrida,programa,archivo,descripción)
nivel(programa,estructura,programa,dato,descripción)
nivel(archivo,estructura,descripción)
nivel(estructura,organización,diccionario,longitud,
    descripción)
```

Modelo de SISDOC

Figura 2.8

Por esta razón se definió el modelo de SISDOC, como un modelo para la documentación de Sistemas con más atributos que los que requiere el modelo mínimo. En la Figura 2.8 se observa la versión final del modelo de SISDOC y en la Figura 2.9 la descripción de sus atributos.

Los atributos del modelo de SISDOC fueron seleccionados con base en las preguntas efectuadas más frecuentemente durante

el mantenimiento de los Sistemas (ver ANEXO C). También se tomaron en cuenta las características mínimas que el personal de mantenimiento debe conocer del Sistema, para poder ejecutar en forma controlada las partes de un Sistema en busca del error o falla durante el mantenimiento.

NIVEL	ATRIB	TIPO	VALOR	APUNTA A
sistema	nombre	directo	monoval.	
	módulo	indirecto	multival	módulo
	descripción	indirecto	monoval.	arch txt
módulo	nombre	directo	monoval.	
	corrida	indirecto	multival	corrida
	descripción	indirecto	monoval.	arch txt
corrida	nombre	directo	monoval.	
	subcorrida	indirecto	multival	corrida
	subprograma	indirecto	multival	programa
	archivo	indirecto	multival	archivo
	descripción	indirecto	monoval.	arch txt
programa	nombre	directo	monoval.	
	estructura	indirecto	multival	estruct.
	programa	indirecto	multival	programa
	dato	directo	multival	
	descripción	indirecto	monoval.	arch txt
archivo	nombre	directo	monoval.	
	estructura	indirecto	monoval.	estruct.
	descripción	indirecto	monoval.	arch txt
estruct	nombre	directo	monoval.	
	organización	directo	monoval.	
	diccionario	indirecto	monoval.	arch txt
	longitud reg.	directo	monoval.	
	descripción	indirecto	monoval.	arch txt

Descripción de los atributos de SISDOC

Figura 2.9

Una vez definido el modelo es necesario definir su manejador para poder accesar la documentación.

Antes de describir en forma general el manejador de SISDOC, es importante mencionar que el modelo de SISDOC es estático, es decir, no se puede (al menos en este trabajo) modificar los niveles de la jerarquía, ni la estructura de los objetos. En otras palabras no podemos agregar o eliminar niveles a la jerarquía ni agregar o eliminar atributos de los objetos.

Sólo podemos agregar y eliminar objetos en cada nivel, o bien, modificar los valores de los atributos de cada objeto.

4 Manejador del Sistema

Para explotar la jerarquía, consultar y modificar los objetos de la base de datos fue necesario crear un manejador. Para esto se definieron primero las preguntas que debería contestar SISDOC y con base en ellas se construyeron las funciones correspondientes.

Algunas de las principales preguntas seleccionadas fueron:

¿Cuales son las características de un archivo, programa, etc.?.

¿Si modifico un archivo qué partes del Sistema tendré que modificar?.

¿Qué archivos utiliza el programa P1?.

¿Cuál es la descripción de la corrida C1?.

¿En qué módulos se utiliza el programa P3?.

etc.

Al efectuar un análisis sobre estas y otras preguntas se observó que pueden agruparse de acuerdo a las partes del modelo a las que se refiere, así se puede preguntar:

- sobre los objetos
- sobre los niveles
- sobre la jerarquía
- sobre la consistencia

A continuación se describirán brevemente las funciones correspondientes a estas cuatro partes del modelo. Como veremos las preguntas no utilizan un patrón rígido, sino que podrán formularse en un lenguaje flexible dentro de un cierto rango del lenguaje natural. Los ejemplos están basados en el Sistema de la Figura 2.2 tomando en cuenta el modelo de SISDOC contenido en la Figura 2.5. Las consultas son realizadas por un **Usuario** y las **Respuestas** son proporcionadas por el Sistema con el propósito de dar la idea de cual puede ser el resultado al usar estas funciones.

4.1 Funciones sobre los objetos del Sistema

Estas funciones son las que permiten manejar la información de los objetos del Sistema. Dan el acceso a cada uno de los objetos a través de sus nombres. Las funciones las podemos agrupar en 4 acciones básicas. Estas son:

- Creación
- Modificación
- Eliminación
- Consulta

4.1.1 Creación

Esta función nos permite agregar nuevos objetos a SISDOC. Por ejemplo:

1) **Usuario:** Quiero crear módulos

Respuesta: dame el nombre del módulo a crear

2) **Usuario:** A continuación crea algunas corridas

Respuesta: dame el nombre de la corrida a crear

4.1.2 Modificación

Esta función permite modificar los valores de los atributos de cualquier objeto del Sistema. Por ejemplo:

3) **Usuario:** Quisiera modificar los programas

Respuesta: dame el nombre del programa a modificar

4) **Usuario:** Hay que hacer modificaciones a las estructuras

Respuesta: dame el nombre de la estructura a modificar

4.1.3 Eliminación

Esta función da la facilidad de eliminar algún objeto del Sistema. Por ejemplo:

5) **Usuario:** Quiero eliminar un sistema

Respuesta: dame el nombre del sistema a eliminar

6) Usuario: Deseo eliminar algunos programas

Respuesta: dame el nombre del programa a eliminar

4.1.4 Consulta

Dentro de esta acción tenemos diferentes casos, dentro de los cuales tenemos:

- Para obtener toda la información de un objeto podemos tener el siguiente dialogo:

  7) Usuario: Dame la información de la corrida C2.

  Respuesta: La corrida C2 utiliza los archivo A1 y A2, el programa P1 y su descripción está en el archivo "C2.doc".

- Para tener acceso a los valores de los objetos del Sistema.

  Antes de mostrar el ejemplo, es importante recordar que existen atributos de tipo indirecto en los cuales hay que mostrar no solo el valor sino también el objeto referido por ese valor. Cuando se trata de archivos de texto, el Sistema proporciona un editor de texto que permite consultar, actualizar o localizar el texto. También hay que recordar que en los atributos multivaluados hay que mostrar todos los valores y permitir que se agreguen, eliminen o modifiquen uno o más valores. Por ejemplo:

  8) Usuario: Cuál es la descripción de la corrida C2.

  Respuesta: La descripción esta en el archivo "c2.doc" y su contenido es el siguiente: (Figura 2.7).

- Para mostrar cuáles son los objetos que hay en un nivel dado se puede efectuar la pregunta siguiente:

  9) Usuario: Cuáles son los archivos que tienes.

  Respuesta: A1, A2, A3, A4

- Para obtener los objetos de un nivel que cumplen con alguna condición sobre sus atributos se puede utilizar el dialogo siguiente:

10) Usuario: Dame los programas que tienen la estructura E1.

Respuesta: P1, P3

4.2 Funciones sobre los niveles del Sistema

Estas funciones nos permiten consultar la información de los niveles. Esta información consiste de:

- Mostrar cuales son los niveles del modelo.

11) Usuario: Dame los niveles del modelo.

Respuesta: sistema, módulo, corrida, programa, archivo, estructura

- Mostrar los atributos de cada nivel.

12) Usuario: Cuáles son los atributos de las corridas.

Respuesta: corrida, programa, archivo, descripción

4.3 Funciones sobre jerarquía

Estas funciones sólo nos permiten consultar la información que se desprende de la jerarquía del modelo. Ejemplos de consulta son:

- Mostrar la dependencia, es decir qué objetos de un nivel inferior son utilizados por un objeto de un nivel superior.

13) Usuario: Qué estructuras son usadas en el módulo M1.

Respuesta: E1

14) Usuario: Qué programas llaman al programa P3.

Respuesta: No hay

- Mostrar la contención, es decir, qué objetos de un nivel superior utilizan un objeto de nivel inferior.

15) Usuario: La estructura E1 en qué corridas existe.

Respuesta: C1, C2, C4

16) Usuario: De la corrida C2 dame qué corridas utiliza.

Respuesta: No hay

4.4 Funciones sobre consistencia

Estas funciones nos permiten verificar que los objetos de la base de datos sean consistentes en el sentido de que no exista un objeto aislado. Es decir, que no haya objetos no referenciados por algún otro, u objetos que hagan referencia a objetos que no existan. Estas funciones son realmente importantes al momento de documentar un Sistema, ya que la tendencia a olvidar describir objetos de un Sistema es realmente grande. Las funciones sobre consistencia siempre son consultas, por ejemplo:

- Mostrar los objetos referenciados por algún objeto y que no existan en la Base de Datos.

17) Usuario: Verifica si el módulo M1 es consistente.

Respuesta: el módulo M1 es consistente

- Mostrar los objetos creados en la Base de Datos y que no son referenciados por algún otro objeto.

18) Usuario: Verifica inconsistencias.

Respuesta: El módulo M5 no es consistente
La corrida C8 no es consistente
...

- Mostrar los objetos que no han sido documentados en forma textual, entendiendo por esto que no se les ha proporcionado información al atributo descripción.

19) Usuario: Verifica los programas que si están documentados.

Respuesta: P1, P2, P3, P4

20) Usuario: Qué estructuras no tienen descripción.

Respuesta: E1

Para utilizar las funciones al actualizar, consultar o modificar la documentación es necesario contar con algún lenguaje de consulta. Este lenguaje a su vez requiere de un controlador.

5 Controlador del Lenguaje

La idea original con respecto al lenguaje de consulta fue hacerlo en lenguaje natural. Esta idea conforme fue madurando, permitió determinar que tal objetivo era en si mismo muy ambicioso y por tanto quedaba fuera del alcance de este trabajo. Se decidió entonces que la forma de consultar y actualizar la información en SISDOC solo diera la sensación de lenguaje natural. Para ello se utilizó un chequeo de palabras claves, dejando que el resto del texto, sea ignorado.

Este lenguaje maneja un conjunto de palabras reservadas y cuenta con un diccionario de sinónimos. De esta forma el lenguaje de consulta se hizo más flexible.

Describiremos ahora la sintaxis del lenguaje para las funciones definidas en la sección 4 de acuerdo a los tipos de ejemplos usados:

Para los objetos tenemos:

1) y 2) crea <nivel>

3) y 4) modifica <nivel>

5) y 6) elimina <nivel>

7) <nivel> <nombre>

8) <atributo> <nivel> <nombre>

9) <nivel>

10) <nivel> <atributo> <valor>

Para los niveles tenemos:

11) niveles

12) atributos <nivel>

Para la jerarquia tenemos:

13) y 14) <nivel> <nivel> <nombre>

15) y 16) <nivel> <nombre> <nivel>

Para la consistencia tenemos:

17) consistencia <nivel> <nombre>

18) inconsistencias

19) <nivel> si descripción

20) <nivel> no descripción

donde:

<nivel> representa el nombre de cualquier nivel

<atributo> representa el nombre de cualquier atributo del modelo o un sinónimo.

<nombre> representa el nombre de cualquier objeto del Sistema.

y, las demás palabras son las palabras reservadas del lenguaje.

6 Ventajas de usar SISDOC

La utilización de SISDOC permite incrementar la facilidad de uso y minimizar algunos de los problemas comúnmente encontrados al consultar, preparar y mantener una documentación.

Las ventajas de tener una documentación manejada a través de SISDOC, sobre la documentación tradicional (mediante manuales) puede resumirse en los siguientes puntos:

- Controla la documentación de un Sistema y en forma adicional puede contestar algunas de las preguntas más comunes durante el mantenimiento de Sistemas evitando con esto búsquedas innecesarias sobre toda la documentación.

- Permite conjuntar los dos manuales típicos de una documentación (el técnico y el de operación) en uno solo.

- Permite hacer búsquedas de los objetos que tienen alguna característica en particular, evitando con esto

búsquedas sobre todos los textos relacionados. Esta es una de las actividades más socorridas durante el mantenimiento de Sistemas.

- Al tener la jerarquía, la contención y la dependencia en línea, permite proporcionar una visión tan general del Sistema y tan particular de él, que tanto el personal de mantenimiento novato como el de más experiencia, obtienen beneficios.

- En vista de que para manipular la parte textual se proporciona un editor de texto, la actualización de la documentación o cualquier otro tipo de anotaciones (bitácora de errores, protecciones, etc.), es tan sencillo que no requiere la reimpresión del manual.

- Dado que es posible explorar los archivos fuente de las corridas y los programas, parte de la carga inicial de la documentación puede hacerse en forma automática.

- Cuando se hace una corrección a la documentación, esta se realiza sobre el manual original y no sobre una copia. Por lo tanto siempre se tiene el original actualizado.

- Una vez cargada la documentación en SISDOC, la impresión del manual en su forma tradicional puede ser producido por medio de algún procesador de texto más sofisticado siguiendo las directivas que le podría dejar SISDOC, ya que conoce cuales son los niveles y objetos del Sistema. Esta opción no fue desarrollada en este trabajo.

- Al estar documentando es frecuente olvidar describir las partes que forman a un objeto (más cuando se hace al final de la programación). SISDOC puede viajar sobre la jerarquía del modelo y por lo tanto deducir automáticamente cuales objetos no han sido documentados o utilizados.

- Cuando un Sistema que cuenta con referencias cruzadas es modificado, y sucede que las modificaciones no son reflejadas en su documentación, las referencias cruzadas se convierten en un peligro potencial cuando son utilizadas para tomar decisiones. En SISDOC al explotar la jerarquía, las referencias son automáticamente generadas al momento de su consulta, obteniendo una información más completa.

- Ya que SISDOC controla la búsqueda de los objetos dentro de él, ya no es necesario buscarlos manualmente y por lo tanto los índices y su control dejan de tener objeto.

- Permite que la documentación de los objetos del Sistema sea creada conforme se crean los objetos mismos.

- En vista de que la mayor parte de los productos del análisis, diseño y especificación de las partes del Sistema son documentos, el control de ellos puede dejarse a SISDOC.

7 Bibliografía del Capitulo

[1] Torguimaraes
Managing Application Program Maintenance Expenditures
Communications of the ACM
Octubre 1983, Vol 26, Num 10
Pag. 739-746

[2] Geral M. Berns
Assessing Software Maintainability
Communications of the ACM
Enero 1984, Vol 27, Num 1
Pag. 14-23

[3] Iris Vessey and Ron Weber
Some Factors Affecting Program Repair Maintenance
Communications of the ACM
Febrero 1983, Vol 26, Num 2
Pag. 128-134

[4] Lee L. Gremillion
Determinants of program Repair Maintenance Requirements
Communications of the ACM
Agosto 1984, Vol 27, Num 8
Pag. 826-832

[5] William E. Perry
A Structure Approach to Systems Testing
Prentice-Hall, Inc., 1978
Pag. 341-355

[6] Frederick Brooks Jr.
The Mythical Man-Month
Essays on Software Engineering
Addison-Wesley Publishing Co., 1975
Pag 163-175

[7] Gail Miles
Improving Users Manuals in Software Engineering Education
Lecture Notes in Computer Science
Springer-Verlag, 1989

CAPITULO 3

IMPLANTACION DE SISDOC

1 Introducción

El objetivo del presente capitulo es describir la forma en que fue implantado SISDOC. Se discutirá la selección del lenguaje de programación, se mostrará la forma de implantación del modelo y su lenguaje de acceso, así como otras facilidades que permite SISDOC.

2 Implantación del Modelo

Como se vio en el capitulo anterior SISDOC está formado por una jerarquía (niveles) y una base de datos (objetos).

Una vez definido el modelo fue necesario seleccionar un manejador que permitiera cargar y explotar la base de datos de SISDOC.

Debido a que las principales funciones de acceso a SISDOC son funciones sobre bases de datos, se buscó un lenguaje adecuado para su programación. El lenguaje debía proporcionar facilidades para el manejo de bases de datos y para el desarrollo de una interfaz de usuario que diera la sensación de lenguaje natural.

El modelo de SISDOC podía ser implantado en una base de datos relacional ya que cumplía con dos de las características relevantes de las bases de datos relacionales, como son:

1) Los datos son presentados mediante relaciones.

2) Se pueden generar nuevas tablas, a partir de las ya existentes, a través de operaciones como la selección, proyección y unión de tablas.

También podía haberse implantado en un lenguaje declarativo como Prolog, ya que este lenguaje puede simular las bases de datos relacionales y además proporciona una máquina de inferencias para la consulta.

Con esto en mente tenía que decidirse si desarrollar el manejador en una base de datos "pseudo-relacional" como DBASE III+ o FOXBASE, simularla en Prolog, o bien desarrollar alguna combinación de ellos. Por lo tanto, se buscó qué combinación de bases-de-datos y ambientes-deductivos había, encontrándose los siguientes esquemas:

1) **Administración elemental de datos.** Los datos son mantenidos y manipulados desde un ambiente deductivo (todos los datos permanecen en memoria almacenados en estructuras adecuadas).

2) **Administrador generalizado de datos.** Consiste de una base de datos de propósito general simulada dentro de un ambiente deductivo.

3) **Acoplamiento mínimo** de un ambiente deductivo con una base de datos de propósito general. Parte de los datos son extraídos de la base de datos antes de ser utilizados por el ambiente deductivo.

4) **Acoplamiento máximo** de un ambiente deductivo con una base de datos. Corresponde a un sistema donde está bien integrado el paso de la información desde y hacia la base de datos.

Dado que el alcance del proyecto no vislumbraba ningún desarrollo espectacular del manejador, se decidió seleccionar el primer esquema, es decir, mantener toda la base de datos en memoria, proporcionándole al ambiente deductivo el mínimo de comandos necesarios para manejar la información.

Cabe, sin embargo, mencionar que el manejador de SISDOC no realizaría deducciones.

A continuación se dan algunas de las razones por las cuales se escogió Prolog como el lenguaje más adecuado para implantar SISDOC.

Si hubiese seleccionado un ambiente de base de datos la creación de la interfaz con el usuario sería algo costosa y el manejo de los atributos multivaluados requeriría de bases de datos extras por cada uno de ellos. Los ambientes de bases de datos como FOXBASE y DBASE III+ proporcionan el manejo de campos tipo texto con capacidad de hasta 4 Kb mientras que Prolog permite simularlos de hasta 64 Kb.

La facilidades para el manejo de bases de datos y las facilidades para el manejo del lenguaje de consulta hicieron de Prolog el candidato más adecuado.

Dado que el Prolog que se tenía al alcance era el Turbo Prolog v1.0, éste fue seleccionado como ambiente de programación para SISDOC.

Desde el principio se decidió que el manejador del modelo no tendría las facilidades que permite un manejador de base de datos de propósito general. Se limitaría a contestar las preguntas de consulta más comunes sobre la documentación de un Sistema. Por ello primero fueron seleccionadas las

preguntas y posteriormente fueron desarrollados los comandos del manejador, como ya se comentó en el capitulo anterior.

El manejador considera que el modelo es estático, es decir, que no contempla reconfiguración, por esto no se proporcionan facilidades para incrementar o disminuir el número de niveles o de atributos, ni para hacer búsquedas de trayectorias navegando a través de sus facilidades de contención y dependencia.

2.1 El Modelo de SISDOC

El modelo de SISDOC (Figura 3.1) está implantado utilizando los predicados y las facilidades de base de datos que proporciona el lenguaje Turbo Prolog. Como éste sólo proporciona dos funciones básicas, creación (assert) y eliminación (retract) de predicados, fue necesario desarrollar otros predicados para manipular el modelo.

```
nivel("estructura",["organizacion","diccionario",
     "longitud","descripción"]).
nivel("archivo",["estructura","descripción"]).
nivel("programa",["estructura","programa","dato",
     "descripción"]).
nivel("corrida",["corrida","programa","archivo",
     "descripción"]).
nivel("módulo",["corrida","descripción"]).
nivel("sistema",["módulo","descripción"]).
```

Modelo de SISDOC

Figura 3.1

Los niveles de SISDOC fueron representados mediante un predicado llamado "nivel", con dos parámetros: el nombre del nivel y la lista de sus parámetros. Este predicado permite definir los diferentes niveles, sus nombres y los atributos de cada uno de ellos.

2.2 La Base de Datos

La base de datos de SISDOC (Figura 3.2 y ANEXO D) está formada por los objetos del modelo, que también fueron representados mediante predicados cuyo functor es el nombre del nivel y sus parámetros los valores de los atributos del nivel respectivo en el modelo.

Por cada objeto de un nivel hay un predicado con sus valores respectivos. Estos predicados residen en disco por lo cual

es necesario leerlos a memoria al iniciar la ejecución de SISDOC y escribirlos de nuevo al disco, al terminar SISDOC.

```
sistema("contabilidad",["captura-pólizas",
        "pólizas-manuales","actualización-saldos",
        "cierre-mensual","cierre-anual","procesos-
        auxiliares"],"contab.sis")
...
módulo("actualización",["m-310","balanza",
        "respaldo1","respaldo2","respaldo3","recupera1",
        "recupera2","recupera3"],"módulos2.mod")
módulo("cierre-anual",["ceros","genpol-cierr",
        "listado-anual","genpol-ap","ini-anual"],
        "módulos3.mod")
...
corrida("transmisión-a-univac",[],["pep"],[],"")
corrida("pólizas",[],["pólizas.prg"],[],"módulos5.cor")
...
programa("list-catalog",["catálogos","catálogos2",
        "listado-cat"],[],["opción"],"progs3.pgr")
programa("crea-hist",["movscont"],[],[],"progs2.pgr")
...
archivo("movs-hist","movscont","archiv1.arc")
archivo("1-val-micro","reportes","capt-pol.arc")
...
estruct("movscont","indexada","procs.dic","128",
        "estruct.est")
        estruct("operaciones","secuencial","procs.dic",
        "200","estruct.est")
...
```

Base de Datos de SISDOC

Figura 3.2

2.3 Los atributos

Los atributos utilizados fueron de dos tipos, monovaluados y multivaluados, en sus dos variedades, directos e indirectos. Es importante mencionar que Turbo Prolog es un compilador que maneja tipos. Esto implica que hay que declarar el tipo de cada dato utilizado por los programas. Los atributos fueron representados mediante tres tipos de parámetros de Turbo Prolog: átomos, cadenas y listas de la siguiente forma:

Los monovaluados:

Directos. Fueron representados por los átomos de Turbo Prolog, ya sean números enteros o flotantes o símbolos.

Indirectos. Fueron representados mediante los átomos y las cadenas de Turbo Prolog. Los símbolos se utilizaron cuando el atributo apuntaba a un objeto dentro del modelo. Las cadenas cuando apuntaban a un objeto fuera del modelo, en particular a los archivo de datos. Estos fueron manipulados a través de predicados especiales junto con el editor de texto proporcionado por Turbo Prolog.

Los multivaluados:

Directos e Indirectos. Fueron representados mediante el uso de listas de átomos. Requirieron también de predicados especiales para ser manejados como si fueran campos simples.

La estructura general (declaración) de los predicados se indica en la Figura 3.3. En la declaración cuando el atributo es monovaluado se indica con la palabra "symbol" (átomo) o "string" (cadena), cuando es multivaluado se utiliza "Listsym" (lista). La descripción de cada parámetro puede observarse directamente en el listado fuente del programa en el ANEXO A y en el listado de compilación en el ANEXO B.

```
sistema(symbol,listsym,string)

módulo(symbol,listsym,string)

corrida(symbol,listsym,listsym,listsym,string)

programa(symbol,listsym,listsym,listsym,string)

archivo(symbol,symbol,string)

estruct(symbol,symbol,string,symbol,string)
```

Estructura General de la Base de Datos de SISDOC

Figura 3.3

3 Lenguaje de Consulta

El lenguaje de consulta, como ya se vio en el capítulo anterior, está formado por palabras reservadas. También maneja sinónimos de las palabras reservadas para dar mayor flexibilidad al lenguaje.

De las funciones mencionadas en el capitulo anterior no todas fueron desarrolladas en esta versión. Las funciones no implementadas son: las que permiten manejar el modelo y una parte de las de consistencia.

Las funciones que permiten modificar el modelo no se pusieron inicialmente ya que la primera versión de SISDOC sería estática. Debido a que SISDOC está desarrollado con base en el conocimiento de la estructura jerárquica que maneja, no puede funcionar para cualquier otro modelo.

Por otro lado, la función de consistencia fue desarrollada e implantada, pero al tiempo de pruebas consumió tal cantidad de memoria que fue necesario eliminar parte de ella y dejar el resto como un programa de utilería externo a SISDOC al igual que otras facilidades.

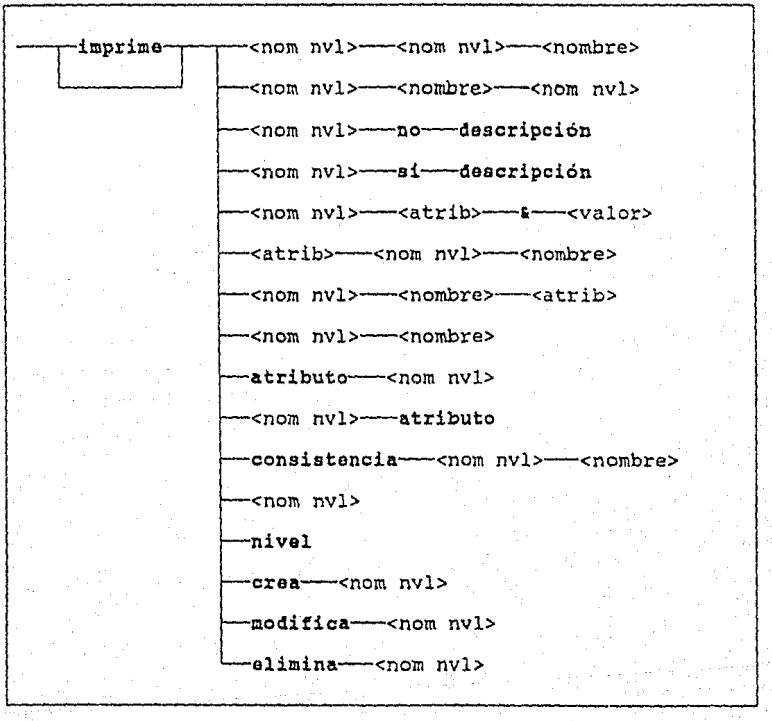
La sintaxis del lenguaje se muestra en la Figura 3.4. Las palabras en negrillas son las palabras reservadas que SISDOC detecta y las demás representan lo que a continuación se define:

<nom nvl> = nombre de cualquier nivel: sistema, módulo, corrida, programa, archivo y estructura.

<nombre> = cualquier nombre de los objetos de los niveles. Por ejemplo en la Figura 3.2 tenemos los siguientes nombres: contabilidad, actualización, cierre-anual, etc.

<atrib> = nombre de cualquiera de los atributos del modelo o uno de sus sinónimos (pueden tener varios). Por ejemplo estructura, descripción, dato, etc.

<valor> = representa el valor del atributo que se busca. Puede ser numérico o alfanumérico por ejemplo: 128, indexada, movscont, etc.



Sintaxis del lenguaje de comandos de SISDOC

Figura 3.4

Ejemplos:

1) Con la sintaxis:

<nom nvl>—<nom nvl>—<nombre>

podemos construir las siguientes consultas:

- qué programas tienen el módulo previo
- qué corridas maneja el sistema financiero
- cuáles son los archivos de la corrida inicial

2) Con la sintaxis:

<nom nvl>—<nombre>—<nom nvl>

podemos construir las siguientes consultas:

- la **estructura e1** en qué **módulos** se maneja
- la corrida c1 en cuáles sistemas está

3) Con <nom nvl>——**no**——**descripción**

ejemplificamos:

- dame los **módulos** que **no** tienen **descripción**
- qué archivos no tienen descripción
- las estructuras sin descripción son

4) Con <nom nvl>——**si**——**descripción**

ejemplificamos:

- cuales son las **corridas** que **si** tienen **descripción**
- qué programas ya tienen descripción

5) Con <nom nvl>——<atrib>——&——<valor>

ejemplificamos:

- dame los **programas** que tengan su **dato** igual a **&fecha**
- qué estructuras tienen su organización &indexada

6) Con <atrib>——<nom nvl>——<nombre>

ejemplificamos:

- cual es la **organización** de la **estructura e1**
- dame las corridas del módulo m1
- imprime la descripción del sistema financiero

7) Con <nom nvl>——<nombre>——<atrib>

ejemplificamos:

- la **estructura e1** que **organización** tiene
- el módulo m1 qué corridas tiene

8) Con <nom nvl>——<nombre>

ejemplificamos:

- dame la información del **sistema s1**
- qué tiene acerca del módulo p1

9) Con atributo——<nom nvl>

ejemplificamos:

- dame los **atributos** de los **programas**
- qué atributos tienen los archivos

10) Con <nom nvl>——**atributo**

ejemplificamos:

- los **sistemas** qué **atributos** tienen

11) Con **consistencia**——<nom nvl>——<nombre>

ejemplificamos:

- verifica la **consistencia** del **sistema** s1
- qué tan consistente es el módulo m1

12) Con <nom nvl>

ejemplificamos:

- dame todos los **módulos** que tengas
- qué corridas hay en tu base de datos

13) Con **nivel**

ejemplificamos:

- cuáles son los **niveles** que tienes

Hasta aquí mostramos ejemplos de consultas, a continuación serán mostrados los ejemplos para crear, modificar o eliminar:

14) Con **crea**——<nom nvl>

podemos escribir:

- **crea** algunas **estructuras**
- hay que crear archivos

15) Con **modifica**——<nom nvl>

podemos escribir:

- **modifica** algunos **módulos**
- quiero modificar las corridas

16) Con **elimina**——<nom nvl>

podemos escribir:

- **elimina módulos**

- quiero eliminar algunos archivos

En estos ejemplos se resaltaron las palabras reservadas de las frases para cada ejemplo. También se utilizaron algunos de los sinónimos de palabras reservadas.

3.1 Diccionario de Palabras Reservadas

Las palabras reservadas que maneja SISDOC son:

atributo
consistencia
crea
elimina
imprime
modifica
nivel
no
si

Además los nombres de los niveles y sus atributos que aparecen en la Figura 3.1 y los nombres de los objetos de la base de datos en la Figura 3.2, y en un caso practico, los del ANEXO D.

3.2 Sinónimos

Los sinónimos como su nombre lo indica son palabras que pueden ser utilizadas en lugar de una palabra reservada. Estas pueden ir aumentando ya que hay opciones que permiten extender su número. El manejo de esto permite que el léxico del lenguaje utilizado sea mayor. Los sinónimos utilizados son almacenados mediante el predicado llamado sinónimos con la sintaxis siguiente:

sinónimos(sinónimo,palabra-reservada)

Como estos predicados son modificables, permanecen en disco y por lo tanto son cargados a memoria al iniciar SISDOC y de nuevo son escritos al disco al terminar SISDOC. Por esta razón pasan a formar parte de la base de datos de SISDOC.

Los sinónimos básicos de las palabras reservadas son:

P. RESERVADA	SINONIMO
archivo	archivos
consistencia	consistente
corrida	corridas
crea	crear
crea	construye

elimina	quita
nivel	niveles
estructura	estructuras
imprime	lista
imprime	reporta
modifica	modificar
modifica	cambia
modifica	altera
modifica	modificación
módulo	módulos
no	sin
programa	programas
sistema	sistemas

Los sinónimos básicos de los atributos son:

ATRIBUTO	SINONIMO
atributo	atributos
dato	datos-de-entrada
corrida	subcorridas
programa	subprogramas
programa	subrutina
programa	subrutinas

Una vez que se han explicado, de manera general, las principales características de SISDOC, procedemos a revisar cada una de sus partes.

4 Descripción del Programa

4.1 Meta de SISDOC

La meta de SISDOC es el predicado **consulta()**. Este tiene por objetivo preparar las ventanas del paquete, cargar los predicados que forman la base de datos e iniciar la ejecución del menú principal. El predicado **consulta()** se muestra a continuación, y en el ANEXO A y B están los demás predicados que utiliza.

```
consulta:-
     makewindow(21,7,0,"",24,0,1,80),
     write("ESC: Fin del menú -- Utilizar las flechas
     para seleccionar la opción."),
     makewindow(22,7,0,"",23,0,1,80),
     write("Esc: Fin      F8: Repite linea   Ctrl S:
     Parar impresión    End: Saltar fin linea"),
     makewindow(2,7,7,"SISDOC: Sistema de
     Documentación/Consultas en Lenguaje Natural",
```

```
0,0,23,80),
proceso(99),
menú_inic.
```

El menú principal que muestra tiene las siguientes opciones:

- Consultar la Base de Datos
- Ver el Lenguaje
- Actualizar el Lenguaje
- Tutorial
- Sistema Operativo
- Almacenar la Base de Datos

- La consulta de la base de datos permite crear, consultar y actualizar la base de SISDOC, mediante el lenguaje de consulta. Es la opción más importante y de hecho la más utilizada.
- Ver el lenguaje permite ver cuales son las palabras reservada y los sinónimos que maneja el lenguaje.
- Actualizar el lenguaje permite agregar más sinónimos.
- EL tutorial nos muestra algunos ejemplos de como consultar la base de datos.
- El sistema operativo nos permite dejar momentáneamente SISDOC, para hacer algo a nivel del sistema operativo
- Almacenar la base de datos nos permite almacenar en disco la información de la base de datos. Esto nos asegura que la información que estamos manejando en memoria esté siendo protegida cada determinado tiempo.

Para terminar la ejecución de SISDOC basta con oprimir la tecla de ESCAPE.

El menú tiene los siguientes predicados:

```
menú_inic:-
     repite,
     menú(8,49,"Menú Principal",
       [ "Consultar la Base de Datos",
         "Ver el Lenguaje",
         "Actualizar el Lenguaje",
         "Tutorial",
```

```
        "Sistema Operativo",
        "Almacenar la Base de Datos"],Selección),
    proceso(Selección),
    Selección=0,!,
    removewindow,removewindow,removewindow.
```

Este predicado tiene por objeto presentar un menú y leer la opción seleccionada. El menú se muestra en una ventana a partir del renglón 8 y la columna 49 con el letrero "Menú principal". Las opciones son las que se presentan en la lista ["Consultar...Datos"]. La opción seleccionada es ejecutada por el predicado proceso(Selección). Para salir del menú basta con digitar la tecla ESCAPE que se interpreta como la opción 0. En caso contrario regresa al punto de "backtrack" e inicia de nuevo.

Los predicados que controlan las opciones se muestran a continuación:

```
proceso(0):-
     write("\nEstás seguro de querer terminar? (s/n):
      "),
     readchar(T),T='s',
     proceso(98).

proceso(1):-
     leeconsul(L),
     itera(L).

proceso(2):-
     veleng.

proceso(3):-
     actleng.

proceso(4):-
     file_str("sisdoc.hlp",TXT),
     display(TXT),
     clearwindow,!.
proceso(4):-
     write(">> sisdoc.hlp no está en el diccionario por
     omisión\n").

proceso(5):-
     makewindow(3,7,0,"",0,0,25,80),
     write("Digita EXIT para regresar\n\n"),
     system(""),!,
     removewindow.
proceso(5):-
     write(">> command.com no está accesible. digita
     RETURN"),
     readchar(_),
     removewindow.
```

```
proceso(6):-
     proceso(98).

proceso(98):-
     write("\nQuieres guardar la base de datos? (s/n):
     "),
     readchar(T),T='s',
     write("\nEspera un momento, estoy guardando la base
     de datos"),
     deletefile("sisdoc.bak"),
     renamefile("sisdoc.dba","sisdoc.bak"),
     save("sisdoc.dba").
proceso(98).

proceso(99):-
     write("Cargando la Base de Datos - favor de
     esperar\n"),
     consult("sisdoc.dba"),!.
proceso(99):-
     write(">> sisdoc.dba no está en el diccionario por
     omisión\n").
```

La opción 1, "Consultar la Base de Datos", consiste en leer, con **leeconsul()**, la primera consulta del usuario en el lenguaje de SISDOC y mandarla como una cadena de caracteres a la interpretación de la misma, con **itera()**.

Las opciones 2 y 3, "Ver y Consultar el Lenguaje", únicamente llaman al predicado encargado de esto, **veleng()** y **actleng()** respectivamente.

La opción 4 "Tutorial" lee el contenido del archivo ayuda denominado "sisdoc.hlp", lo convierte en texto y lo despliega en la pantalla mediante el editor de texto. En caso de que no exista tal archivo manda un mensaje de error.

La opción 5, "Sistema Operativo", permite abandonar el ambiente de Prolog para poder ejecutar comandos del sistema operativo. En caso de que no exista el archivo llamado "command.com", no permite dejar el ambiente y manda un mensaje de error el Sistema.

La opción 6, "Almacenar la Base de Datos", permite generar una copia de la base de datos que se encuentra en la memoria, creando para esto el archivo de respaldo con la extensión ".bak".

La opción 99 tiene por objetivo leer la base de datos a memoria y es llamada desde **consulta()**. En caso de que no encuentre el archivo con la base de datos, manda un error.

4.2 Consulta de la Base de Datos

Este es el módulo más importante de SISDOC por ser la interfaz con el usuario. A través de el se efectúa la consulta de la documentación de uno o varios Sistemas.

Los predicados que lo hacen son los siguientes:

```
proceso(1):-
     leeconsul(L),
     itera(L).

leeconsul(QUERY):-
     nl,nl,write("Consulta: "),
     readln(QUERY).

itera(STR):-
     STR >< "",
     nl,nl,
     scan(STR,List),
     filtra(List,List1),
     pars(List1,Q),
     findall(A,evalúa(Q,A),L),
     único(L,L1),
     imp_result(L1,STR),
     fail.

itera(STR):-
     STR >< "",
     leeconsul(L),
     itera(L).
```

La consulta es interpretada mediante el predicado **proceso(1)** el cual simplemente lee la consulta y llama a **itera()** que es el que realiza todo lo demás. El predicado **itera()** lleva a cabo el siguiente ciclo:

- Separa el texto en una lista de símbolos mediante el predicado **scan()**

- Elimina los símbolos desconocidos mediante el predicado **filtra()**

- Interpreta la consulta, determinando su tipo y los parámetros empleados mediante el predicado **pars()**

- Evalúa la consulta, reuniendo todas las soluciones en una lista, mediante **findall()**

- Elimina las soluciones duplicadas con **único()**

- Imprime los resultados y falla para permitir la iteración mediante la segunda instancia de **itera()**.

Esta simplemente repite el ciclo al ejecutar de nuevo los predicados de **proceso(1)**.

4.3 Lectura de la Consulta

La lectura de la consulta lo hace el predicado **leeconsul()** el cual termina cuando se le proporciona el caracter ESCAPE como respuesta. Como respuesta regresa el texto de la consulta en una cadena de caracteres.

4.4 Separación de símbolos

La separación del texto en una lista de símbolos se hace con los predicados **pega_car()** y **scan()**. El primero de los predicados esta definido como sigue:

```
pega_car(STR,Sym_ent,Sym_sal,STR1):-
     frontchar(STR,Char,Resto),
     str_char(Car,Char),
     isname(Car),
     concat(Sym_ent,Car,Nvosym),
     pega_car(Resto,Nvosym,Sym_sal,STR1),!.

pega_car(STR,Sym_ent,Sym_sal,STR1):-
     frontchar(STR,Char,Resto),
     str_char(Car,Char),
     Car<="9",
     Car>="0",
     concat(Sym_ent,Car,Nvosym),
     pega_car(Resto,Nvosym,Sym_sal,STR1),!.

pega_car(STR,Sym_ent,Sym_sal,STR1):-
     frontchar(STR,'*',Resto),
     concat(Sym_ent,"*",Nvosym),
     pega_car(Resto,Nvosym,Sym_sal,STR1),!.
...

pega_car(STR,Sym,Sym,STR):-!.
```

Pega_car() tiene por objetivo tomar caracter por caracter de la Cadena, concatenarlo al simbolo de Entrada y regresarlo como Salida junto con la cadena sobrante Resto. Esto lo hace sólo si el primer caracter es "pegable", es decir, es una letra, número o alguno de los siguientes caracteres: "*", ".", "-" y "&". En caso contrario ignora el caracter, que de seguro se trata de un caracter no "pegable", o copia el de entrada al de salida. El predicado trabaja recursivamente hasta que encuentra un caracter no pegable. **Pega_car()** es llamado por el predicado **scan()**.

Esta forma de construir los símbolos se debe al tipo de nombres de los archivos, programas y corridas que son

utilizados en el ambiente UNISYS. Estos están calificados en primer instancia por el nombre del proyecto, a continuación por el nombre del archivo y por ultimo el nombre del elemento. Cuando el nombre del proyecto aparece se separa del nombre del archivo por un asterisco. El nombre del archivo se separa del nombre del elemento por un punto. Por lo tanto tenemos que los nombres de los archivo pueden ser de la forma:

proyecto*archivo.

archivo.

Los nombres de los programas y las corridas son:

proyecto*archivo.elemento

archivo.elemento

Los identificadores proyecto, archivo y elemento están formados por cadenas de letras, números o por los caracteres "-" y "/".

El predicado **scan()** es el encargado de ir recolectando las palabras en una lista de los símbolos que regresa **pega_car()**; es recursivo y termina cuando la cadena de caracteres de entrada es vacía. El predicado se define como sigue:

```
scan("",[]).

scan(STR,[TOK|List]):-
     pega_car(STR,"",TOK,STR1),
scan(STR1,List),!.
```

4.5 Eliminación de símbolos desconocidos

La eliminación de los símbolos desconocidos para SISDOC se efectúa copiando de la lista de entrada a una lista de salida, sólo aquellos símbolos que son conocidos y eliminando los demás. Para llevar a cabo esto utiliza el predicado **filtra()** y **conocida()** los cuales son definidos como sigue:

```
filtra([],[]).
filtra([H|T],[X|L]):-
     sinónimos((H,X),!,
     filtra(T,L).
filtra([H|T],["&",Resto|L]):-
     frontchar(H,'&',Resto),!,
     filtra(T,L).
filtra([H|T],L):-
     not(conocida(H)),!,
     filtra(T,L).
filtra([H|T],[H|L]):-!,
```

```
        filtra(T,L).


    conocida(X):-
        nivel(_,AL),
        miembro(X,AL),!.
    conocida(X):-
        nivel(X,_),!.
    conocida(X):-
        sinónimos((X,_);sinónimos((_,X),!.
    conocida(X):-
        nom(_,X),!.
```

El predicado filtra sólo copia aquellos símbolos que:

- Son sinónimos de una palabra reservada o de un atributo

- Son del tipo &valor, que significa que no busque por sinónimos o los símbolos conocidos en SISDOC, posiblemente lo conoce pero verificarlo implicaría buscar en todos los valores de los objetos de la base de datos. Esto se hacía así originalmente pero causaba problemas en el tiempo de respuesta por lo cual se decidió dejar pasar este tipo de símbolos. Para reconocerlos se utilizó el símbolo "&".

- Son palabras conocidas: niveles, atributos, sinónimos o nombre de algún objeto de la base de datos.

El predicado **conocida()**, verifica si el símbolo es el nombre de un atributo, el nombre de un nivel, un sinónimo o el nombre de alguno de los objetos de SISDOC.

4.6 Interpretación de la consulta

Para interpretar la consulta solicitada se utiliza el predicado **pars()** el cual efectúa la verificación de la sintaxis y regresa el tipo de consulta con los parámetros respectivos. Esta debe apegarse a la sintaxis mostrada en la Figura 3.4. La verificación se hace aplicando regla por regla la sintaxis. En caso de sobrar alguna palabra o no concordar con ninguna de las posibles reglas, se indica con el predicado **error()** y con el mensaje adecuado. Los predicados implicados son:

```
    pars(List,Tposolic):-
        sintaxis(List,Tposolic,Resto),
        vefin(Resto),!.
    pars(List,_):-
        error(List),
```

```
        fail.

    error(List):-
        write(">> "),
        miembro(Y,List),
        not(conocida(Y)),!,
        write("\nDesconozco la palabra: ",Y),nl.

    error(_):-
        write("Lo siento, no la puedo reconocer <<").


    vefin([]).
```

La sintaxis del lenguaje mostrada en la Figura 3.4, es la misma que tiene el programa. El predicado empleado para la identificación de la sintaxis es **sintaxis()**, el cual recibe una lista con la consulta ya simplificada y la compara con las reglas que tiene. Verifica que cada uno de las palabras de la consulta cumplan con lo establecido, si se trata de un nivel, de un objeto, etc. Al primer éxito se detiene y regresa el tipo de consulta. En caso contrario simplemente falla.

Es importante notar que el control de la impresión simplemente crea una instancia del predicado **impresora()** el cual es utilizada en los predicados de impresión y llama de nuevo a la verificación de la sintaxis.

```
    sintaxis([imprime|List],Tposolic,Resto):-
        writedevice(printer),
        assert(impresora(si)),
        sintaxis(List,Tposolic,Resto),!.

    sintaxis([imprime|T],s_(imprime),T):-
        writedevice(screen),
        retract(impresora(si)),!.

    sintaxis([Nvl1,Nvl2,Nom2|Resto],s_een(Nvl1,Nvl2,Nom2),
    Resto):-
        nivel(Nvl1,_),
        nivel(Nvl2,_),
        nom(Nvl2,Nom2),!.

    sintaxis([Nvl1,Nom1,Nvl2|Resto],s_een(Nvl1,Nom1,Nvl2),
    Resto):-
        nivel(Nvl1,_),
        nivel(Nvl2,_),
        nom(Nvl1,Nom1),!.

    sintaxis([Nvl,no,descripción|Resto],s_ed(Nvl,no,
    descripción),Resto):-
        nivel(Nvl,_),!.
    ...
```

4.7 Evaluación de la consulta

Para evaluar la consulta se utiliza al predicado **evalúa()** junto con algunos otros que llevan a cabo las funciones del lenguaje comentadas en capítulos anteriores. La identificación de la consulta le pasa el tipo de consulta en una estructura (llamada solic) junto con los parámetros necesarios y ella encuentra las soluciones que deja en una lista. El predicado responsable es:

```
evalúa(s_een(Nvl1,Nvl2,Nom2),Nom1):-
     inter(Nvl1,Nom1,Nvl2,Nom2).
evalúa(s_een(Nvl2,Nvl1,Nom1),Nom2):-
     inter(Nvl1,Nom1,Nvl2,Nom2).

evalúa(s_een(Nvl1,Nom1,Nvl2),Nom2):-
     inter(Nvl1,Nom1,Nvl2,Nom2).
evalúa(s_een(Nvl2,Nom2,Nvl1),Nom1):-
     inter(Nvl1,Nom1,Nvl2,Nom2).

evalúa(s_ed(Nvl,no,descripción),Nom):-
     vedesc(Nvl,Nom,0).
evalúa(s_ed(Nvl,si,descripción),Nom):-
     vedesc(Nvl,Nom,1).
...
```

En estas instancias hay algunas que manejan el resultado en forma diferente a las demás. Cuando se trata de la descripción, la consistencia o la información de un objeto o la actualización de la base de datos (crear, modificar o eliminar objetos) no existe un símbolo como respuesta, sino todo un proceso que depende de que se haya solicitado. La forma de darse cuenta de esto es observando que el segundo parámetro del predicado **evalúa()** no regresa dato alguno.

Para cumplir con su objetivo el predicado **evalúa()** utiliza los siguientes predicados:

- Para encontrar las contenciones y dependencias utiliza el predicado **inter()**, el cual tiene por objeto ir intersectando por pares las jerarquías estructura-archivo, estructura-programa, etc. y regresando un valor cada vez. Genera todas las soluciones a través del "backtrack" del predicado **miembro()**. Tiene dos tipos de intersección, el primero es entre niveles contiguos (sistema-módulo,...) y el segundo entre niveles separados (sistema-corridas,...). En algunos casos utiliza el predicado **write()** para obligar al lenguaje a regresar.

```
inter(archivo,Arch,corrida,Corr):-
     corrida(Corr,_,_,Lista,_),
     miembro(Arch,Lista).
inter(corrida,Corr,módulo,Mod):-
     módulo(Mod,Lista,_),
     miembro(Corr,Lista).
inter(módulo,Mod,sistema,Sis):-
     sistema(Sis,Lista,_),
     miembro(Mod,Lista).
...
```

- Para determinar que objetos han sido o no documentados tenemos el predicado **vedesc()**. El número cero significa que aún no tienen descripción alguna, el uno es el caso contrario.

```
vedesc(estructura,Nom,0):-
     estruct(Nom,_,_,_,Descrip),Descrip="".
vedesc(estructura,Nom,1):-
     estruct(Nom,_,_,_,Descrip),Descrip<>"".
vedesc(archivo,Nom,0):-
     archivo(Nom,_,Descrip),Descrip="".
vedesc(archivo,Nom,1):-
     archivo(Nom,_,Descrip),Descrip<>"".
...
```

- Para obtener los valores que tiene un atributo de un objeto en particular, o que objetos tienen un predicado dado, se utiliza el predicado **atrib()**, el cual tiene una instancia por cada atributo de cada nivel.

```
atrib(corrida,Nom,subcorrida,Valor):-
     corrida(Nom,Lista,_,_,_),
     miembro(Valor,Lista).
atrib(corrida,Nom,subprograma,Valor):-
     corrida(Nom,_,Lista,_,_),
     miembro(Valor,Lista).
atrib(corrida,Nom,programa,Valor):-
     corrida(Nom,_,Lista,_,_),
     miembro(Valor,Lista).
...
```

- Para verificar la consistencia del nivel proporcionado, se utiliza el predicado **consist()**, el cual revisa todos los valores de sus atributos multivaluados viajando por la jerarquía. **Consist()** utiliza el predicado **subconsist()**, quien verifica cada valor del atributo multivaluado.

```
consist(sistema,[]).
consist(sistema,Nom):-
     sistema(Nom,Listmod,_),!,
     write(" "),
     subconsist(módulo,Listmod).
```

```
consist(sistema,Nom):-
     write("  No existe el sistema ",Nom),nl,!.
...


subconsist(Nvl,Lista):-
     miembro(Nom,Lista),
     write("Verificando el(la) ",Nvl," >>
     ",Nom),nl,
     consist(Nvl,Nom).
```

- Para desplegar los datos de un objeto de la base de datos tenemos al predicado **danvl()**, el cual muestra el nombre de los atributos y sus valores.

```
danvl(estructura,Nom):-
     estruct(Nom,Org,Dic,Long,Descrip),!,
     write("El nombre de la estructura es: ",Nom),nl,
     write("\nSu organización es: ",Org),nl,
     write("\nSu diccionario de datos está en el
     archivo",Dic," y es: "),nl,
     impestr(Dic,Texto1),
     write(Texto1),
     write("\nSu longitud de registro es: ",Long),nl,
     write("\nSu descripción está en el archivo"
     ,Descrip," y es: "),nl,
     nl.
...
```

- Para crear nuevos objetos en la base de datos, solicitando para ello los valores de sus atributos, tenemos el predicado **agreganvl()**. Este simplemente solicita dato por dato, si ya existe el objeto a dar de alta, simplemente falla.

```
agreganvl(estructura):-
     write("\n PROPORCIONA LOS DATOS DE LA NUEVA
     ESTRUCTURA:\n"),
     write("dame el nombre de la estructura\n"),
     readln(Nombre), Nombre<>"",
     not(estruct(Nombre,_,_,_,_)),
     write("dame el tipo de organizacion\n"),
     editsym("",Organiz,"Organizacion",3,50,5,25),
     write("nombre del docto para el direc. de datos:
     ",Nombre),nl,
     concat(Nombre,".dic",Dic),
     editstr(Dic,Dicect,"Nombre",3,50,5,30),
     editestr(Dicect),
     write("dame la longitud del registro\n"),
     editsym("",Longreg,"Longitud",3,50,5,15),
     write("nombre del docto para la descripción de la
     estructura: ",Nombre),nl,
```

```
        concat(Nombre,".est",Desc),
        editstr(Desc,Descrito,"Nombre",3,50,5,30),
        editestr(Descrito),
        assert(estruct(Nombre,Organiz,Dicect,Longreg,
        Descrito)),
        agreganvl(estructura),!.
    agreganvl(estructura).
    ...
```

- Para modificar y eliminar objetos de la base de datos respectivamente, tenemos a **modificanvl()** y **eliminanvl()**.

```
    modificanvl(estructura) :-
        write("\n PROPORCIONA LOS NUEVOS DATOS DE LA
        ESTRUCTURA:\n"),
        write("Dame el nombre de la estructura o ESPACIOS
        para terminar\n"),
        readln(Nombre), Nombre<>"",
        estruct(Nombre,Organiz,Dic,Longreg,Descrito),
        write("Modifica el tipo de organizacion\n"),
        editsym(Organiz,Norganiz,"Organizacion",3,50,5,25),
        write("Modifica su dicc. de datos: ",Dic), nl,
        editestr(Dic),
        write("dame la longitud del registro\n"),
        editsym(Longreg,Nlongreg,"Longitud",3,50,5,15),
        write("Modifica el nombre del docto de su
        descripción\n"),
        editstr(Descrito,NDescrito,"Nombre",3,50,5,30),
        write("Modifica su descripción en: ",NDescrito),nl,
        editestr(NDescrito),
        retract(estruct(Nombre,Organiz,Dic,Longreg,
        Descrito)),
        assert(estruct(Nombre,NOrganiz,Dic,NLongreg,
        NDescrito)),
        modificanvl(estructura),!.
    modificanvl(estructura).
    ...

    eliminanvl(estructura):-
        write("\nDAME EL NOMBRE DE LA ESTRUCTURA A
        ELIMINAR:\n"),
        wreadln(Nom),Nom<>"",
        estruct(Nom,Org,Dic,Lon,Des),
        retract(estruct(Nom,Org,Dic,Lon,Des)),!.
    ...
```

En estos predicados es importante resaltar, que para los diferentes tipos de atributos y valores, fueron desarrollados predicados especiales que los manejan. Estos son **editsym()** y **editint()**, **editstr()** y **editlist()** para el manejo de los valores directos, indirectos y los atributos multivaluados respectivamente. Estos predicados están definidos como:

- Con el predicado **editsym()** se puede modificar un símbolo dentro del editor de Turbo Prolog. Se crea una ventana de edición del tamaño que se le indique en sus parámetros. Cuando se digita la tecla ESCAPE indica que elimine los cambio, con la F10 indica que los acepte.

```
editsym(Nom,Nvonom,Letrero,R,C,Nren,Ncol):-
     makewindow(15,112,7,Letrero,R,C,Nren,Ncol),
     editmsg(Nom,Nvonom,"F10 o ESC","","",1,"",Diof10),
     Diof10=0,
     removewindow.
editsym(Nom,Nom,_,_,_,_,_):-
     removewindow.
```

- Con el predicado **editsrtr()**, al igual que con **editlist()** y **editint()**, se editan cadenas de caracteres como el nombre de los archivos, listas de símbolos y números respectivamente, con la misma filosofía del editor de texto que se usa en **editsym()**.

```
editstr(Nom,Nvonom,Letrero,R,C,Nren,Ncol):-
     makewindow(15,112,7,Letrero,R,C,Nren,Ncol),
     editmsg(Nom,Nvonom,"F10 o ESC","","",1,"",Diof10),
     Diof10=0,
     removewindow.
editstr(Nom,Nom,_,_,_,_,_):-
     removewindow.

editlist(List,Nvalist,Letrero,R,C,Nren,Ncol):-
     makewindow(15,112,7,Letrero,R,C,Nren,Ncol),
     descan(List,Texto),
     editmsg(Texto,Nvotexto,"F10 o ESC","","",
     1,"",Diof10),
     Diof10=0,
     scan(Nvotexto,Nvalist),
     removewindow.
editlist(List,List,_,_,_,_,_):-
     removewindow.

editint(Num,Nvonum,Letrero,R,C,Nren,Ncol):-
     makewindow(15,112,7,Letrero,R,C,Nren,Ncol),
     str_int(Texto,Num),
     editmsg(Texto,Nvotexto,"F10 o ESC","","",
     1,"",Diof10),
     Diof10=0,
     str_int(Nvotexto,Nvonum),
     removewindow.
editint(Num,Num,_,_,_,_,_):-
     removewindow.
```

El predicado **descan()** a continuación, permite convertir una lista de símbolos en una cadena de caracteres para que puedan ser manipulados a través del editor de texto de Prolog. El caracter salto de renglón ("\n") sirve para separar los simbolos en el editor.

```
descan([Tok|List],String):-
     descan(List,Nstring),
     concat(Tok,"\n",Temp),
     concat(Temp,Nstring,String).
descan([],"").
```

El predicado que sigue a continuación es similar a los anteriores en filosofía. **Editestr()** se utiliza para modificar los atributos de tipo texto, es decir los documentos. Crea una copia del texto editado y lo deja en el mismo archivo pero con extensión "bak".

```
editestr(Nom) :-
     existfile(Nom),
     openread(archivo, Nom),
     file_str(Nom, Ent),
     editmsg(Ent, Sal,"EDITANDO",Nom,"F10 para terminar,
     ESC para abortar", 1, "ayuda.hlp", Diof10),
     clearwindow,
     closefile(archivo),
     Diof10=0,
     fronttoken(Nom, Nvonom,_),
     concat(Nvonom, ".bak",Nombak),
     renamefile(Nom,Nombak),
     openwrite(archivo,Nom),
     writedevice(archivo),
     write(Sal),
     writedevice(screen),
     closefile(archivo),!.
     editestr(Nom) :-
     not(existfile(Nom)),
     editmsg("", Sal,"CREANDO",Nom, "F10 para terminar,
     ESC para abortar", 1, "ayuda.hlp", Diof10),
     clearwindow,
     Diof10=0,
     openwrite(archivo,Nom),
     writedevice(archivo),
     write(Sal),
     writedevice(screen),
     closefile(archivo),!.

editestr(_):-
     clearwindow.
```

Este predicado da la facilidad de que cuando no se conocen los comandos del editor de texto, se pueden consultar en la ayuda en línea con la tecla F1. Esta información está contenida en el archivo "**ayuda.hlp**"

4.8 Eliminación de soluciones duplicadas

En vista de que Prolog encuentra todas las posibles soluciones a una consulta, estas pueden repetirse, por lo tanto es necesario eliminar los duplicados mediante el predicado **único()**, el cual sólo copia a una nueva lista los símbolos que no existan ya. Se define como sigue:

```
único([],[]).
único([H|T],L):-
     miembro(H,T),!,
     único(T,L).
único([H|T],[H|L]):-
único(T,L).
```

4.9 Impresión de resultados

Como los resultados de las consultas son simples símbolos (excepto cuando se manejan los archivos de texto) siempre son impresos a cuatro columnas en el video o en la impresora. El predicado encargado de esto es **imp_result()**:

```
imp_result(L1,_):-
     not(impresora(si)),
     write_list(0,L1),!.

imp_result(L1,STR):-
     write("\n\n\n",STR,"\n"),
     write_list(0,L1),
     flush(printer),
     writedevice(screen),
     retract(impresora(si)).
```

4.10 Consulta del lenguaje

Esta opción permite consultar los elementos del lenguaje a través de un menú. Los elementos son las palabras reservadas y los sinónimos. Los predicados encargados son **veleng()**, el cual permite mostrar el menú de opciones y ejecutar al predicado escogido.

```
veleng:-
     repite,
     menú(5,40,"Lenguaje",
     ["Palabras Reservadas",
      "Sinónimos de P. Reservadas",
      "Sinónimos de Atributos"
     ],Selección),
     nl,veleng1(Selección),Selección=0,!.
```

Los predicados que se presentan a continuación son los que controlan cada opción:

```
velengl(0).

velengl(1):-
     write("Niveles\n********\n"),
     findall(X,sinonnvl(_,X),L),
     único(L,L1),
     write_list(0,L1),nl.

velengl(1):-
      write("\n\nDigita RETURN para continuar"),
     readchar(_).

velengl(2):-
     writef("%-15 %-15\n","Sinónimo","P. Reservada"),
     write("*************** ***************\n"),
     sinonnvl(E,S),
     writef("%-15 %-15\n",E,S),
     fail.

velengl(2):-
     write("\n\nDigita RETURN para continuar"),
     readchar(_).

velengl(3):-
     writef("%-15 %-15\n","Sinónimo","Atributo"),
     write("*************** ***************\n"),
     sinonatrib(E,S),
     writef("%-15 %-15\n",E,S),
     fail.

velengl(3):-
     write("\n\nDigita RETURN para continuar"),
     readchar(_).
```

Veleng(0) es el que termina con la ejecución del Menú. **Veleng(1)** el que encuentra todos los niveles de la jerarquía y los lista. **Veleng(2)** lista las parejas de sinónimo-palabra reservada que tiene en la base de datos. Por último, **veleng(3)** hace los mismo que la opción anterior pero ahora con las parejas sinónimos-atributos.

4.11 Actualización del Lenguaje

Esta opción permite, mediante un menú, crear nuevos sinónimos para las palabras reservadas y atributos. En vista de que eliminar un atributo es una actividad poco utilizada no hay predicado que se encargue de eso. Los predicados encargados de la actualización son:

```
actleng:-
     repite,
     menú(5,40,"Actualiza el Lenguaje",
       ["Nuevos sinónimos para P. Reserv.",
        "Nuevos sinónimos para Atributo"
       ],CHOICE),
     nl,actlengl(CHOICE),CHOICE=0,!.
```

Con estos predicados se controla el menú que permite agregar nuevos sinónimos al lenguaje, tanto para las palabras reservadas como para los atributos.

```
actlengl(0).

actlengl(1):-
     nvosinnvl.
actlengl(2):-
     nvosinatrib.

nvosinnvl:-
     dasinónimo(E),
     write("Sinónimo: "),
     readln(SYNONYM),
     SYNONYM><"",
     assert(Sinónimo(SYNONYM,E))
     ,nvosinnvl.

nvosinatrib:-
     daatrib(E),
     write("Sinónimo: "),
     readln(SYNONYM),
     SYNONYM><"",
     assert(Sinónimo(SYNONYM,E)),
     nvosinatrib.
```

Por último, los predicados a continuación son los encargados de solicitar la palabra reservada o el nombre del atributo al cual le asignarán un sinónimo. Utiliza para esto un menú con las posibles palabras reservadas y los atributos. Localiza la palabra seleccionada con **índice()** y la regresa.

```
dasinónimo(E):-
     findall(X,sinonnvl(_,X),L),
     único(L,L1),
     menú(2,49,"P. Reservada",L1,C),
     índice(L1,C,E).

daatrib(E):-
     findall(X,daatrib1(X),L),
     único(L,L1),
     menú(2,49,"Atributo",L1,C),
     índice(L1,C,E).
```

```
daatribl(Atrib):-
     nivel(_,List),
     miembro(Atrib,List).
```

5 Bibliografía del Capítulo

[1] Cristina Loyo
Prolog en términos imperativos
Memorias de la IV Reunión Nacional de Inteligencia Artificial, Puebla
Feb. 1987, LIMUSA.

[2] Cristina Loyo
Fundamentos de Prolog, Notas de Clase
Facultad de Ciencias, UNAM
Abr. 1987

[3] Christopher John Hogger
Introduction to Logic Programming
Academic Press
1984

[4] W. F. Clocksin, C. S. Mellish
Programming in Prolog
Springer-Verlag
1984

[5] Gregory L. Lazarev
WHY PROLOG?
Justifying Logic Programming for Practical Applications
Prentice Hall
1989

[6] J. W. Lloyds
Foundations of Logic Programming
Springer-Verlag
2nd extended edition, 1987

[7] Michael Covington
Programming in Logic, Part 1
Tech Journal
Dic. 1985
Pag 83-95

[8] Michael Covington, Andre Vellino
Prolog Arrives
Tech Journal
Nov. 1986
Pag 53-69

[9] Namir Clement Shammas
Turbo Prolog

Byte
Sep. 1986
Pag 293-295, 335-340

[10] Owner's Handbook
Turbo Prolog
Borland International, Inc.
1986

[11] Foxbase
User's Manual
Fox Software, Inc.
May. 1988

[12] Christopher D.S. Moss
Intelligent Databases
Byte, enero 1987
Pag. 97-106

ESTA TESIS NO DEBE SALIR DE LA BIBLIOTECA

CAPITULO 4

CONCLUSIONES

1 Conclusiones

Después de haber desarrollado SISDOC, y de haber consultado una amplia bibliografía al respecto, debemos reconocer que "No hay herramienta suficiente, por sofisticada que sea, que permita documentar Sistemas, cuando los que los idean, desarrollan o mantienen, no consideran la documentación como parte integral de los Sistemas". Sin embargo, creemos que una herramienta como la que hemos creado puede ayudar en buena medida a convencer al usuario de las bondades de una documentación.

Como conclusiones del desarrollo de esta herramienta hablaremos de tres aspectos:

1) SISDOC como herramienta

2) SISDOC como parte del proyecto

3) SISDOC como fuente de nuevas técnicas

1.1 Como herramienta

- SISDOC cumplió con todo lo esperado y comentado en el capítulo 2. Mostró que su carga inicial es realmente costosa para Sistemas ya desarrollados por el problema mismo de la documentación. Sin embargo, como su carga requiere de una síntesis de los diferentes elementos que forman el o los Sistemas, proporciona una guía para crear o actualizar y mantener la documentación de los mismos.

- Permite al novato conocer los Sistemas viajando a través de su documentación mediante preguntas, que de otra manera sería una actividad prolongada y costosa.

- Permite al personal de mantenimiento una consulta exacta y certera sobre el Sistema.

1.2 Como parte del proyecto

- Se observó que la documentación de un Sistema y la consulta de la misma pueden ser manejadas con un modelo que tenga como mínimo las funciones que tiene SISDOC. Sin embargo, es necesario dotar a SISDIC con facilidades extras que faciliten el manejo de otras características

de la documentación como las gráficas, las versiones, etc.

- La decisión de usar Turbo Prolog fue acertada para una primera versión, sin embargo, para futuros desarrollos se recomendaría sustituirlo por otro Prolog más versátil y poderoso.

1.3 Como fuente de nuevas técnicas

- El desarrollo de SISDOC permitió vislumbrar nuevas técnicas que pueden ser aplicadas en la programación de Sistemas.

2 Mejoras

Conforme SISDOC fue utilizado, ya sea durante la carga de la documentación o durante la consulta de la misma, se observó que aunque el número y el tipo de preguntas a las que responde es aceptable, es necesario realizar algunas mejoras. Para hablar de ellas las dividimos en tres aspectos:

1) Mejoras al modelo.

2) Mejoras al manejador

3) Mejoras a la implantación

2.1 Mejoras al modelo

Creemos que sería conveniente:

- Desarrollar una nueva versión de SISDOC con un modelo dinámico, es decir, que permita crear y modificar en forma dinámica el número de niveles y de atributos y por lo tanto, reestructurar la base de datos agregando nuevos objetos o valores para ellos.

- Crear relaciones más generales que permitan definir el tipo de atributo que uno desee, como listas de listas, predicados u otras cosas.

2.2 Mejoras al manejador

En este aspecto hay varias ideas que surgieron como:

- Desarrollar un manejador más sofisticado que permita como mínimo, las facilidades proporcionadas por una base de datos relacional. En particular que permita tener acceso a los objetos que está describiendo cuando esto

sea posible, es decir, mostrar directamente los archivos, bases de datos, fuentes, etc. utilizados en la operación.

- Permitir las búsquedas dentro de un texto a través de varios archivos de texto.

- Dotarlo de facilidades que permitan: la diagramación, el control de versiones, la definición de diferentes objetos y por último, de una interfaz gráfica con los usuarios.

3.3 Mejoras a la implantación

A lo largo de la programación se detectaron varios aspectos que mejorar como:

- Utilizar memoria virtual que permita evitar los problemas de saturación de memoria, al manejar bases de datos que contengan gran cantidad de niveles y/o atributos.

- Cambiar la versión del compilador de Prolog, para resolver las limitaciones de Turbo Prolog que son mencionadas a continuación:

  . Utilización de tipos

  . Limitaciones de memoria

  . Colector de basura ineficiente

  . Con algunas fallas dentro del compilador.

Se requiere que el compilador a seleccionar ofrezca:

  . Facilidades para leer y escribir clausulas en la base de datos

  . Un mayor número de predicados interconstruidos

  . Manejo de predicados de segundo orden

  . Manejo de memoria extendida o expandida.

- Eliminar el lenguaje pseudo natural el cual consume gran parte del tamaño de SISDOC sustituyéndolo por alguna interfaz más simple como menús.

En resumen podemos decir que SISDOC cumplió con los objetivos planteados y que, además de los beneficios que aporta, podemos mencionar que, en nuestro caso, ha abierto la puerta

de la aplicación de la Inteligencia Artificial en el desarrollo y mantenimiento de Sistemas.

ANEXO A

LISTADO FUENTE DE SISDOC

```
/* SISDOC.PRO  */
code=4600,trail=120,check_cmplo,diagnostics
DOMAINS
nombre, tipo, organiz, cveproy = symbol
texto = string
long, decim, longreg, valor = integer
listsym, LIST = symbol*
listtxt = string*
listint = integer*
file = archivo
solic =
     s_een(symbol,symbol,symbol);
     s_cen(symbol,symbol,symbol);
     s_en(symbol,symbol);
     s_e(symbol);
     s_ce(symbol);
     s_c(symbol,symbol);
     s_(symbol);
     s_ecv(symbol,symbol,symbol);
     s_ed(symbol,symbol,symbol);
     s_agreganvl(symbol,symbol)

/*definición de la base de datos*/
DATABASE

impresora(symbol)
sinónimos(symbol,symbol)

/*  estruct(nombre, organiz, diccionario-datos, longreg, descrito) */
     estruct(symbol, symbol, string,  symbol,  string)

/*  archivo(nombre, estructura, descrito) */
     archivo(symbol, symbol, string)

/*  programa(nombre, listestr, listprog, list-datos-ent, descrito)  */
     programa(symbol, listsym, listsym, listsym, string)

/*  corrida(nombre, listcorr, listprog, listarch, descrito)*/
     corrida(symbol, listsym, listsym, listsym, string)

/*  módulo(nombre, listcorr, descrito)  */
     módulo(symbol, listsym, string)

/*  sistema(nombre, listmod, descrito)  */
     sistema(symbol, listsym, string)


/*  EL MODELO DE SISDOC */
PREDICATES
nivel(symbol,listsym)
```

```
nom(symbol,symbol)
CLAUSES
nivel("estructura",["organizacion","diccionario","longitud","descripción"]).
nivel("archivo",["estructura","descripción"]).
nivel("programa",["estructura","programa","dato","descripción"]).
nivel("corrida",["corrida","programa","archivo","descripción"]).
nivel("módulo",["corrida","descripción"]).
nivel("sistema",["módulo","descripción"]).

/*predicado que proporciona los nombres de los objetos*/
nom(estructura,Nom):- estruct(Nom,_,_,_,_).
nom(archivo,Nom):- archivo(Nom,_,_).
nom(programa,Nom):- programa(Nom,_,_,_,_).
nom(corrida,Nom):-  corrida(Nom,_,_,_,_).
nom(módulo,Nom):-   módulo(Nom,_,_).
nom(sistema,Nom):-  sistema(Nom,_,_).


/*-----------------------------------------------
                    RUTINAS ACCESORIAS
   ----------------------------------------------- */

PREDICATES

/* Manejo de Listas */
miembro(symbol,listsym)
maxlen(listsym,INTEGER,INTEGER)
listlen(listsym,INTEGER)
único(listsym,listsym)
repite
indice(listsym,INTEGER,SYMBOL)

CLAUSES

miembro(H,[H|_]).
miembro(H,[_|T]):-
     miembro(H,T).

indice([X|_],1,X):- !.
indice([_|L],N,X):-
     N>1,N1=N-1,indice(L,N1,X).

único([],[]).
único([H|T],L):-
     miembro(H,T),!,
     único(T,L).
único([H|T],[H|L]):-
     único(T,L).

maxlen([H|T],MAX,MAX1):-
     str_len(H,LEN),     LEN>MAX,!,
     maxlen(T,LEN,MAX1).
maxlen([_|T],MAX,MAX1):-maxlen(T,MAX,MAX1).
```

```
maxlen([],LEN,LEN).

listlen([],0).
listlen([_|T],N):-
     listlen(T,X),
      N=X+1.

     repite. repite:-repite.

/* ---------------------------------
          IMPRESION DE RESULTADOS
    --------------------------------- */

PREDICATES
impestr(string,symbol)
writelist(INTEGER,INTEGER,listsym)
write_list(INTEGER,listsym)
imp_result(listsym,symbol)

CLAUSES
impestr(Descrip,Valor):-
     existfile(Descrip),
     openread(archivo,  Descrip),
     file_str(Descrip,Valor),
     closefile(archivo).

writelist(_,_,[]).
writelist(L1,ANTKOL,[H|T]):-
     field_str(L1,0,ANTKOL,H),
     L11=L1+1,writelist(L11,ANTKOL,T).

write_list(_,[]).
write_list(_,[X]):-!,write(X).
write_list(4,[H|T]):-!,write(H),nl,write_list(0,T).
write_list(3,[H|T]):-
       str_len(H,LEN),LEN>13,!,write(H),nl,write_list(0,T).
write_list(N,[H|T]):-
       str_len(H,LEN),LEN>13,!,N1=N+2,writef("%-27",H),write_list(N1,T).
write_list(N,[H|T]):-
       N1=N+1,writef("%-13",H),write_list(N1,T).

imp_result(L1,_):-
     not(impresora(si)),
     write_list(0,L1),!.

imp_result(L1,STR):-
     write("\n\n\n\n",STR,"\n"),
     write_list(0,L1),
     flush(printer),
     writedevice(screen),
     retract(impresora(si)).

/*------------
      SCANNER
```

```
   ------------ */

PREDICATES

scan(SYMBOL,listsym)
pega_car(SYMBOL,symbol,symbol,SYMBOL)
filtra(listsym,listsym)
conocida(SYMBOL)

CLAUSES
pega_car(STR,Sym_ent,Sym_sal,STR1):-
      frontchar(STR,Char,Resto),
      str_char(Car,Char),
      isname(Car),
      concat(Sym_ent,Car,Nvosym),
      pega_car(Resto,Nvosym,Sym_sal,STR1),!.

pega_car(STR,Sym_ent,Sym_sal,STR1):-
      frontchar(STR,Char,Resto),
      str_char(Car,Char),
      Car<="9",
      Car>="0",
      concat(Sym_ent,Car,Nvosym),
      pega_car(Resto,Nvosym,Sym_sal,STR1),!.

pega_car(STR,Sym_ent,Sym_sal,STR1):-
      frontchar(STR,'*',Resto),
      concat(Sym_ent,"*",Nvosym),
      pega_car(Resto,Nvosym,Sym_sal,STR1),!.

pega_car(STR,Sym_ent,Sym_sal,STR1):-
      frontchar(STR,'.',Resto),
      concat(Sym_ent,".",Nvosym),
      pega_car(Resto,Nvosym,Sym_sal,STR1),!.

pega_car(STR,Sym_ent,Sym_sal,STR1):-
      frontchar(STR,'-',Resto),
      concat(Sym_ent,"-",Nvosym),
      pega_car(Resto,Nvosym,Sym_sal,STR1),!.

/* permite leer valores precedidos por signo & para acelerar el filtrado
sin buscar en todos los valores de las atributos */
pega_car(STR,Sym_ent,Sym_sal,STR1):-
      frontchar(STR,'&',Resto),
      Sym_ent = "",
      pega_car(Resto,"&",Sym_sal,STR1),!.

pega_car(STR,Sym_ent,Sym_sal,STR1):-
      Sym_ent = "",
      frontchar(STR,_,Resto),
      pega_car(Resto,Sym_ent,Sym_sal,STR1),!.

pega_car(STR,Sym,Sym,STR):-!.
```

```
scan("",[]).

scan(STR,[TOK|List]):-
      pega_car(STR,"",TOK,STR1),
      scan(STR1,List),!.

conocida(X):-nivel(_,AL),miembro(X,AL),!.
conocida(X):-nivel(X,_),!.
conocida(X):-sinónimos(X,_);sinónimos(_,X),!.
conocida(X):-nom(_,X),!.


filtra([],[]).
filtra([H|T],[X|L]):-
      sinónimos(H,X),!,
      filtra(T,L).
/* si se trata de un valor de atributo (&valor) no lo busques aceptalo */
filtra([H|T],["&",Resto|L]):-
      frontchar(H,'&',Resto),!,
      filtra(T,L).
filtra([H|T],L):-
      not(conocida(H)),!,
      filtra(T,L).
filtra([H|T],[H|L]):-
      !,filtra(T,L).

/*-------------------------------------------------------
READING THE KEYBORD - Not all of the keys defined here are used
by the program. It has been generalized to show what keyboard
routines one might implement, and how to go about it.
-------------------------------------------------------*/

DOMAINS
   ROW,COL,LEN = INTEGER

   KEY = cr ; esc ; break ; tab ; btab ; del ; bdel ; ins ;
       end ; home ; ftast(INTEGER) ; up ; down ; left ; right ;
       ctrlleft; ctrlright; ctrlend; ctrlhome; pgup; pgdn;
       chr(CHAR) ; otherspec

PREDICATES
   readkey(KEY)
   readkey1(KEY,CHAR,INTEGER)
   readkey2(KEY,INTEGER)
CLAUSES
readkey(KEY):-
readchar(T),char_int(T,VAL),readkey1(KEY,T,VAL).

readkey1(KEY,_,0):-
!,readchar(T),char_int(T,VAL),readkey2(KEY,VAL).
readkey1(cr,_,13):-!.
readkey1(esc,_,27):-!.
readkey1(chr(T),T,_) .
```

```
readkey2(up,72):-!.
readkey2(down,80):-!.
readkey2(ftast(N),VAL):-VAL>58,VAL<70,N=VAL-58,!.
readkey2(otherspec,_).


/* -------------------------
     POPUP MENU SYSTEM
 -------------------------
cr and F10: activa la opción
Esc:          aborta              */

PREDICATES
menú(ROW,COL,STRING,listsym,INTEGER)
menú1(ROW,listsym,ROW,INTEGER,INTEGER)
menú2(ROW,listsym,ROW,INTEGER,INTEGER,KEY)

CLAUSES
menú(L1,KOL,TXT,List,CHOICE):-
      shiftwindow(21),
      maxlen(List,0,ANTKOL),
      listlen(List,LEN),ANTL1=LEN,LEN>0,
      HH1=ANTL1+2,HH2=ANTKOL+2,
      makewindow(3,7,7,TXT,L1,KOL,HH1,HH2),
      HH3=ANTKOL,
      writelist(0,HH3,List),cursor(0,0),
      menú1(0,List,ANTL1,ANTKOL,CH),
      CHOICE=1+CH,
      removewindow,
      shiftwindow(22),
      shiftwindow(2).

menú1(L1,List,MAXL1,ANTKOL,CHOICE):-
      field_attr(L1,0,ANTKOL,112),
      cursor(L1,0),
      readkey(KEY),
      menú2(L1,List,MAXL1,ANTKOL,CHOICE,KEY).

menú2(_,_,_,_,-1,esc):-!.
menú2(L1,_,_,_,CH,ftast(10)):-!,CH=L1.
menú2(L1,_,_,_,CH,cr):-!,CH=L1.
menú2(L1,List,MAXL1,ANTKOL,CHOICE,up):-
      L1>0,!,      field_attr(L1,0,ANTKOL,7),
      L11=L1-1,
      menú1(L11,List,MAXL1,ANTKOL,CHOICE).

menú2(L1,List,MAXL1,ANTKOL,CHOICE,down):-
      L1<MAXL1-1,!,
      field_attr(L1,0,ANTKOL,7),
      L11=L1+1,
      menú1(L11,List,MAXL1,ANTKOL,CHOICE).

menú2(L1,List,MAXL1,ANTKOL,CHOICE,_):-
      menú1(L1,List,MAXL1,ANTKOL,CHOICE).
```

```
/* ------------------------
          EDICION
    ------------------------ */

PREDICATES
editsym(symbol,symbol,string,integer,integer,integer,integer)
editstr(string,string,string,integer,integer,integer,integer)
editlist(listsym,listsym,string,integer,integer,integer,integer)
editint(integer,integer,string,integer,integer,integer,integer)
descan(listsym,symbol)
editestr(string)

CLAUSES

descan([Tok|List],String):-
      descan(List,Nstring),
      concat(Tok,"\n",Temp),
      concat(Temp,Nstring,String).
descan([],"").

editsym(Nom,Nvonom,Letrero,R,C,Nren,Ncol):-
      makewindow(15,112,7,Letrero,R,C,Nren,Ncol),
      editmsg(Nom,Nvonom,"F10 o ESC","","",1,"",Diof10),
      Diof10=0,
      removewindow.
editsym(Nom,Nom,_,_,_,_,_,_):-
      removewindow.

editstr(Nom,Nvonom,Letrero,R,C,Nren,Ncol):-
      makewindow(15,112,7,Letrero,R,C,Nren,Ncol),
      editmsg(Nom,Nvonom,"F10 o ESC","","",1,"",Diof10),
      Diof10=0,
      removewindow.
editstr(Nom,Nom,_,_,_,_,_,_):-
      removewindow.

editlist(List,Nvalist,Letrero,R,C,Nren,Ncol):-
      makewindow(15,112,7,Letrero,R,C,Nren,Ncol),
      descan(List,Texto),
      editmsg(Texto,Nvotexto,"F10 o ESC","","",1,"",Diof10),
      Diof10=0,
      scan(Nvotexto,Nvalist),
      removewindow.
editlist(List,List,_,_,_,_,_,_):-
      removewindow.

editint(Num,Nvonum,Letrero,R,C,Nren,Ncol):-
      makewindow(15,112,7,Letrero,R,C,Nren,Ncol),
      str_int(Texto,Num),
      editmsg(Texto,Nvotexto,"F10 o ESC","","",1,"",Diof10),
      Diof10=0,
      str_int(Nvotexto,Nvonum),
```

```
      removewindow.
editint(Num,Num,_,_,_,_,_):-
      removewindow.

editestr(Nom):-
      existfile(Nom) ,
      openread(archivo, Nom) ,
      file_str(Nom, Ent),
      editmsg(Ent, Sal,"EDITANDO",Nom, "F10 para terminar, ESC para abortar", 1,
      "ayuda.hlp", Diof10),
      clearwindow,
      closefile(archivo),
      Diof10=0,
      fronttoken(Nom, Nvonom,_),
      concat(Nvonom, ".bak",Nombak),
      renamefile(Nom,Nombak),
      openwrite(archivo,Nom),
      writedevice(archivo),
      write(Sal),
      writedevice(screen),
      closefile(archivo),!.
editestr(Nom):-
      not(existfile(Nom)),
      editmsg("", Sal,"CREANDO",Nom, "F10 para terminar, ESC para abortar", 1,
      "ayuda.hlp", Diof10),
      clearwindow,
      Diof10=0,
      openwrite(archivo,Nom),
      writedevice(archivo),
      write(Sal),
      writedevice(screen),
      closefile(archivo),!.
editestr(_):-clearwindow.

/* ------------
      PARSER
   ------------ */

PREDICATES
pars(Listsym,solic)
sintaxis(Listsym,solic,Listsym)
error(listsym)
vefin(listsym)

CLAUSES

pars(List,Tposolic):-
      sintaxis(List,Tposolic,Resto),
      vefin(Resto),!.
pars(List,_):-
      error(List),fail.

error(List):-
      write(">> "),miembro(Y,List),
```

```
    not(conocida(Y)),!,
    write("\nDesconozco la palabra: ",Y),nl.
error(_):-
    write("Lo siento, no la puedo reconocer <<").

    vefin([]).

/*si quiere listar prende la imp. si falla la sintaxis apaga la imp.*/
sintaxis([imprime|List],Tposolic,Resto):-
    writedevice(printer),
    assert(impresora(si)),
    sintaxis(List,Tposolic,Resto),!.

sintaxis([imprime|T],s_(imprime),T):-
    writedevice(screen),
    retract(impresora(si)),!.

/*que nvl1 tiene el nvl2 nom o el nvl2 nom que nvl1 tiene*/
sintaxis([Nvl1,Nvl2,Nom2|Resto],s_een(Nvl1,Nvl2,Nom2),Resto):-
    nivel(Nvl1,_),
    nivel(Nvl2,_),
    nom(Nvl2,Nom2),!.

/*el nvl1 nom en que nvl2 está o en que nvl2 está el en1 nom*/
sintaxis([Nvl1,Nom1,Nvl2|Resto],s_een(Nvl1,Nom1,Nvl2),Resto):-
    nivel(Nvl1,_),
    nivel(Nvl2,_),
    nom(Nvl1,Nom1),!.

/* que niveles no tienen descripción */
sintaxis([Nvl,no,descripción|Resto],s_ed(Nvl,no,descripción),Resto):-
    nivel(Nvl,_),!.

/* que niveles si tienen descripción */
sintaxis([Nvl,si,descripción|Resto],s_ed(Nvl,si,descripción),Resto):-
    nivel(Nvl,_),!.

/* que niveles tienen su atributo igual a &valor */
sintaxis([Nvl,Atrib,"&",Valor|Resto],s_ecv(Nvl,Atrib,Valor),Resto):-
    nivel(Nvl,_),
    nivel(Nvl,List),
    miembro(Atrib,List),!.

/*dime el valor del atributo del nvl nom o el nvl nom que valor tiene su atrib*/
sintaxis([Atrib,Nvl,Nom|Resto],s_cen(Atrib,Nvl,Nom),Resto):-
    nivel(Nvl,List),
    miembro(Atrib,List),!.
sintaxis([Nvl,Nom,Atrib|Resto],s_cen(Atrib,Nvl,Nom),Resto):-
    nivel(Nvl,List),
    miembro(Atrib,List),!.

/*existe el nvl nom*/
sintaxis([Nvl,Nom|Resto],s_en(Nvl,Nom),Resto):-
    nom(Nvl,Nom),!.
```

```
/*dime los atrib del nvl o el nvl que atrib tiene*/
sintaxis([atributo,Nvl|Resto],s_ce(Nvl),Resto):-
     nivel(Nvl,_).
sintaxis([Nvl,atributo|Resto],s_ce(Nvl),Resto):-
     nivel(Nvl,_).

/*  verifica si solicita la consistencia de un nivel */
sintaxis([consistencia,Nvl,Nom|Resto],s_c(Nvl,Nom),Resto):-
     nom(Nvl,Nom),!.

/*que objetos hay*/
sintaxis([Nvl|Resto],s_e(Nvl),Resto):-
     nivel(Nvl,_),!.

/*que niveles hay*/
sintaxis([nivel|Resto],s_(nivel),Resto).

/*crea o modifica niveles*/
sintaxis([crea,Nvl|Resto],s_agreganvl(crea,Nvl),Resto):-
     nivel(Nvl,_),!.
sintaxis([modifica,Nvl|Resto],s_agreganvl(modifica,Nvl),Resto):-
     nivel(Nvl,_),!.
sintaxis([elimina,Nvl|Resto],s_agreganvl(elimina,Nvl),Resto):-
     nivel(Nvl,_),!.


/* ---------------------------------
      EVALUACION DEL TPO DE CONSULTA
---------------------------------- */

PREDICATES

evalúa(solic,symbol)
inter(symbol,symbol,symbol,symbol)
vedesc(symbol,symbol,integer)
atrib(symbol,symbol,symbol,symbol)
consist(symbol,symbol)
subconsist(symbol,listsym)
danvl(symbol,symbol)
agreganvl(symbol)
modificanvl(symbol)
eliminanvl(symbol)

CLAUSES

evalúa(s_een(Nvl1,Nvl2,Nom2),Nom1):-inter(Nvl1,Nom1,Nvl2,Nom2).
evalúa(s_een(Nvl2,Nvl1,Nom1),Nom2):-inter(Nvl1,Nom1,Nvl2,Nom2).

evalúa(s_een(Nvl1,Nom1,Nvl2),Nom2):-inter(Nvl1,Nom1,Nvl2,Nom2).
evalúa(s_een(Nvl2,Nom2,Nvl1),Nom1):-inter(Nvl1,Nom1,Nvl2,Nom2).

evalúa(s_ed(Nvl,no,descripción),Nom):-vedesc(Nvl,Nom,0).
evalúa(s_ed(Nvl,si,descripción),Nom):-vedesc(Nvl,Nom,1).
```

```
evalúa(s_ecv(Nvl,Atrib,Valor),Nom):-atrib(Nvl,Nom,Atrib,Valor).

evalúa(s_cen(Atrib,Nvl,Nom),Valor):-atrib(Nvl,Nom,Atrib,Valor).

evalúa(s_c(Nvl,Nom),""):-consist(Nvl,Nom).

evalúa(s_e(Nvl),Nom):-nom(Nvl,Nom).

evalúa(s_en(Nvl,Nom),""):-danvl(Nvl,Nom).

evalúa(s_ce(Nvl),Atrib):-nivel(Nvl,List),miembro(Atrib,List).

evalúa(s_(nivel),Nom):-nivel(Nom,_).

evalúa(s_agreganvl(crea,Nvl),""):-agreganvl(Nvl).
evalúa(s_agreganvl(modifica,Nvl),""):-modificanvl(Nvl).
evalúa(s_agreganvl(elimina,Nvl),""):-eliminanvl(Nvl).

inter(estructura,Estr,programa,Prog):-
      programa(Prog,Lista,_,_,_),
      miembro(Estr,Lista).
inter(estructura,Estr,archivo,Arch):-
      archivo(Arch,Estr,_).
inter(programa,Prog,corrida,Corr):-
      corrida(Corr,_,Lista,_,_),
      miembro(Prog,Lista).
inter(archivo,Arch,corrida,Corr):-
      corrida(Corr,_,_,Lista,_),
      miembro(Arch,Lista).
inter(corrida,Corr,módulo,Mod):-
      módulo(Mod,Lista,_),
      miembro(Corr,Lista).
inter(módulo,Mod,sistema,Sis):-
      sistema(Sis,Lista,_),
      miembro(Mod,Lista).
inter(estructura,Estr,corrida,Corr):-
      inter(estructura,Estr,programa,Prog),
      inter(programa,Prog,corrida,Corr),
      write("").
inter(estructura,Estr,módulo,Mod):-
      inter(estructura,Estr,corrida,Corr),
      inter(corrida,Corr,módulo,Mod),
      write("").
inter(estructura,Estr,sistema,Sis):-
      inter(estructura,Estr,módulo,Mod),
      inter(módulo,Mod,sistema,Sis),
      write("").
inter(archivo,Arch,módulo,Mod):-
      inter(archivo,Arch,corrida,Corr),
      inter(corrida,Corr,módulo,Mod),
      write("").
inter(archivo,Arch,sistema,Sis):-
      inter(archivo,Arch,módulo,Mod),
```

```
      inter(módulo,Mod,sistema,Sis),
      write("").
inter(programa,Prog,módulo,Mod):-
      inter(programa,Prog,corrida,Corr),
      inter(corrida,Corr,módulo,Mod),
      write("").
inter(programa,Prog,sistema,Sis):-
      inter(programa,Prog,módulo,Mod),
      inter(módulo,Mod,sistema,Sis),
      write("").
inter(programa,Prog,programa,Subprog):-
      programa(Prog,_,listprog,_,_),
      miembro(Subprog,listprog),
      write("").
inter(corrida,Corr,sistema,Sis):-
      inter(corrida,Corr,módulo,Mod),
      inter(módulo,Mod,sistema,Sis),
      write("").
inter(corrida,Corr,corrida,Subcorr):-
      corrida(Corr,Listcorr,_,_,_),
      miembro(Subcorr,Listcorr),
      write("").

vedesc(estructura,Nom,0):-estruct(Nom,_,_,_,Descrip),Descrip="".
vedesc(estructura,Nom,1):-estruct(Nom,_,_,_,Descrip),Descrip<>"".
vedesc(archivo,Nom,0):-archivo(Nom,_,Descrip),Descrip="".
vedesc(archivo,Nom,1):-archivo(Nom,_,Descrip),Descrip<>"".
vedesc(programa,Nom,0):-programa(Nom,_,_,_,Descrip),Descrip="".
vedesc(programa,Nom,1):-programa(Nom,_,_,_,Descrip),Descrip<>"".
vedesc(corrida,Nom,0):-corrida(Nom,_,_,_,Descrip),Descrip="".
vedesc(corrida,Nom,1):- corrida(Nom,_,_,_,Descrip),Descrip<>"".
vedesc(módulo,Nom,0):-módulo(Nom,_,Descrip),Descrip="".
vedesc(módulo,Nom,1):-módulo(Nom,_,Descrip),Descrip<>"".
vedesc(sistema,Nom,0):-sistema(Nom,_,Descrip),Descrip="".
vedesc(sistema,Nom,1):-sistema(Nom,_,Descrip),Descrip<>"".

atrib(estructura,Nom,longitud,Long):-
      estruct(Nom,_,_,Long,_).
atrib(estructura,Nom,organizacion,Org):-
      estruct(Nom,Org,_,_,_).
atrib(estructura,Nom,diccionario,""):-
      not(impresora(si)),
      estruct(Nom,_,Diccionario,_,_),
      editestr(Diccionario),
      clearwindow,!.
atrib(archivo,Nom,diccionario,Valor):-
      impresora(si),
      archivo(Nom,_,Diccionario),
      impestr(Diccionario,Valor),!.

atrib(estructura,Nom,descripción,""):-
      not(impresora(si)),
      estruct(Nom,_,_,_,Descrip),
      editestr(Descrip),
```

```
        clearwindow,!.
atrib(estructura,Nom,descripción,Valor):-
        impresora(si),
        estruct(Nom,_,_,_,Descrip),
        impestr(Descrip,Valor),!.

atrib(archivo,Nom,estructura,Valor):-
        archivo(Nom,Valor,_).
atrib(archivo,Nom,descripción,""):-
        not(impresora(si)),
        archivo(Nom,_,Descrip),
        editestr(Descrip),
        clearwindow,!.
atrib(archivo,Nom,descripción,Valor):-
        impresora(si),
        archivo(Nom,_,Descrip),
        impestr(Descrip,Valor),!.
atrib(programa,Nom,estructura,Valor):-
        programa(Nom,Lista,_,_,_),
        miembro(Valor,Lista).
atrib(programa,Nom,subprograma,Valor):-
        programa(Nom,_,Lista,_,_),
        miembro(Valor,Lista).
atrib(programa,Nom,dato,Valor):-
        programa(Nom,_,_,Listdato,_),
        miembro(Valor,Listdato).
atrib(programa,Nom,descripción,""):-
        not(impresora(si)),
        programa(Nom,_,_,_,Descrip),
        editestr(Descrip),
        clearwindow,!.
atrib(programa,Nom,descripción,Valor):-
        impresora(si),
        programa(Nom,_,_,_,Descrip),
        impestr(Descrip,Valor),!.

atrib(corrida,Nom,subcorrida,Valor):-
        corrida(Nom,Lista,_,_,_),
        miembro(Valor,Lista).
atrib(corrida,Nom,subprograma,Valor):-
        corrida(Nom,_,Lista,_,_),
        miembro(Valor,Lista).
atrib(corrida,Nom,programa,Valor):-
        corrida(Nom,_,Lista,_,_),
        miembro(Valor,Lista).
atrib(corrida,Nom,archivo,Valor):-
        corrida(Nom,_,_,Lista,_),
        miembro(Valor,Lista).
atrib(corrida,Nom,descripción,""):-
        not(impresora(si)),
        corrida(Nom,_,_,_,_,Descrip),
        editestr(Descrip),
        clearwindow,!.
atrib(corrida,Nom,descripción,Valor):-
```

```
        impresora(si),
        corrida(Nom,_,_,_,Descrip),
        impestr(Descrip,Valor),!.

atrib(módulo,Nom,corrida,Valor):-
        módulo(Nom,Lista,_),
        miembro(Valor,Lista).
atrib(módulo,Nom,descripción,""):-
        not(impresora(si)),
        módulo(Nom,_,Descrip),
        editestr(Descrip),
        clearwindow,!.
atrib(módulo,Nom,descripción,Valor):-
        impresora(si),
        módulo(Nom,_,Descrip),
        impestr(Descrip,Valor),!.

atrib(sistema,Nom,descripción,""):-
        not(impresora(si)),
        sistema(Nom,_,Descrip),
        editestr(Descrip),
        clearwindow,!.
atrib(sistema,Nom,descripción,Valor):-
        impresora(si),
        sistema(Nom,_,Descrip),
        impestr(Descrip,Valor),!.


/*verifica que el nivel Nvl sea consistente viendo que sus Nvl lo sean */
consist(sistema,[]).
consist(sistema,Nom):-
        sistema(Nom,Listmod,_),!,
        write(" "),
        subconsist(módulo,Listmod).
consist(sistema,Nom):-
        write("  No existe el sistema ",Nom),nl,!.

consist(módulo,[]).
consist(módulo,Nom):-
        módulo(Nom,Listcorr,_),!,
        subconsist(corrida,Listcorr).
consist(módulo,Nom):-
        write("  No existe el módulo ",Nom),nl,!.

consist(corrida,[]).
consist(corrida,Nom):-
        corrida(Nom,Listcorr,_,_,_),
        subconsist(corrida,Listcorr).
consist(corrida,Nom):-
        corrida(Nom,_,Listprog,_,_),
        subconsist(programa,Listprog).
consist(corrida,Nom):-
        corrida(Nom,_,_,Listarch,_),!,
        subconsist(archivo,Listarch).
```

```
consist(corrida,Nom):-
     write("  No existe la corrida o subcorrida ",Nom),nl,!.

consist(programa,[]).
consist(programa,Nom):-
     programa(Nom,Listestr,_,_,_),
     subconsist(estructura,Listestr).
consist(programa,Nom):-
     programa(Nom,_,Listprog,_,_),!,
     subconsist(programa,Listprog).
consist(programa,Nom):-
     write("  No existe el programa o subprograma",Nom),nl,!.

consist(archivo,[]).
consist(archivo,Nom):-
     archivo(Nom,Estr,_),
     subconsist(estructura,[Estr]),!.
consist(archivo,Nom):-
     write("  No existe el archivo ",Nom),nl,!.

consist(estructura,[]).
consist(estructura,Nom):-
     estruct(Nom,_,_,_,_),!.
consist(estructura,Nom):-
     write("  No existe la estructura ",Nom),nl,!.

subconsist(Nvl,Lista):-
     miembro(Nom,Lista),
     write("Verificando el(la) ",Nvl," >> ",Nom),nl,
     consist(Nvl,Nom).


danvl(estructura,Nom):-
     estruct(Nom,Org,Dic,Long,Descrip),!,
     write("El nombre de la estructura es: ",Nom),nl,
     write("\nSu organizacion es: ",Org),nl,
     write("\nSu diccionario de datos está en el archivo",Dic," y es: "),nl,
     impestr(Dic,Texto1),
     write(Texto1),
     write("\nSu longitud de registro es: ",Long),nl,
     write("\nSu descripción está en el archivo ",Descrip," y es: "),nl,
     nl.

danvl(archivo,Nom):-
     archivo(Nom,Estr,Descrip),!,
     write("El nombre del archivo es: ",Nom),nl,
     write("\nSu estructura se llama: ",Estr),nl,
     write("\nSu descripción está en el archivo ",Descrip," y es: "),nl,
     nl.

danvl(programa,Nom):-
     programa(Nom,Listarch,Listprog,Listdato,Descrip),!,
     write("El nombre del programa es: ",Nom),nl,
     write("\nLa lista de estructuras es:"),nl,
```

```
      write_list(0,Listarch),nl,
      write("\nLa lista de subprogramas es:"),nl,
      write_list(0,Listprog),nl,
      write("\nSu lista de datos de entrada es:"),nl,
      write_list(0,Listdato),nl,
      write("\nSu descripción está en el archivo ",Descrip," y es: "),nl,
      nl.

danvl(corrida,Nom):-
      corrida(Nom,Listcorr,Listprog,Listarch,Descrip),!,
      write("El nombre de la corrida es: ",Nom),nl,
      write("\nLa lista de subcorridas es:"),nl,
      write_list(0,Listcorr),nl,
      write("\nLa lista de programas es:"),nl,
      write_list(0,Listprog),nl,
      write("\nLa lista de archivos es:"),nl,
      write_list(0,Listarch),nl,
      write("\nSu descripción está en el archivo ",Descrip," y es: "),nl,
      nl.

danvl(módulo,Nom):-
      módulo(Nom,Listcorr,Descrip),!,
      write("El nombre del módulo es: ",Nom),nl,
      write("\nLa lista de corridas es:"),nl,
      write_list(0,Listcorr),nl,
      write("\nSu descripción está en el archivo ",Descrip," y es: "),nl,
      nl.

danvl(sistema,Nom):-
      sistema(Nom,Listmod,Descrip),!,
      write("El nombre del sistema es: ",Nom),nl,
      write("\nLa lista de módulos es:"),nl,
      write_list(0,Listmod),nl,
      write("\nSu descripción está en el archivo ",Descrip," y es: "),nl,
      nl.


agreganvl(estructura):-
      write("\n PROPORCIONA LOS DATOS DE LA NUEVA ESTRUCTURA:\n"),
      write("dame el nombre de la estructura\n"),
      readln(Nombre), Nombre<>"",
      not(estruct(Nombre,_,_,_,_)),
      write("dame el tipo de organizacion\n"),
      editsym("",Organiz,"Organizacion",3,50,5,25),
      write("nombre del docto para el dicec. de datos: ",Nombre),nl,
      concat(Nombre,".dic",Dic),
      editstr(Dic,Dicect,"Nombre",3,50,5,30),
      editestr(Dicect),
      write("dame la longitud del registro\n"),
      editsym("",Longreg,"Longitud",3,50,5,15),
      write("nombre del docto para la descripción de laestructura: ",Nombre),nl,
      concat(Nombre,".est",Desc),
      editstr(Desc,Descrito,"Nombre",3,50,5,30),
      editestr(Descrito),
```

```
        assert(estruct(Nombre,Organiz,Dicect,Longreg,Descrito)),
        agreganvl(estructura),!.
agreganvl(estructura).

agreganvl(archivo):-
        write("\n PROPORCIONA LOS DATOS DEL NUEVO ARCHIVO:\n"),
        write("dame el nombre del archivo\n"),
        readln(Nombre), Nombre<>"",
        not(archivo(Nombre,_,_)),
        write("dame la estructura\n"),
        editsym("",Estr,"Estructura",3,50,5,25),
        write("nombre del docto para la descripción del archivo: ",Nombre),nl,
        concat(Nombre,".arc",Desc),
        editstr(Desc,Descrito,"Nombre",3,50,5,30),
        editestr(Descrito),
        assert(archivo(Nombre,Estr,Descrito)),
        agreganvl(archivo),!.
agreganvl(archivo).

agreganvl(programa):-
        write("\n PROPORCIONA LOS DATOS DEL NUEVO PROGRAMA:\n"),
        write("dame el nombre del programa\n"),
        readln(Nombre), Nombre<>"",
        not(programa(Nombre,_,_,_,_)),
        write("dame la lista de estructuras\n"),
        editlist([],Listarch,"Lista de Estructuras",3,50,15,30),
        write("dame la lista de subprogramas\n"),
        editlist([],Listprog,"Lista de Subprogramas",3,50,15,30),
        write("dame la lista de datos de entrada\n"),
        editlist([],Listdatos,"Lista de Datos",3,50,15,30),
        write("nombre del docto para la descripción del programa: ",Nombre), nl,
        concat(Nombre,".pgr",Desc),
        editstr(Desc,Descrito,"Nombre",3,50,5,30),
        editestr(Descrito),
        assert(programa(Nombre,Listarch,Listprog,Listdatos,Descrito)),
        agreganvl(programa), !.
agreganvl(programa).

agreganvl(corrida):-
        write("\n PROPORCIONA LOS DATOS DE LA NUEVA CORRIDA:\n"),
        write("dame el nombre de la corrida\n"),
        readln(Nombre), Nombre<>"",
        not(corrida(Nombre,_,_,_,_)),
        write("dame la lista de subcorridas\n"),
        editlist([],Listcorr,"Lista de Subcorridas",3,50,15,30),
        write("dame la lista de programas\n"),
        editlist([],Listprog,"Lista de Programas",3,50,15,30),
        write("dame la lista de archivos\n"),
        editlist([],Listarch,"Lista de Archivos",3,50,15,30),
        write("Nombre del docto para la descripción de la corrida: ",Nombre), nl,
        concat(Nombre,".cor",Desc),
        editstr(Desc,Descrito,"Nombre",3,50,5,30),
        editestr(Descrito),
        assert(corrida(Nombre,Listcorr,Listprog,Listarch,Descrito)),
```

```
      agreganvl(corrida), !.
agreganvl(corrida).

agreganvl(módulo):-
      write("\n PROPORCIONA LOS DATOS DEL NUEVO PROCESO:\n"),
      write("dame el nombre del módulo\n"),
      readln(Nombre), Nombre<>"",
      not(módulo(Nombre,_,_)),
      write("dame la lista de corridas\n"),
      editlist([],Listcorr,"Lista de Corridas",3,50,15,30),
      write("nombre del docto para la descripción del módulo: ",Nombre), nl,
      concat(Nombre,".mod",Desc),
      editstr(Desc,Descrito,"Nombre",3,50,5,30),
      editestr(Descrito),
      assert(módulo(Nombre,Listcorr,Descrito)),
      agreganvl(módulo), !.
agreganvl(módulo).

agreganvl(sistema):-
      write("\n PROPORCIONA LOS DATOS DEL NUEVO SISTEMA:\n"),
      write("dame el nombre del sistema\n"),
      readln(Nombre), Nombre<>"",
      not(sistema(Nombre,_,_)),
      write("dame la lista de módulos\n"),
      editlist([],Listmod,"Lista de Módulos",3,50,15,30),
      write("nombre del docto para la descripción del sistema: ",Nombre), nl,
      concat(Nombre,".sis",Desc),
      editstr(Desc,Descrito,"Nombre",3,50,5,30),
      editestr(Descrito),
      assert(sistema(Nombre,Listmod,Descrito)),
      agreganvl(sistema), !.
agreganvl(sistema).


/*modifica los datos de las estructuras que se den */
modificanvl(estructura):-
      write("\n PROPORCIONA LOS NUEVOS DATOS DE LA ESTRUCTURA:\n"),
      write("Dame el nombre de la estructura o ESPACIOS para terminar\n"),
      readln(Nombre), Nombre<>"",
      estruct(Nombre,Organiz,Dic,Longreg,Descrito),
      write("Modifica el tipo de organizacion\n"),
      editsym(Organiz,Norganiz,"Organizacion",3,50,5,25),
      write("Modifica su dicc. de datos: ",Dic), nl,
      editestr(Dic),
      write("dame la longitud del registro\n"),
      editsym(Longreg,Nlongreg,"Longitud",3,50,5,15),
      write("Modifica el nombre del docto de su descripción\n"),
      editstr(Descrito,NDescrito,"Nombre",3,50,5,30),
      write("Modifica su descripción en: ",NDescrito), nl,
      editestr(NDescrito),
      retract(estruct(Nombre,Organiz,Dic,Longreg,Descrito)),
      assert(estruct(Nombre,NOrganiz,Dic,NLongreg,NDescrito)),
      modificanvl(estructura),!.
modificanvl(estructura).
```

```
modificanvl(archivo):-
      write("\n PROPORCIONA LOS NUEVOS DATOS DEL ARCHIVO:\n"),
      write("Dame el nombre del archivo o ESPACIOS para terminar\n"),
      readln(Nombre), Nombre<>"",
      archivo(Nombre,Estr,Descrito),
      write("Modifica la estructura\n"),
      editsym(Estr,Nestr,"Estructura",3,50,5,25),
      write("Modifica el nombre del docto de su descripción\n"),
      editstr(Descrito,NDescrito,"Nombre",3,50,5,30),
      write("Modifica su descripción en: ",NDescrito), nl,
      editestr(NDescrito),
      retract(archivo(Nombre,Estr,Descrito)),
      assert(archivo(Nombre,Nestr,NDescrito)),
      modificanvl(archivo),!.
modificanvl(archivo).

modificanvl(programa):-
      write("\nPROPORCIONA LOS NUEVOS DATOS DEL PROGRAMA:\n"),
      write("dame el nombre del programa o ESPACIOS para terminar\n"),
      readln(Nombre), Nombre<>"",
      programa(Nombre,Listarch,Listprog,Listdatos,Descrito),
      write("modifica la lista de archivos\n"),
      editlist(Listarch,NLarch,"Lista de Archivos",3,50,15,30),
      write("modifica la lista de programas\n"),
      editlist(Listprog,NLprog,"Lista de Programas",3,50,15,30),
      write("modifica la lista de datos de entrada\n"),
      editlist(Listdatos,NListdatos,"Lista de Datos",3,50,15,30),
      write("Modifica el nombre del docto de su descripción\n"),
      editstr(Descrito,NDescrito,"Nombre",3,50,5,30),
      write("modifica su descripción en : ",NDescrito), nl,
      editestr(NDescrito),
      retract(programa(Nombre,Listarch,Listprog,Listdatos,Descrito)),
      assert(programa(Nombre,NLarch,NLprog,NListdatos,NDescrito)),
      modificanvl(programa),!.
modificanvl(programa).

modificanvl(corrida):-
      write("\nPROPORCIONA LOS NUEVOS DATOS DE LA CORRIDA:\n"),
      write("dame el nombre de la corrida o ESPACIOS para terminar\n"),
      readln(Nombre), Nombre<>"",
      corrida(Nombre,Listcorr,Listprog,Listarch,Descrito),
      write("modifica la lista de corridas\n"),
      editlist(Listcorr,Nlistcorr,"Lista de Corridas",3,50,15,30),
      write("Modifica la lista de programas\n"),
      editlist(Listprog,Nlistprog,"Lista de Programas",3,50,15,30),
      write("Modifica la lista de archivos\n"),
      editlist(Listarch,Nlistarch,"Lista de Archivos",3,50,15,30),
      write("Modifica el nombre del docto de su descripción\n"),
      editstr(Descrito,NDescrito,"Nombre",3,50,5,30),
      write("Modifica su descripción en: ",NDescrito), nl,
      editestr(NDescrito),
      retract(corrida(Nombre,Listcorr,Listprog,Listarch,Descrito)),
      assert(corrida(Nombre,Nlistcorr,Nlistprog,Nlistarch,NDescrito)),
```

```
      modificanvl(corrida), !.
modificanvl(corrida).

modificanvl(módulo):-
      write("\nPROPORCIONA LOS NUEVOS DATOS DEL MODULO:\n"),
      write("dame el nombre del módulo o ESPACIOS para terminar\n"),
      readln(Nombre), Nombre<>"",
      módulo(Nombre,Listcorr,Descrito),
      write("Modifica la lista de corridas\n"),
      editlist(Listcorr,Nlistcorr,"Lista de Corridas",3,50,15,30),
      write("Modifica el nombre del docto de su descripción\n"),
      editstr(Descrito,NDescrito,"Nombre",3,50,5,30),
      write("Modifica su descripción en: ",NDescrito), nl,
      editestr(NDescrito),
      retract(módulo(Nombre,Listcorr,Descrito)),
      assert(módulo(Nombre,Nlistcorr,NDescrito)),
      modificanvl(módulo), !.
modificanvl(módulo).

modificanvl(sistema):-
      write("\nPROPORCIONA LOS NUEVOS DATOS DEL SISTEMA:\n"),
      write("dame el nombre del sistema o ESPACIOS para terminar\n"),
      readln(Nombre), Nombre<>"",
      sistema(Nombre,Listmod,Descrito),
      write("Modifica la lista de módulos\n"),
      editlist(Listmod,Nlistmod,"Lista de Módulos",3,50,15,30),
      write("Modifica el nombre del docto de su descripción\n"),
      editstr(Descrito,NDescrito,"Nombre",3,50,5,30),
      write("Modifica su descripción en: ",NDescrito),nl,
      editestr(NDescrito),
      retract(sistema(Nombre,Listmod,Descrito)),
      assert(sistema(Nombre,Nlistmod,NDescrito)),
      modificanvl(sistema),!.
modificanvl(sistema).

eliminanvl(estructura):-
      write("\nDAME EL NOMBRE DE LA ESTRUCTURA A ELIMINAR:\n"),
      readln(Nom),Nom<>"",
      estruct(Nom,Org,Dic,Lon,Des),
      retract(estruct(Nom,Org,Dic,Lon,Des)),!.

eliminanvl(archivo):-
      write("\nDAME EL NOMBRE DEL ARCHIVO A ELIMINAR:\n"),
      readln(Nom),Nom<>"",
      archivo(Nom,Est,Des),
      retract(archivo(Nom,Est,Des)),!.

eliminanvl(programa):-
      write("\nDAME EL NOMBRE DEL PROGRAMA A ELIMINAR:\n"),
      readln(Nom),Nom<>"",
      programa(Nom,Est,Prog,Dat,Des),
      retract(programa(Nom,Est,Prog,Dat,Des)),!.

eliminanvl(corrida):-
```

```
      write("\nDAME EL NOMBRE DE LA CORRIDA A ELIMINAR:\n"),
      readln(Nom),Nom<>"",
      corrida(Nom,Corr,Prog,Arch,Des),
      retract(corrida(Nom,Corr,Prog,Arch,Des)),!.

eliminanvl(módulo):-
      write("\nDAME EL NOMBRE DEL MODULO A ELIMINAR:\n"),
      readln(Nom),Nom<>"",
      módulo(Nom,Corr,Des),
      retract(módulo(Nom,Corr,Des)),!.

eliminanvl(sistema):-
      write("\nDAME EL NOMBRE DEL SISTEMA A ELIMINAR:\n"),
      readln(Nom),Nom<>"",      sistema(Nom,Mod,Des),
      retract(sistema(Nom,Mod,Des)),!.

eliminanvl(Nvl):-write("\nNo existe tal ",Nvl).

/* --------------
   VER EL LENGUAJE
   -------------- */

PREDICATES

dasinónimo(symbol)
daatrib(symbol)
daatrib1(symbol)
veleng velengl(INTEGER)
actleng actlengl(INTEGER)
nvosinnvl
nvosinatrib

CLAUSES

veleng:-
      repite,
      menú(5,40,"Lenguaje",
           ("Palabras Reservadas",
            "Sinónimos de P. Reservadas"
           ),Selección),
      nl,velengl(Selección),
      Selección=0,!.


velengl(0).


velengl(1):-
      write("Palabras Reservadas\n------------------\n"),
      findall(X,sinónimos(_,X),L),único(L,L1),write_list(0,L1),
      nl.

velengl(1):-
      write("\n\nDigita RETURN para continuar"),
```

```
     readchar(_).

veleng1(2):-
     writef("%-15 %-15\n","Sinónimo","P. Reservada"),
     write("--------------- ---------------\n"),
     sinónimos(E,S),writef("%-15 %-15\n",E,S),fail.

veleng1(2):-
     write("\n\nDigita RETURN para continuar"),
     readchar(_).

/* ---------------------
    ACTUALIZAR EL LENGUAJE
    ---------------------- */
CLAUSES
dasinónimo(E):-
     findall(X,sinónimos(_,X),L),
     único(L,L1),
     menú(2,49,"P. Reservada",L1,C),
     índice(L1,C,E).
daatrib(E):-
     findall(X,daatrib1(X),L),
     único(L,L1),
     menú(2,49,"Atributo",L1,C),
     índice(L1,C,E).

daatrib1(Atrib):-
     nivel(_,List),
     miembro(Atrib,List).


actleng:-
     repite,
     menú(5,40,"Actualiza el Lenguaje",
       [ "Nuevos sinónimos para P. Reserv.",
         "Nuevos sinónimos para Atributo"
       ],CHOICE),
     nl,actleng1(CHOICE),CHOICE=0,!.

actleng1(0).
actleng1(1):-nvosinnvl.
actleng1(2):-nvosinatrib.

nvosinnvl:-
     dasinónimo(E),
     write("Sinónimo: "),
     readln(SYNONYM),SYNONYM><"",
     assert(sinónimos(SYNONYM,E)),
     nvosinnvl.

nvosinatrib:-
     daatrib(E),
     write("Sinónimo: "),
     readln(SYNONYM),
```

```
    SYNONYM><"",
    assert(sinónimos(SYNONYM,E)),
    nvosinatrib.


/*-------------------------
  PROCEDIMIENTOS DE CONSULTA
  ------------------------- */
PREDICATES
itera(SYMBOL)
leeconsul(symbol)

CLAUSES
leeconsul(QUERY):-
     nl,nl,
     write("Consulta: "),
     readln(QUERY).

     itera(STR):-
          STR >< "",
          nl,nl,
          scan(STR,List),
          filtra(List,List1),
          pars(List1,Q),
          findall(A,evalúa(Q,A),L),
          único(L,L1),
          imp_result(L1,STR),
          fail.

     itera(STR):-
          STR >< "",
          leeconsul(L),
          itera(L).

/*--------------------------------
            MENU PRINCIPAL
  --------------------------------*/

PREDICATES

consulta
menú_inic
proceso(INTEGER)

GOAL
consulta.

CLAUSES
consulta:-
     makewindow(21,7,0,"",24,0,1,80),
     write("ESC: Fin del menú -- Utilizar las flechas para seleccionar la
     opción."),
     makewindow(22,7,0,"",23,0,1,80),
     write("Esc: Fin      F8: Repite línea    Ctrl S: Parar impresion     End:
```

```
     SYNONYM><"",
     assert(sinónimos(SYNONYM,E)),
     nvosinatrib.


/*-------------------------
   PROCEDIMIENTOS DE CONSULTA
   ------------------------- */
PREDICATES
itera(SYMBOL)
leeconsul(symbol)

CLAUSES
leeconsul(QUERY):-
     nl,nl,
     write("Consulta: "),
     readln(QUERY).

     itera(STR):-
          STR >< "",
          nl,nl,
          scan(STR,List),
          filtra(List,List1),
          pars(List1,Q),
          findall(A,evalúa(Q,A),L),
          único(L,L1),
          imp_result(L1,STR),
          fail.

     itera(STR):-
          STR >< "",
          leeconsul(L),
          itera(L).

/*-------------------------------
             MENU PRINCIPAL
   -------------------------------*/

PREDICATES

consulta
menú_inic
proceso(INTEGER)

GOAL
consulta.

CLAUSES
consulta:-
     makewindow(21,7,0,"",24,0,1,80),
     write("ESC: Fin del menú -- Utilizar las flechas para seleccionar la
     opción."),
     makewindow(22,7,0,"",23,0,1,80),
     write("Esc: Fin      F8: Repite linea    Ctrl S: Parar impresion     End:
```

```
      Saltar fin linea"),
      makewindow(2,7,7,"SISDOC: Sistema de Documentación/Consultas en Lenguaje
      Natural",0,0,23,80),
      proceso(99),
      menú_inic.

menú_inic:-
      repite,
      menú(8,49,"Menú Principal",
         [ "Consultar la Base de Datos",
           "Ver el Lenguaje",
           "Actualizar el Lenguaje",
           "Tutorial",          "Sistema Operativo",
           "Almacenar la Base de Datos"
         ],Selección),
      proceso(Selección),
      Selección=0,!,
      removewindow,removewindow,removewindow.

proceso(0):-
      write("\nEstas seguro de querer terminar? (s/n): "),
      readchar(T),T='s',
      proceso(98).

proceso(1):-
      ieeconsul(L),itera(L).

proceso(2):-
      veleng.

proceso(3):-
      actleng.

proceso(4):-
      file_str("sisdoc.hlp",TXT),display(TXT),clearwindow,!.
proceso(4):-
      write(">> sisdoc.hlp no está en el diccionario por omisión\n").

proceso(5):-
      makewindow(3,7,0,"",0,0,25,80),
      write("Digita EXIT para regresar\n\n"),
      system(""),!,
      removewindow.
proceso(5):-
      write(">> command.com no está accesible. digita RETURN"),
      readchar(_),
      removewindow.
      /* guarda la base de datos. Como se puede no guardarla
      siempre regresa exito */

proceso(6):-
      proceso(98).
```

```
proceso(98):-
     write("\nQuieres guardar la base de datos? (s/n): "),
     readchar(T),T='s',
     write("\nEspera un momento, estoy guardando la base de datos"),
     deletefile("sisdoc.bak"),
     renamefile("sisdoc.dba","sisdoc.bak"),
     save("sisdoc.dba").
proceso(98).


proceso(99):-
     write("Cargando la Base de Datos - favor de esperar\n"),
     consult("sisdoc.dba"),!.
proceso(99):-
     write(">> sisdoc.dba no está el el diccionario por omisión\n").

/*---------------------- FIN DEL PROGRAMA -----------------------*/
```

ANEXO B

LISTADO DE COMPILACION

```
Predicate Name  Dbase Determ Size  Doml -- flowpattern
--------------- ----- ------ ----- ------------------------
goal            NO    YES    118  --
impresora       YES   NO     104 symbol -- i
sinónimos       YES   NO     120 symbol,symbol -- i,o
sinónimos       YES   NO     120 symbol,symbol -- o,i
sinónimos       YES   NO     118 symbol,symbol -- o,o
estruct         YES   NO     194
symbol,symbol,string,symbol,string
                                  -- i,o,o,o,o
estruct         YES   NO     192
symbol,symbol,string,symbol,string
                                  -- o,o,o,o,o
estruct         YES   NO     194
symbol,symbol,string,symbol,string
                                  -- o,o,o,i,o
estruct         YES   NO     194
symbol,symbol,string,symbol,string
                                  -- o,i,o,o,o
archivo         YES   NO     150 symbol,symbol,string --
i,o,o
archivo         YES   NO     148 symbol,symbol,string --
o,o,o
archivo         YES   NO     150 symbol,symbol,string --
o,i,o
archivo         YES   NO     155 symbol,symbol,string --
i,i,o
programa        YES   NO     228
symbol,listsym,listsym,listsym,string
                                  -- i,o,o,o,o
programa        YES   NO     226
symbol,listsym,listsym,listsym,string
                                  -- o,o,o,o,o
corrida         YES   NO     228
symbol,listsym,listsym,listsym,string
                                  -- i,o,o,o,o
corrida         YES   NO     226
symbol,listsym,listsym,listsym,string
                                  -- o,o,o,o,o
módulo          YES   NO     166 symbol,listsym,string --
i,o,o
módulo          YES   NO     163 symbol,listsym,string --
o,o,o
sistema         YES   NO     166 symbol,listsym,string --
i,o,o
sistema         YES   NO     163 symbol,listsym,string --
o,o,o
nivel           YES   NO     116 symbol,listsym -- o,o
nivel           YES   YES    106 symbol,listsym -- i,o
```

```
nom             NO     NO       393 symbol,symbol -- o,i
nom             NO     NO       491 symbol,symbol -- i,i
nom             NO     NO       491 symbol,symbol -- i,o
miembro         NO     NO        84 symbol,listsym -- i,i
miembro         NO     NO        78 symbol,listsym -- o,i
maxlen          NO     YES      174 listsym,integer,integer --
i,i,o
listlen         NO     YES       83 listsym,integer -- i,o
único           NO     YES      204 listsym,listsym -- i,o
repite          NO     NO        37  --
indice          NO     YES       98 listsym,integer,symbol --
i,i,o
impestr         NO     YES      121 string,symbol -- i,i
impestr         NO     YES      154 string,symbol -- i,o
writelist       NO     YES      120 integer,integer,listsym --
i,i,i
write_list      NO     YES      533 integer,listsym -- i,i
imp_result      NO     NO       545 listsym,symbol -- i,i
scan            NO     NO       186 symbol,listsym -- i,o
pega_car        NO     YES     1281
symbol,symbol,symbol,symbol -- i,i,o,o
filtra          NO     YES      596 listsym,listsym -- i,o
conocida        NO     NO       333 symbol -- i
readkey         NO     YES       64 key -- o
readkey1        NO     YES      166 key,char,integer -- o,i,i
readkey2        NO     YES      192 key,integer -- o,i
menú            NO     YES      359
row,col,string,listsym,integer
                                    -- i,i,i,i,o
menú1           NO     YES      110
row,listsym,row,integer,integer
                                    -- i,i,i,i,o
menú2           NO     YES      349
row,listsym,row,integer,integer,key
                                    -- i,i,i,i,o,i
editsym         NO     NO       230
symbol,symbol,string,integer,integer,integ
er,integer
                                    -- i,o,i,i,i,i,i
editstr         NO     NO       200
string,string,string,integer,integer,integ
er,integer
                                    -- i,o,i,i,i,i,i
editlist        NO     NO       260
listsym,listsym,string,integer,integer,int
eger,integer
                                    -- i,o,i,i,i,i,i
editint         NOT USED
integer,integer,string,integer,integer,int
eger,integer
descan          NO     YES      172 listsym,symbol -- i,o
editestr        NO     YES      786 string -- i
pars            NO     YES      124 listsym,solic -- i,o
```

```
sintaxis          NO     NO       3665 listsym,solic,listsym --
i,o,o
error             NO     YES       190 listsym -- i
vefin             NO     YES        30 listsym -- i
evalúa            NO     NO        935 solic,symbol -- i,o
inter             NO     NO       1982
symbol,symbol,symbol,symbol -- i,o,i,i
inter             NO     NO       1982
symbol,symbol,symbol,symbol -- i,o,i,o
inter             NO     NO       1982
symbol,symbol,symbol,symbol -- i,i,i,o
inter             NO     NO       1982
symbol,symbol,symbol,symbol -- i,i,i,i
vedesc            NO     NO       1154 symbol,symbol,integer --
i,o,i
atrib             NO     NO       3827
symbol,symbol,symbol,symbol -- i,o,i,i
atrib             NO     NO       3792
symbol,symbol,symbol,symbol -- i,i,i,o
consist           NO     NO       1653 symbol,symbol -- i,i
subconsist        NO     NO        109 symbol,listsym -- i,i
danvl             NO     YES      1585 symbol,symbol -- i,i
agreganvl         NO     YES      3653 symbol -- i
modificanvl       NO     YES      5308 symbol -- i
eliminanvl        NO     YES      3159 symbol -- i
dasinónimo        NO     YES       228 symbol -- o
daatrib           NO     YES       223 symbol -- o
daatrib1          NO     NO         65 symbol -- o
veleng            NO     YES       142  --
veleng1           NO     NO        514 integer -- i
actleng           NO     YES       142  --
actleng1          NO     YES        56 integer -- i
nvosinnvl         NO     YES       206  --
nvosinatrib       NO     YES       206  --
itera             NO     YES       388 symbol -- i
leeconsul         NO     YES       104 symbol -- o
consulta          NO     NO        282  --
menú_inic         NO     YES       162  --
proceso           NO     NO        852 integer -- i
----------------- ------ ------ ------ ------------------------
Total size                      53317
```

Es importante observar en este listado el uso que se le da a cada uno de los parámetros de cada predicado del programa.

ANEXO C

AYUDAS EN LINEA

1 Lenguaje de Consulta

SISDOC tiene por objetivo documentar Sistemas y permitir la consulta de la documentación de los sistemas en línea.

La forma de utilizar a SISDOC es a través de menús y de un subconjunto del lenguaje natural vía palabras reservadas.

Para las actividades mas genéricas se utilizan los menús, y para las consultas de la documentación se utiliza un lenguaje de palabras reservadas el cual tiene la siguiente gramática:

Ver la sección 3 del Capítulo 3.

2 Comandos del Editor de Texto

Dado que el editor de texto es una parte integral del ambiente de Turbo Prolog, las funciones y comandos estan en idioma ingles. Estos son:

2.1 Funciones especiales

F U N C T I O N K E Y S

F1 Pop-Up Help	F2 Goto to Line
F3 Search Srch Again	F4 Replace R. Again
F5 Copy Copy Again	F6 Move Text
F7 Delete Text	F8 Aux Edit
F9 Extern copy View wind	F10 End editor Resize wind

2.2 Combinaciones especiales

SPECIAL COMBINATIONS

Ctrl	<-	Move to previous
Ctrl	->	Move to next word
Ctrl	PgUp	Goto to start of
Ctrl	PgDn	Goto the end of
Ctrl	Home	Goto start of page
Ctrl	End	Goto end of page
Ctrl	BKS	Delete actual line
Ctrl	V	Insert/Overwrite
Del		Delete
Ins		Insert/Overwrite

2.3 Comandos compatibles con WORDSTAR

CURSOR MOVEMENTS:	COMMAND	KEY
Character left	Ctrl-S	
Character right	Ctrl-D	
Word left	Ctrl-A	Ctrl -
Word right	Ctrl-F	Ctrl -
Line up	Ctrl-E	
Line down	Ctrl-X	
Page up	Ctrl-R	PgUp
Page down	Ctrl-C	PgDn
To left on line	Ctrl-Q-S	Home
To right on line	Ctrl-Q-D	End
To top of page	Ctrl-Q-R	Ctrl - Home
To bottom of pag	Ctrl-Q-X	Ctrl - End
To top of file	Ctrl-Q-R	Ctrl - PgUp
To end of file	Ctrl-Q-C	Ctrl - PgDn
To beginning of block	Ctrl-Q-B	
To end of block	Ctrl-Q-K	
To last cursor position	Ctrl-Q-P	

INSERT & DELETE	COMMAND	KEY
Insert mode on/off	Ctrl-V	Ins
Insert line	Ctrl-N	
Delete line	Ctrl-Y	
Delete to end of line	Ctrl-Q-Y	
Delete right word	Ctrl-T	
Delete char under cursor	Ctrl-G	Del
Delete left character	Ctrl-H	

BLOCK COMMANDS	COMMAND	KEY
Mark block begin	Ctrl-K-B	F7
Mark block end	Ctrl-K-K	F8
Hide/display block	Ctrl-K-H	
Copy block	Ctrl-K-C	
Move block	Ctrl-K-V	
Delete block	Ctrl-K-Y	
Read block from disk	Ctrl-K-R	
Write block to disk	Ctrl-K-W	
Print block	Ctrl-K-P	

MISC. COMMANDS	COMMAND	KEY
Tab	Ctrl-I	
Repeat last find/replace	Ctrl-L	
Find	Ctrl-Q-F	
Find & replace	Ctrl-Q-A	
Auto tab on/off	Ctrl-Q-I	

ANEXO D

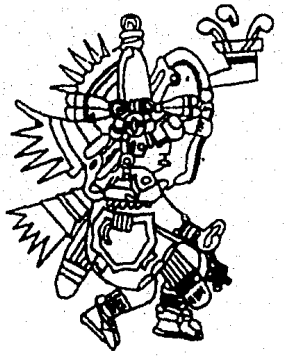
EJEMPLO DE UNA BASE DE DATOS

```
sinónimos("altera","modifica")
sinónimos("archivos","archivo")
sinónimos("atributos","atributo")
sinónimos("cambia","modifica")
sinónimos("con","si")
sinónimos("consistente","consistencia")
sinónimos("construye","crea")
sinónimos("corridas","corrida")
sinónimos("crear","crea")
sinónimos("datos-de-entrada","dato")
sinónimos("eliminar","elimina")
sinónimos("estructuras","estructura")
sinónimos("lista","imprime")
sinónimos("modificacion","modifica")
sinónimos("modificar","modifica")
sinónimos("módulos","módulo")
sinónimos("programas","programa")
sinónimos("quita","elimina")
sinónimos("reporta","imprime")
sinónimos("sin","no")
sinónimos("sistemas","sistema")
sinónimos("subcorridas","corrida")
sinónimos("subprogramas","programa")
sinónimos("subrutina","programa")
sinónimos("subrutinas","programa")

estruct("bajas","secuencial","procs.dic","10","estruct.est")
estruct("catálogos","indexada","catálogos.dic","120","estruc.est")
estruct("cuentas-bm","secuencia","procs.dic","120","estruct.est")
estruct("fechas","","fechas.dic","","cier-men.est")
estruct("historico","","historico.dic","","cier-men.est")
estruct("list-ref-micro","secuencial","procs.dic","128","estruct.est")
estruct("listado-act1","","cier-men.dic","","cier-men.est")
estruct("listado-act2","","listado-act2.dic","","cier-men.est")
estruct("movscont","indexada","procs.dic","128","estruct.est")
estruct("número-folio","secuencial","procs.dic","10","estruct.est")

archivo("abs","elementos","proc-aux.arc")
archivo("corr","elementos","proc-aux.arc")
archivo("ftes","elementos","proc-aux.arc")
archivo("alterna","secuencial","proc-aux.arc")
archivo("contab*fichas-men","movscont","pol-man.arc")
archivo("contab*fichas-mov","aplicacion","pol-man.arc")
archivo("contab*fichas-tri","aplicacion","pol-man.arc")
archivo("bajas","secuencial","proc-aux.arc")
archivo("bup-hist","movscont","cier-men.arc")
archivo("bup-movs","movscont","cier-men.arc")

programa("act-
mensual",["saldos","reportes"],[],["respuesta"],"progs2.pgr")
```

```
programa("act-saldos",["movscont","saldos","catálogos","listado-
act1","listado-act2"],[],["fproceso"],"progs2.pgr")
programa("acu-movscont",["movscont"],[],[],"progs4.pgr")
programa("altas-ctas",["saldos","reportes","card-reader"],["dig-
ver"],[],"progs3.pgr")
programa("amplia-
hist",["movscont","historico"],[],["fvalor"],"progs2.pgr")
programa("contab*abe.asigna-cifra",[],[],[],"progs.pgr")
programa("contab*abe.com66",[],[],[],"progs.pgr")
programa("contab*abe.crea-apli66",[],[],[],"progs.pgr")
programa("auditoria2",["saldos","movscont","reportes","sort"],["dig-
ver"],["opción","fecha-1","fecha-2","cuenta2"],"progs2.pgr")
programa("bajas-ctas",["saldos","reportes","bajas"],[],[],"progs4.pgr")

corrida("corr.listado",[],[],[],"corr.cor")
corrida("act-mensual",[],["act-mensual"],["saldos","nsaldos","list-act-
m"],"corr.cor")
corrida("act-saldos",[],["act-saldos"],["saldos-
nvo","saldos","catálogos","listado-act1","listado-
act2","fvalor"],"corr.cor")
corrida("altas-ctas",[],["altas-ctas"],["saldos","listado-
cat"],"corr.cor")
corrida("baja-ctas",[],["bajas-ctas"],["saldos","bajas","listado-
baj"],"corr.cor")
corrida("baja-mes-1",[],[],["i-baja-mes1","saldos-xxxx","hism-
xxxx","póliza-ap","movs-xxxx","resmes1"],"corr.cor")
corrida("baja-mes1",[],[],["i-baja-mes1","saldos-xxxx","hism-
xxxx","póliza-ap","saldos-xxxx","movs-xxxx","resmes1"],"corr.cor")
corrida("baja-mes2",[],[],[],"corr.cor")
corrida("baja-mes3",[],[],["i-baja-mes3","saldos-xxxx","hism-
xxxx","póliza-ap","movs-xxxx","cnt"],"corr.cor")

módulo("actualización-saldos",["m-
310","balanza","respaldo1","respaldo2","respaldo3","recupera1","recupera2
","recupera3"],"módulos2.mod")
módulo("captura-pólizas",["pólizas","transmisión-a-mainframe","m-
valid","m-111"],"módulos.mod")
módulo("cierre-anual",["ceros","genpol-cierr","listado-anual","genpol-
ap","ini-anual"],"módulos3.mod")
módulo("cierre-mensual",["cop-410","m-410","ini-arch","listpol-
esp","list-esp","respaldo-820","resp-mes1","resp-mes2","resp-mes3","resp-
mes4","baja-mes1","baja-mes2","baja-mes3","baja-mes4"],"módulos3.mod")
módulo("modelo",[],"modelo.mod")
módulo("pólizas-manuales",["m-211","m-212","m-213","m-214","m-215","m-
216","m-217","m-218","m-219","m-220","m-221","m-222","m-223","genf-esp-
men","genf-esp-mvo","genf-esp-tri","fichas-men","fichas-mov","fichas-
tri"],"módulos1.mod")
módulo("si",[],"si.mod")

sistema("contabilidad",["captura-pólizas","pólizas-
manuales","actualización-saldos","cierre-mensual","cierre-
anual","procesos-auxiliares"],"contab.sis")
```

QUETZALCOATL

Quetzalcóatl, fue quizás el más complejo y fascinante de todos los Dioses mesoamericanos. Su concepto primordial, sin duda muy antiguo en el área, parece haber sido el de un monstruo serpiente celeste con funciones dominantes de fertilidad y creatividad. A este núcleo se agregaron gradualmente otros aspectos: la leyenda lo había mezclado con la vida y los hechos del gran Rey sacerdote Topiltzin, cuyo título sacerdotal era el propio nombre del Dios del que fue especial devoto. En el momento de la conquista, Quetzalcóatl, considerado como Dios único desempeñaba varias funciones: Creador, Dios del viento, Dios del planeta Venus, héroe cultural, arquetipo del sacerdocio, patrón del calendario y de las actividades intelectuales en general, etc. Un análisis adicional es necesario para poder desentrañar los hilos aparentemente independientes que entran al tejido de su complicada personalidad.



IMPRESO EN LOS TALLERES DE:
EDITORIAL QUETZALCOATL, S. A.
MEDICINA No. 37 LOCALES 1 Y 2 (ENTRADA POR PASEO DE LAS FACULTADES) FRENTE A LA FACULTAD DE MEDICINA DE C. U.
MEXICO 20, D. F. TELEFONOS 658-71-66 Y 658-70-88