59
24.

Universidad Nacional Autónoma de México

Escuela Nacional de Estudios Profesionales
Acatlán

SIPRES, UN SISTEMA PARA AUTOMATIZAR EL PROCEDIMIENTO DE ARQUEO AL FONDO FIJO DE LA ENEP ACATLÁN

Memoria del Desempeño Profesional
que para obtener el título de
Licenciado en Matemáticas Aplicadas y Computación
Presenta
Jaime Vergara Prado

Asesor: Mtro. Víctor José Palencia Gómez.

1997.

AGRADECIMIENTOS

A quienes debo todo lo que soy: mis padres, Juventina y Guillermo.

A mis hermanos, Guadalupe, Juan y Guillermo, por el apoyo y cariño que siempre me han brindado.

A mis cuñadas, Irene y Fabiola, por hacer felices a mis hermanos, a mi sobrina Nancy, por llenar la vida de mi hermana.

A los profesores de la carrera, que me formaron como profesional y como persona, y que se convirtieron ya en mis amigos: Beatriz Clavel, Manuel Valadez y Víctor Palencia.

A mis amigos de tanto tiempo, con quienes he compartido alegrías, tristezas, trabajos, éxitos y fracasos: Claudia, Dolores, Elba, Laura, Angélica, Fernando, Lorenzo, Rubí, José, Carmen, Alan, Mauricio, Eduardo, Julián, Javier, Alberto, Blanca, Delia, Clau, Claus, Alex, Hilda, Paty y Sergio.

A todos los que han creído en mí.

A M., por todo.

ÍNDICE.

INTRODUCCIÓN.

Para una persona que desea obtener un título profesional, elegir una determinada forma de hacerlo depende mucho de qué tan definidas tenga las metas que desea alcanzar en su vida, por lo menos en el corto plazo; de sus características personales; e incluso, también depende de las circunstancias que en ese momento influyan en su modo de vivir.

Haber analizado los tres puntos anteriores dio por resultado el presente trabajo, el cual constituye una memoria de desempeño profesional, puesto que pretende mostrar la experiencia profesional del autor en el uso, práctica y aplicación de los conocimientos adquiridos en la licenciatura de Matemáticas Aplicadas y Computación en un proyecto específico: *SIPRES*, el sistema que automatiza los procedimientos relativos al fondo fijo de la ENEP Acatlán.

Haber elegido a *SIPRES* como materia de este estudio obedece a que, a pesar de ser un sistema relativamente pequeño en comparación con otros en que ha participado quien esto escribe durante su labor como analista y líder de proyecto en el Departamento de Sistemas de Información de la ENEP Acatlán,

ha sido el que más conocimientos le permitió adquirir: un nuevo lenguaje de programación, una nueva metodología de programación y un nuevo ambiente de desarrollo de aplicaciones.

Los objetivos que este estudio persigue (y que evidentemente son independientes a los del sistema como tal) son:

- Describir el funcionamiento del Departamento de Sistemas de Información de la ENEP Acatlán.
- Proporcionar, en el marco de la realización de *SIPRES*, una visión general acerca de Visual Objects (la plataforma de desarrollo de aplicaciones Windows con que está estructurado el sistema), de la programación orientada a eventos y de la programación orientada a objetos, las cuales son utilizadas al programar en dicha plataforma.
- Preparar el cimiento de futuros desarrollos de sistemas para ambiente Windows.

Para cubrir los objetivos planteados anteriormente, se dividió el contenido del estudio en cuatro capítulos, el primero de los cuales describe el funcionamiento del Departamento de Sistemas de Información de la ENEP Acatlán y la forma como surgió el proyecto de automatización del procedimiento de arqueo al fondo fijo del plantel.

En el segundo capítulo se explica en qué consiste el procedimiento de arqueo al fondo fijo de la Escuela y la necesidad de automatización de dicho procedimiento.

En el tercer capítulo se da una breve introducción a Visual Objects 1.0c, así como a conceptos básicos de programación orientada a eventos y de programación orientada a objetos.

En el cuarto capítulo se presenta el análisis, diseño y desarrollo del sistema, así como los conceptos básicos de Teoría de Sistemas utilizados para realizarlos.

Finalmente, se procede a plantear las conclusiones que la elaboración de la memoria inspiró al autor.

Se da paso ahora a la exposición del material que conforma este estudio, esperando que resulte de utilidad e interés para el lector.

1. EN BUSCA DE NUEVOS PROYECTOS.

El contenido de este capítulo describe el funcionamiento del Departamento de Sistemas de Información (DSI) de la ENEP Acatlán, la necesidad de cambiar el enfoque de sus proyectos, el aprendizaje de una nueva plataforma de desarrollo de aplicaciones y la manera como se gestó el sistema que a la postre dio origen a esta memoria.

1.1. EL DEPARTAMENTO DE SISTEMAS DE INFORMACIÓN.

El DSI es uno de los dos departamentos que componen al Centro de Cómputo de la escuela (el otro es el Departamento de Servicios de Cómputo), el cual está adscrito a la Coordinación de Servicios Académicos. Es fácil definir en pocas palabras cuál es la labor del departamento: desarrollar sistemas de cómputo encaminados a agilizar los procesos académico-administrativos de la institución y a proporcionar información segura, confiable y oportuna que

apoye a la toma de decisiones en cualquier nivel de la estructura de la ENEP; así como, administrar la red de área local de la escuela. El DSI está integrado por un jefe de departamento, líderes de proyecto, analistas, capturistas y prestadores de servicio social.

Cada líder de proyecto tiene a su cargo la dirección, de principio a fin, del desarrollo de un sistema. Para lograr lo anterior se auxilia de:

- Un grupo de analistas (el número de ellos es variable) quienes se encargan de desarrollar partes de los módulos del sistema.

- Personal de apoyo que realiza la documentación de usuario y, cuando el sistema ya está funcionando, se encarga de dar soporte técnico a los usuarios.

- Eventualmente, de algún prestador de servicio social que realiza labores diversas, ayudando a los demás integrantes del equipo de trabajo en lo que sea necesario.

El único líder de proyecto que no desarrolla sistemas es el administrador de la red de área local, pero el sistema de trabajo y la estructura de su equipo se adaptan al esquema descrito en el párrafo anterior.

El líder de proyecto debe entregar informes periódicos al jefe del departamento en los que describe los avances del sistema y los detalles que han surgido durante su desarrollo. Por su parte, el jefe da el visto bueno y, en su caso, propone alternativas para mejorar la calidad del trabajo.

En lo que respecta a los proyectos que conduce el departamento, el número de ellos se ha incrementado considerablemente durante los últimos dos años y son ya muchos los órganos de la escuela que recurren al DSI requiriendo la automatización de algún proceso, siendo ésta la forma más común en que se originan los sistemas; no obstante, todavía son más aquellos que recurren a él motivados por una falla en su nodo de red. Es digno de destacar que existen sistemas que nacen por iniciativa propia de algunos de los integrantes del departamento, quienes después muestran los beneficios que aportaría su posible implantación al órgano para el que fueron diseñados, invirtiendo así el procedimiento usual del surgimiento de un proyecto.

Independientemente de la naturaleza del nacimiento del proyecto, su desarrollo sigue una secuencia de pasos que puede describirse como sigue:

- El jefe del DSI y el candidato a líder del proyecto se entrevistan con el usuario para conocer sus necesidades de información, de las cuales se obtendrá posteriormente el diseño del sistema.

- Una vez conocida la naturaleza del proyecto, éste se asigna formalmente a un líder y se designa a su equipo, utilizando criterios de perfil profesional y de disponibilidad.
- El equipo de trabajo realiza el análisis del sistema para comprender al máximo lo que se requiere hacer.

- El equipo entrega al jefe del departamento un estudio de factibilidad del proyecto, del cual dependerá su realización. En realidad es raro que algún proyecto sea desechado, puesto que el DSI tiene que responder a las necesidades de información de la escuela como mejor se pueda, influyendo, incluso, en la decisión de proporcionar equipo de cómputo (no necesariamente nuevo) al órgano en cuestión, o bien utilizando los propios recursos del departamento para que con ellos se pueda iniciar la puesta en marcha del sistema.

- Se establece un plan de actividades para llevar a cabo las fases de diseño, instrumentación y mantenimiento del sistema, tal como lo marca la metodología de sistemas que se imparte en las asignaturas que cubren esta área en la carrera de Matemáticas Aplicadas y Computación (MAC). Dichas fases, junto con la de análisis mencionada anteriormente, serán explicadas más a detalle dentro del marco del caso de estudio que ocupa esta memoria y que se describe en los capítulos posteriores.

Una vez conocida la forma de trabajo del DSI, se procederá a describir cómo es que se presentó la oportunidad de desarrollar el sistema que es materia de este estudio.

1.2. ES TIEMPO DE CAMBIAR.

La plataforma de desarrollo (lenguaje de programación) utilizada para realizar los primeros sistemas del DSI que se distinguieron por sus buenos resultados fue Clipper. Este es el caso de programas tales como CIESIS (Sistema de Administración de los Procesos de Inscripciones y Evaluaciones del Centro de Idiomas Extranjeros) y SISPA (Sistema de Planta Docente), por citar sólo algunos.

En el DSI se elaboraron varias librerías que podían ser utilizadas cada vez que se desarrollaba un nuevo proyecto. De este modo, las funciones para crear menús, pantallas de búsqueda, cuadros de diálogo, etc., se incorporaban a cada nuevo sistema, que generalmente se ajustaba al esquema "altas-bajas-cambios-consultas-reportes"; algunas veces más complicado que otras, pero, en esencia, el resultado era el mismo.

A principios de 1996 ascendió a veinte el número de proyectos desarrollados por el DSI, todos ellos, como ya se mencionó, de naturaleza muy similar. Se sentía ya la necesidad de un cambio, tanto de enfoque de los sistemas como de plataforma[1]. Lo primero pudo solventarse con la idea de incorporar a los programas modelos matemáticos orientados a apoyar la toma de decisiones (esto es factible, puesto que todos los líderes de proyecto y todos los analistas del DSI son egresados o cursan los semestres finales de MAC); lo segundo no resultaba tan simple.

La nueva plataforma tenía que ajustarse a los recursos de cómputo con los que en aquel entonces contaba la mayoría de los órganos de la escuela, y que se pueden resumir en equipos con procesador 286 y, en el mejor y no más frecuente de los casos, 386; además, tenía que ajustarse también a la capacidad del equipo asignado a los desarrolladores del DSI: equipos con procesador 486 con 4 MB en memoria RAM. Por otro lado, se había generado una gran cantidad de información por medio de los programas en Clipper almacenada en archivos .DBF con índices .NTX, imposible de abandonar, reutilizable por nuevos desarrollos a tal grado que la nueva plataforma debería ser capaz de manejar dicha información de una manera eficaz. En suma, eran tantas las cualidades que se exigían del nuevo lenguaje que parecía muy difícil poder encontrarlo.

Sin embargo, ocurrieron algunos sucesos que hicieron las circunstancias más favorables. Computer Associates creó Visual Objects (VO), una herramienta de desarrollo de aplicaciones para ambiente Windows totalmente compatible con Clipper y, por tanto, con sus bases de datos e índices.

[1] En ese tiempo empezaba a generalizarse el uso de lenguajes visuales de desarrollo de aplicaciones Windows, tales como Visual Basic, Delphi y Java, mientras que en el DSI se seguía desarrollando bajo ambiente DOS.

Parecía tan natural, prácticamente obligado el cambio hacia VO, que fue adquirido por el departamento a pesar de saber que la creación de aplicaciones bajo esta plataforma iba a producirse en condiciones complicadas debido a la capacidad con la que contaban sus equipos, y que su instalación en las máquinas de los usuarios con computadora 386 dependía de que fueran abastecidos de por lo menos dos MB de memoria RAM[1] (por supuesto que quedaban descartados los órganos con computadora 286, los cuales tendrían que seguir trabajando con programas en Clipper). Esta situación también habría de verse modificada poco después.

El departamento fue dotado con 5 equipos pentium y con 10 simms de 4 MB de memoria. Por otro lado, algunos órganos de la escuela recibieron computadoras con procesador 486 con 4 MB en RAM y muchas con procesador 286 fueron remplazadas por lo menos con una de procesador 386. Existían ya condiciones de desarrollo y había mercado para trabajar. Lo que hacía falta ahora era aprender a manejar la nueva herramienta: VO.

1.3. EL APRENDIZAJE DE VO.

VO era un producto nuevo. No había libros (ahora ya hay algunos) ni cursos, sólo se contaba con el manual de usuario; en estas condiciones se imponía la necesidad de un aprendizaje autodidacta (cabe señalar que se había adquirido apenas la versión llamada *pre-release*).

Se designó a uno de los integrantes del departamento para que se dedicara única y exclusivamente a estudiar la nueva plataforma, con el fin de que posteriormente impartiera un curso al resto. No hubo mucho éxito en este primer contacto con la herramienta debido a dos razones.

La primera razón fue el error de haber enfocado el estudio de VO casi exclusivamente a la migración hacia él de las aplicaciones Clipper hechas por el DSI; debido al gran tamaño de estos programas y a la poca experiencia en VO, no se pudo avanzar demasiado en este aspecto[2].

[1] Las aplicaciones que se ejecutan bajo el ambiente Windows utilizan la memoria extendida de la computadora, es decir, aquella que se encuentra por encima de 1 MB en equipos con procesador 286 o superior. Para este tipo de aplicaciones, mientras mayor sea la memoria extendida, más rápida será su ejecución.

[2] A pesar de lo que pudiera pensarse, es mucho mejor aprender a utilizar VO enfrentándose a su estilo de programación, que pretendiendo migrar las aplicaciones Clipper hacia él.

La segunda razón fue el hecho de que la persona asignada al estudio del nuevo lenguaje tuvo que dejar esta tarea para desarrollar (en Clipper, obviamente) un sistema que tenía prioridad absoluta[1].

Lo poco que se pudo practicar con lo que realmente constituía el lenguaje de programación de VO sirvió para detectar serias deficiencias en su forma de compilar y correr las aplicaciones, además de ciertos detalles en el manejo de los eventos, los cuales parecían poderse estructurar de una manera más eficaz.

Debido a la carga de trabajo existente en el departamento, pasó un poco de tiempo antes de que se volviera a intentar trabajar con VO. Llegó al DSI la versión 1.0 y prácticamente nadie la tomó en cuenta, a pesar de que algunos desarrolladores la instalaron. Posteriormente, Computer Associates colocó en su página de Internet una actualización de VO, la versión 1.0c, disponible para todo público. Después de obtener la actualización, se decidió que era el momento de probar suerte de nuevo.

Esta vez la estrategia fue distinta. Los capítulos de los seis tomos del manual de usuario se dividieron entre todos los miembros del departamento con el objeto de estudiarlos a fondo y después presentar una exposición al respecto. Como es obvio, en muchas ocasiones fue necesario estudiar capítulos adicionales para poder comprender los propios, lo que proporcionaba una mejor visión del contenido del manual. Hubo algunos desarrolladores que comenzaron a sobresalir en el manejo de la herramienta y que aportaban experiencias valiosas durante las exposiciones. Puede decirse que las exposiciones fueron un éxito: la gran mayoría de los desarrolladores del departamento manejaban más que aceptablemente los editores visuales que proporciona VO para crear menús, ventanas, controles, iconos y servidores de bases de datos.

Sin embargo, el conocimiento adquirido tenía ciertas carencias; no bastaba saber hacer una ventana con botones y un gran menú con opciones que no realizaban ninguna tarea útil más que desplegar mensajes. La experiencia de los programadores más avanzados en el conocimiento de VO decía que el intentar programar un proceso específico (por ejemplo un pequeño

[1] Se trataba de *PPESIS*, un sistema en el que se pudiera capturar el plan de estudios de la carrera de Periodismo y en el que se pudieran emitir en forma de reporte sus contenidos y su bibliografía; era de prioridad absoluta porque dicho plan estaba en proceso de actualización, y se hizo en Clipper porque en ese entonces la División de Humanidades sólo contaba con una máquina con procesador 286. En estas circunstancias, se tuvo que formar un nuevo equipo de desarrollo que se ocupara del proyecto y, debido a la carga de trabajo existente en el DSI, el encargado de la investigación en VO tuvo que abandonar esta labor para incorporarse al trabajo de *PPESIS*.

sistema de asistencias del personal) era lo que realmente forzaba a aprender los puntos finos del lenguaje que no se detallan ni siquiera en el manual de usuario. Era el momento de buscar nuevos proyectos, cuyos requerimientos obligaran al programador a explotar verdaderamente y en mayor medida los recursos de su nueva herramienta de desarrollo.

1.4. EL NUEVO PROYECTO.

A mediados del mes de septiembre de 1996, asistieron el jefe del DSI, una compañera de equipo y quien esto escribe a las instalaciones del Departamento de Presupuesto de la escuela para sostener una plática con el jefe de dicho departamento (C.P. Raúl García) en la que relataría sus necesidades de automatización de cierto proceso a cargo del cual se hallaba: el arqueo al fondo fijo de la ENEP.

La descripción del proceso de arqueo al fondo fijo de la escuela es materia del siguiente capítulo.

2. EL ARQUEO AL FONDO FIJO DE LA ENEP ACATLÁN.

En el presente capítulo se plantea el caso de estudio que ocupa esta memoria, siendo necesario para ello exponer algunos conceptos básicos de contabilidad, así como la forma particular en que estos se aplican en el procedimiento de arqueo al fondo fijo de la ENEP Acatlán.

2.1. INTRODUCCIÓN.

El fondo fijo es una cantidad de dinero que el Patronato Universitario[1] asigna a la escuela en una cuenta bancaria para solventar sus gastos menores, tales

[1] El Patronato Universitario es una de las seis autoridades universitarias(las otras cinco son la Junta de Gobierno, el Consejo Universitario, el rector, los directores de facultades, escuelas e institutos y los consejos técnicos de facultades y escuelas); sus funciones principales son administrar el patrimonio universitario y sus recursos ordinarios y extraordinarios, gestionar el mayor incremento de dicho patrimonio y entregar al Consejo Universitario informes sobre las finanzas de la Universidad. *Legislación Laboral Universitaria (1992; pp. 33-36).*

como pagos a proveedores, acreedores y rembolsos a la dependencia por erogaciones cuya justificación lo amerite. Cada vez que se dispone de una cantidad del fondo fijo, se debe realizar un trámite que justifique ante el Patronato el desembolso de dicha suma y éste a su vez debe reincorporarla al banco por medio de un cheque.

El Departamento de Presupuesto es el órgano de la ENEP encargado de manejar el fondo fijo, e informa periódicamente a la Secretaría Administrativa[1] (de la cual depende) el estado de esa suma por medio de un documento llamado arqueo al fondo fijo.

El objetivo del Departamento de Presupuesto es realizar el control y el registro oportuno de los recursos financieros asignados a la ENEP, además de vigilar que su distribución y uso esté acorde con las políticas institucionales y de la propia escuela.

Además de encargarse del fondo fijo, dicho departamento realiza las siguientes funciones:

- Controlar el ejercicio del presupuesto asignado a la escuela, a través de un sistema contable de registro.
- Establecer los mecanismos de control para los Ingresos Extraordinarios, en apego a las políticas dictadas por la Dirección de la escuela y en cumplimiento estricto a la reglamentación vigente.
- Elaborar el Anteproyecto Anual de Presupuesto de la ENEP Acatlán, en colaboración con los órganos responsables, acorde a las disposiciones dictadas por la administración central y la Dirección de la propia escuela, utilizando la técnica de Presupuesto por Programas.
- Asesorar a los diferentes órganos de la escuela, en la aplicación del ejercicio presupuestal y en la formulación del Anteproyecto del Presupuesto de cada área.
- Conciliar periódicamente los registros presupuestales de la escuela ante las instancias normativas de la U.N.A.M.

[1] La Secretaría Administrativa es el órgano encargado de planear, dirigir, coordinar, supervisar y controlar los servicios y recursos humanos, materiales y financieros en que se apoyan las funciones académicas y de extensión, coadyuvando al logro de los objetivos sustantivos de la escuela. *ENEP Acatlán: Manual de Organización 1995 (1995; pp. 307-310).*

- Coordinar y vigilar la recepción de los pagos que la comunidad efectúa para la obtención de los servicios que así lo requieren, estableciendo los controles y reportes correspondientes.

- Establecer los mecanismos, registros y controles para que el pago de nómina al personal se realice en forma oportuna.

- Supervisar la recepción, pago y devolución de nóminas del personal, estableciendo los controles necesarios, en atención a las disposiciones del Patronato Universitario al respecto.

- Vigilar, controlar y reportar periódicamente respecto al ejercicio presupuestal de cada órgano responsable, en apoyo a la toma de decisiones en este renglón.

- Prever y proponer a la Unidad de Administración y Recursos, las modificaciones al presupuesto que se requieran durante el ejercicio, buscando el cumplimiento de las metas y objetivos previstos.

- Mantener actualizados los controles y registros de los servicios y apoyos administrativos, aplicando en su caso, modelos y sistemas automatizados[1].

Fue precisamente el arqueo al fondo fijo el proceso que el Departamento de Presupuesto solicitó al personal del DSI que le automatizara. Para comprender mejor cómo se realiza esta tarea, el equipo de desarrollo del proyecto tuvo que adquirir nociones básicas de contabilidad y comprender la forma particular en que se aplican en la ENEP. Dichas nociones se presentan en la siguiente sección.

2.2. CONCEPTOS BÁSICOS DE CONTABILIDAD[2].

2.2.1. CONCEPTO DE CUENTA.

Supóngase que un organismo pudiese agrupar por un lado todos los valores y documentos que constituyen un saldo a favor y por otro todos los que constituyen un saldo en contra; en contabilidad, el primer grupo se denomina

[1] *ENEP Acatlán: Manual de Organización 1995 (1995; pp. 349-352).*

[2] La contabilidad puede definirse como la disciplina que coordina y dispone en libros adecuados las anotaciones de las operaciones efectuadas por un organismo, con el objeto de poder conocer la situación de dicho organismo, determinar los resultados obtenidos y explicar las causas que han producido estos resultados. *Boter (1962; p. 7).*

activo, el segundo pasivo, y se dice que el capital de la empresa es igual al activo menos el pasivo.

La clasificación de las cuentas en activo y pasivo es muy parca para fines contables; es necesario ordenar y categorizar también los elementos que conforman tanto el pasivo como el activo. Por ejemplo, un negocio podría descomponer su activo en dinero en efectivo, mercancía y documentos que representan cantidades que le deben, mientras que su pasivo podría descomponerse en documentos de diferente naturaleza que representen las cantidades que debe. A cada una de esas categorías o subclasificaciones se les puede identificar por medio de un concepto que represente una abstracción de los respectivos valores que constituyen el capital, y, en contabilidad, a cada uno de esos conceptos se le llama cuenta.

En el caso de la ENEP Acatlán, el capital es el ya mencionado fondo fijo. El activo se representa por medio de las cuentas de efectivo, de deudores diversos y de documentos, mientras que el pasivo está conformado por las cuentas de fondo fijo, de proveedores, de acreedores diversos, de ingresos extraordinarios y de cuentas a reserva de comprobar, cada una de las cuales es descrita a continuación.

- *Efectivo*. Es el dinero disponible por la ENEP en sus diferentes cuentas bancarias, en cheques y en contrarrecibos.

- *Deudores diversos*. Los deudores diversos son órganos de la escuela que tienen asignada una parte del fondo fijo, llamada subfondo o fondo revolvente, para solventar sus propios gastos. El Departamento de Presupuesto asigna este subfondo por medio de pólizas que disminuyen la cantidad en bancos y aumentan la cuenta del deudor diverso en cuestión.

- *Documentos*. Justifican el empleo de cantidades pertenecientes al fondo fijo. Los datos que contiene cada documento son: nombre de la persona o empresa a favor de la cual se realiza el documento, concepto, importe, fecha de expedición y número de folio que lo identifica. El destino final de los documentos es un cheque emitido por el Patronato que reintegrará el importe al banco. Antes de que un documento genere un cheque debe pasar por los siguientes estados:

    - *Original*. Se dice que un documento es un original cuando es recibido por el departamento para que realice su trámite.

    - *Por tramitar*. En este estado se encuentran aquellos documentos que el Departamento de Presupuesto remitió a la Secretaría Administrativa

y a la Dirección para que avalen con su firma la continuación del trámite.

- *En trámite*. Una vez obtenida la firma del Secretario Administrativo y del Director de la escuela, el departamento envía el documento a la Unidad de Proceso, dependencia del Patronato, para que ésta proceda a la emisión del cheque[1].

Cada documento tiene asociada una procedencia que representa una cuenta del pasivo:

- *Fondo fijo*. Un documento con esta procedencia indica que se dispuso de una cantidad del fondo fijo (representada por el importe) para pagar a deudores diversos o bien a personal de la ENEP por realizar alguna tarea académica o administrativa que le acarreó desembolsos que la escuela debe pagar. Cuando dicho deudor o dicho personal presenta sus justificantes de gastos ante el Departamento de Presupuesto, éste realiza el trámite necesario para que el Patronato emita un cheque a favor de la ENEP y se reintegre al banco la cantidad correspondiente al importe del documento y con ello termine la gestión del mismo.

- *Proveedores*. Un proveedor es una persona o una empresa que proporciona un servicio o un bien a la escuela. Cuando la procedencia del documento es un proveedor, el departamento lleva a cabo el trámite para que le sea pagado a tiempo el dinero que se le debe. En el momento en que el documento llega al Patronato, éste se encarga de emitir un contrarrecibo que, por medio del Departamento de Presupuesto, le será entregado al proveedor para que pueda cobrar su cheque; en el momento de entregar el contrarrecibo termina para el departamento la gestión de dicho documento (como puede apreciarse, en este caso no se dispuso de capital del fondo fijo).

- *Acreedores diversos*. Los acreedores diversos son aquellos a quienes la ENEP les debe por conceptos diferentes a los de un proveedor. Para esta procedencia se realiza un procedimiento similar al de proveedores, con la diferencia de que es el Departamento de Presupuesto el que debe pagar directamente al acreedor. De este modo, cuando el documento llega al Patronato éste emite un cheque que se deposita en el banco para que después el departamento,

[1] Un estado adicional por el que no necesariamente tiene que pasar un documento es el de *rechazo*, el cual se produce sólo si el documento no está correctamente elaborado; en este caso el Patronato lo devuelve al Departamento de Presupuesto y el documento regresa al estado en trámite.

mediante una póliza, retire la cantidad correspondiente para pagar al acreedor, terminando así la gestión del documento (nótese cómo esta procedencia incrementa momentáneamente la cantidad en bancos).

- *Ingresos extraordinarios*. Cuando el documento procede de una actividad relativa a la captación de este tipo de ingresos (por ejemplo devoluciones por concepto de cursos cancelados), su trámite llega hasta un cheque emitido por el Patronato que deposita el importe correspondiente en el banco y posteriormente el Departamento de Presupuesto elabora otro cheque, esta vez a favor de la Tesorería de la UNAM.

- *Cuentas a reserva de comprobar*. Las cuentas a reserva de comprobar implican un doble proceso. Primero se genera un documento con procedencia cuentas a reserva de comprobar, que dará por resultado un depósito al banco. Posteriormente, se genera un documento con procedencia fondo fijo por el mismo importe que el anterior, el cual llegará hasta el estado de contrarrecibo. Cuando el Departamento de Presupuesto entrega el contrarrecibo a la persona o empresa que generó la cuenta, se termina la gestión de ambos documentos.

La idea de asociar cada documento con una procedencia se debe a que el Departamento de Presupuesto registra los movimientos del capital de acuerdo a la teoría contable de la partida doble.

2.2.2. LA PARTIDA DOBLE.

Las operaciones que un organismo realiza ocasionan la entrada y salida de valores de todo tipo; cada valor que entra debe registrarse en la cuenta que corresponde, mientras que cada valor que sale debe ser deducido de la cuenta respectiva. La partida doble es una teoría que considera que a todo cargo corresponde un abono, es decir, si una cuenta recibe un valor de los que a ella corresponden (cuenta deudora) entonces debe existir otra que entrega uno de los valores de su tipo (cuenta acreedora).

Para ejemplificar lo anterior las siguientes figuras muestran el caso de un original manejado por el Departamento de Presupuesto de la ENEP con procedencia proveedores e importe de $100.00.

ORIGINAL		P. TRAMITAR		EN TRÁMITE		CONTRARRECIBO	
CARGO	ABONO	CARGO	ABONO	CARGO	ABONO	CARGO	ABONO
100	0	0	0	0	0	0	0

PROVEEDORES	
CARGO	ABONO
0	100

Figura 2.1. El documento está en estado de original; la cuenta de original es deudora y la de proveedores es acreedora.

ORIGINAL		P. TRAMITAR		EN TRÁMITE		CONTRARRECIBO	
CARGO	ABONO	CARGO	ABONO	CARGO	ABONO	CARGO	ABONO
100	100	100	0	0	0	0	0
=							

PROVEEDORES	
CARGO	ABONO
0	100

Figura 2.2. El documento está en estado de por tramitar; la cuenta de original está saldada, la de por tramitar es deudora y la de proveedores sigue siendo acreedora.

ORIGINAL		P. TRAMITAR		EN TRÁMITE		CONTRARRECIBO	
CARGO	ABONO	CARGO	ABONO	CARGO	ABONO	CARGO	ABONO
100	100	100	100	100	0	0	0
=		=					

PROVEEDORES	
CARGO	ABONO
0	100

Figura 2.3. El documento está en estado de en trámite; la cuenta de por tramitar está saldada, la de en trámite es deudora y la de proveedores continúa como acreedora.

ORIGINAL		P. TRAMITAR		EN TRÁMITE		CONTRARRECIBO	
CARGO	ABONO	CARGO	ABONO	CARGO	ABONO	CARGO	ABONO
100	100	100	100	100	100	100	0
=		=		=			

PROVEEDORES	
CARGO	ABONO
0	100

Figura 2.4. El documento está en estado de contrarrecibo; la cuenta de en trámite está saldada, la de contrarrecibo es deudora y la de proveedores permanece como acreedora.

ORIGINAL		P. TRAMITAR		EN TRÁMITE		CONTRARRECIBO	
CARGO	ABONO	CARGO	ABONO	CARGO	ABONO	CARGO	ABONO
100	100	100	100	100	100	100	100
=		=		=		=	

PROVEEDORES	
CARGO	ABONO
100	100
=	

Figura 2.5. El contrarrecibo ha sido entregado al proveedor; la cuenta de contrarrecibo y la cuenta de proveedores están saldadas, y se termina la gestión del documento.

El ejemplo anterior permite observar cómo, al utilizar la teoría de la partida doble, en cada operación la suma de las cuentas deudoras debe ser igual a la suma de las cuentas acreedoras.

2.2.3. EL ESTADO DEL FONDO FIJO.

Resulta más que evidente la necesidad de los directivos de una organización de conocer el estado en el que se encuentra su capital. En el caso de la ENEP se ha mencionado ya que el Departamento de Presupuesto informa a la Secretaría Administrativa el estado del fondo fijo por medio de un arqueo (también llamado corte de valores), cuya estructura se muestra en la figura 2.6.

EFECTIVO		PROCEDENCIA DE LOS VALORES	
BANCOS	$	FONDO FIJO	$
CHEQUES	$		
CONTRARRECIBOS	$	INGRESOS EXTRAORDINARIOS	$
DOCUMENTOS		CUENTAS A RESERVA DE COMPROBAR	$
ORIGINALES	$		
POR TRAMITAR	$	ACREEDORES DIVERSOS	$
EN TRÁMITE	$		
EN RECHAZO	$	PROVEEDORES	$
DEUDORES DIVERSOS	$		
TOTAL DE VALORES	$	**TOTAL DE PROCEDENCIAS**	$

Figura 2.6. El arqueo al fondo fijo. Dado que se utiliza la teoría de la partida doble, el total de valores debe ser igual al total de procedencias.

En la siguiente sección se explican las tareas que lleva a cabo el Departamento de Presupuesto para elaborar el arqueo al fondo fijo.

2.3. DESCRIPCIÓN DEL PROCEDIMIENTO DE ARQUEO AL FONDO FIJO DE LA ENEP ACATLÁN.

2.3.1. ACTIVIDADES.

El Departamento de Presupuesto recibe diariamente documentos relacionados con cuentas del activo para su trámite. Cada documento se registra en pólizas de contabilidad y es canalizado a la Secretaría Administrativa y a la Dirección para su firma y posteriormente a la Unidad de Proceso para que se emita el cheque o el contrarrecibo correspondiente. Cabe señalar que el Departamento de Presupuesto guarda una relación detallada de los documentos que se encuentran fuera de sus instalaciones, siendo ésta la segunda redacción de los datos de cada documento (la primera fue el registro en libros de contabilidad).

Los importes de los documentos pertenecientes a la cuenta de deudores diversos se envían al Área de Contabilidad del mismo departamento para que se registren en un programa contable realizado por la Dirección General de Servicios de Cómputo Académico. Dicho programa, entre otras tareas, actualiza los saldos de cada órgano o individuo catalogado como deudor diverso (única parte automatizada del procedimiento de arqueo antes de *SIPRES*).

En lo que se refiere a los cheques y contrarrecibos que recibe el Departamento de Presupuesto del Patronato, estos son asociados, por medio del folio, al documento que les dio origen.

Cuando la Secretaría Administrativa solicita el informe de la situación financiera de la escuela al Departamento de Presupuesto (esto ocurre generalmente al inicio de cada mes) éste tiene que preparar el arqueo al fondo fijo actualizando el estado de las cuentas del pasivo y del activo con respecto al arqueo anterior. Las actividades que involucra la realización del arqueo son:

- Determinar el saldo en bancos considerando los cheques emitidos por el departamento y los cheques y contrarrecibos que recibió del Patronato desde el último arqueo así como el estado de la cuenta de bancos en el corte de valores anterior.

- Determinar el saldo en la cuenta de deudores diversos, para lo cual se apoya en un informe elaborado por el Área de Contabilidad, pues, como ya se dijo, éste tiene capturada la información por medio de un programa.

- Analizar la información de los documentos que se están gestionando al momento del arqueo sin importar su estado (original, por tramitar, en trámite

o en rechazo) clasificándolos por procedencia (fondo fijo, proveedores, acreedores diversos, ingresos extraordinarios o cuentas a reserva de comprobar) y obteniendo los saldos respectivos para cada cuenta.

- Elaborar el arqueo con la información clasificada y los saldos actualizados, incluyendo un resumen que tiene una estructura como la que se indica en la figura 2.6 -que constituye la parte principal del escrito-, además de una página por cada cuenta. Las cuentas del activo contienen los datos de los documentos que las originaron, incluyendo las de cheques y contrarrecibos (como puede apreciarse, ésta es la tercera redacción de la misma información), mientras que las cuentas del pasivo contienen listados con los saldos de órganos, empresas e individuos.

2.3.2. OBSERVACIONES.
Después de conocer el procedimiento de arqueo utilizado por el Departamento de Presupuesto, se identificaron los siguientes aspectos:

- Los datos de cada documento original eran redactados hasta en tres ocasiones.

- La actualización de los saldos de cada cuenta se realizaba por medio de calculadora, salvo la de deudores diversos.

- A diferencia del caso de deudores diversos, no existía un catálogo de proveedores con saldos actualizados.

- Se utilizaba mecanografía para redactar el arqueo con las desventajas que esto ocasionaba (posibles errores de escritura, almacenamiento de un gran volumen de papel, necesidad de escribir de nuevo todo el documento en caso de que se requiriera un nuevo original, etc.).

- El estado del fondo fijo se actualizaba una vez al mes, con cada arqueo, pues con el procedimiento ya descrito en el apartado anterior resultaba impráctico mantener la información al día (se hubiese tenido que redactar un arqueo diariamente).

Se había identificado ya un problema y se tenía también una alternativa de solución: la automatización del procedimiento de arqueo.

2.4. ¿CUÁL ES EL PROBLEMA?

El problema que se intentó resolver en este estudio se formuló de la siguiente manera:

¿Cómo puede estructurarse el procedimiento de arqueo al fondo fijo de la ENEP Acatlán para que, de manera eficaz, se pueda conocer en cualquier momento el estado actual del fondo fijo de la escuela?

La solución propuesta fue el desarrollo de *SIPRES*, el sistema automatizado de arqueo al fondo fijo, el cual tenía por objetivos:

- Agilizar los procesos de registro de la información relacionada con la cuenta de documentos del activo.
- Mantener actualizado el estado del fondo fijo de la escuela con base en una información confiable, segura y oportuna.
- Emitir de manera automatizada el arqueo al fondo fijo.
- Mantener un catálogo de deudores diversos y de proveedores con saldos actualizados.
- Permitir, aprovechando la red de área local con que cuenta la escuela, la consulta simultánea a la información del sistema a órganos tales como la Dirección, la Secretaría Administrativa y cualquiera que la Dirección así disponga sin necesidad de solicitar dicha información por escrito al Departamento de Presupuesto.

En el siguiente capítulo se describen las herramientas de programación utilizadas para el desarrollo de *SIPRES*.

3. HERRAMIENTAS DE PROGRAMACIÓN.

Este capítulo introduce al lector a tres temas principales: Visual Objects, la programación orientada a eventos y la programación orientada a objetos que, en conjunto, constituyen las herramientas de desarrollo con las que se creó *SIPRES*.

3.1. LA PLATAFORMA: VISUAL OBJECTS 1.0C.

VO es un sistema de desarrollo orientado a objetos que, basándose en Clipper, produce aplicaciones de bases de datos que pueden manejarse bajo el sistema operativo Windows de Microsoft.

En los siguientes apartados se describirán las características más importantes de VO. Es inevitable mencionar en esta sección conceptos que

pertenecen a la terminología de la Programación Orientada a Objetos (OOP[1]) tales como clase, método y objeto, pero su explicación se dará detalladamente en la sección 3.3.

3.1.1. EL ENTORNO DE DESARROLLO INTEGRADO (IDE) DE VO.

EL IDE es un ambiente gráfico para la creación de aplicaciones y librerías, y está compuesto por vistas (browsers) de componentes de programa[2], editores visuales, un editor de código, un compilador y un depurador. Siendo el propio VO una aplicación Windows, su IDE permite al programador utilizar todas las comodidades que presenta este sistema operativo, tales como la apertura de varias ventanas con diferente información, los menús con barra de herramientas y teclas aceleradoras y el uso del ratón, entre otras.

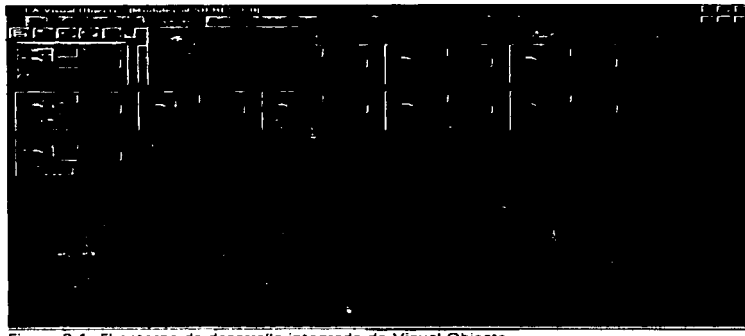

Figura 3.1. El entorno de desarrollo integrado de Visual Objects.

[1] A lo largo de este texto los acrónimos responden a las siglas en inglés del término que se esté manejando, pero su descripción se presenta en español.

[2] VO tiene una jerarquía de tres niveles de componentes de programa. El nivel más alto es el de las aplicaciones (el equivalente a programas y librerías en Clipper). Cada aplicación contiene varios módulos (igual que en Clipper), los cuales constituyen el segundo nivel de la jerarquía. Finalmente, cada módulo está compuesto de entidades, las cuales pueden ser constantes, funciones, clases, métodos, así como ventanas, menús, iconos y reportes creados mediante los editores visuales del IDE.

Mediante el IDE se puede crear una aplicación, compilarla, ejecutarla y depurarla sin necesidad de cambiar de ambiente. Esta característica debe ser muy bien recibida por los programadores de Clipper, recordando que para realizar estas mismas actividades sobre un programa en este lenguaje es necesario interactuar permanentemente con un editor de texto y con el sistema operativo.

Los componentes de programa son almacenados en una base de datos, llamada repositorio, la cual conoce qué componentes necesita cada aplicación y las interdependencias entre ellos. Existen varias formas (vistas) de presentar la información de los componentes de programa en la pantalla: por aplicaciones, por módulos, por entidades, por clases y por errores de compilación.

Los editores visuales de VO se utilizan para generar especificaciones de campo, servidores de bases de datos, ventanas, menús, iconos y reportes (los cuales también son entidades de un módulo) de una manera muy simple para el programador, quien no tiene que escribir código, sino dibujar y llenar la información que el editor le va solicitando; el código lo genera VO automáticamente.

El editor de código fuente permite al programador personalizar los resultados obtenidos con los editores visuales o bien escribir sus propias rutinas. Este editor es capaz de dividir los módulos en entidades y de escribir en un color diferente las palabras reservadas del lenguaje y los comentarios del programa.

3.1.2. AMBIENTE DE TIEMPO DE EJECUCIÓN DE VO.

Los lenguajes de programación ejecutan sus aplicaciones en uno de estos estilos:

- *Interpretado*. El sistema de tiempo de ejecución traduce línea por línea el código fuente a código de máquina; una línea de código es analizada tantas veces como se ejecuta. El intérprete no conoce los tipos de variables con los que trabaja ni su localización en memoria. De este modo, para realizar una simple operación aritmética se requiere que el intérprete localice las variables involucradas, defina sus tipos, determine las operaciones a realizar y asigne la memoria para el resultado[1].

[1] Dentro del estilo interpretado se encuentran los seudocompiladores, los cuales transforman el código fuente a alguna forma intermedia simbólica llamada seudocódigo que posteriormente es traducida por un intérprete, línea por línea como ya se mencionó. Clipper utiliza esta técnica, y su seudocódigo se llama P-Code.

- *Compilado*. Un compilador traduce todo el código fuente a código de máquina una sola vez. Para poder trabajar de este modo, las variables y funciones que utiliza el programa deben tener declarados sus tipos para que el compilador los reconozca y sepa cuánto espacio de memoria debe reservarles.

De lo expuesto anteriormente, resulta claro por qué los compiladores producen ejecutables más rápidos que los intérpretes, aunque también se puede apreciar que los lenguajes compilados son menos flexibles con el programador que los interpretados.

VO ejecuta sus aplicaciones en el modo compilado. Sin embargo, su compilador es capaz de averiguar el tipo de una variable no declarada basándose en su uso, lo cual dota al desarrollador de una cierta flexibilidad al programar. Esta característica del compilador en VO es muy útil cuando se migran programas de Clipper hacia él[1], pero no es recomendable hacer uso de ella en programas escritos puramente en el lenguaje de VO.

Debido a que el repositorio de VO conoce perfectamente a cada elemento de programa y a sus interrelaciones, la compilación se realiza única y exclusivamente sobre aquellas entidades que se vieron afectadas por alguna modificación al programa, permitiendo que esta operación sea más eficaz.

3.1.3. LIBRERÍAS.

VO ofrece varias librerías que pueden ser incorporadas a las aplicaciones que se generan con él:

- Librería de Clases DBF y SQL. Permiten el manejo de bases de datos de formato DBF y SQL en el estilo orientado a objetos.

- Librería de Clases GUI. Esta librería proporciona una extensa gama de herramientas para programar la Interface Gráfica de Usuario (GUI), es decir, para generar ventanas, menús y controles (botones, listas, etc.) con el estilo de Windows.

- Librería de Clases de Reportes. Permite la conexión de una aplicación hecha en VO con CA-RET, un reporteador que incorpora VO para procesar documentos y permitir la impresión de reportes.

[1] Al utilizar la técnica de seudocompilación, Clipper permite al programador no declarar las variables que utiliza en su aplicación. Si VO no tuviera un compilador flexible en ese aspecto, al migrar una aplicación Clipper hacia él se produciría un error por cada variable no declarada.

- Librería de Clases de Sistema. Esta librería contiene elementos de los cuales se auxilian las clases DBF, SQL y GUI para trabajar.
- Librería de sistema. Se asocia automáticamente a toda aplicación. Contiene las funciones y comandos que soporta el lenguaje de programación de VO.

- Librería DBF. Contiene los comandos y funciones para el tradicional manejo de bases de datos de Clipper (Xbase). Su equivalente en el lenguaje puro de VO sería la Librería de Clases DBF.

- Librería terminal. Soporta las funciones tradicionales de entrada-salida de datos mediante los comandos @ ...SAY...GET de Clipper; la salida en pantalla se realiza en modo texto (25 x 80 caracteres). Su equivalente en modo gráfico sería la librería de Clases GUI.

- Librería Windows API. Contiene la definición de funciones de la Interfaz de Programación de Aplicaciones (API) de Windows. La API es una librería con cientos de funciones hechas en C que generan las ventanas en las que Windows presenta los resultados de una operación, que ejecutan las aplicaciones, que tratan cadenas o que identifican tipos de datos; en otras palabras, puede decirse que es el lenguaje de Windows[1].

3.1.4. CONTROLADORES ODBC.

ODBC son las iniciales de Conectividad Abierta de Bases de Datos, un sistema de Microsoft que se propone la estandarización del acceso a bases de datos por medio de una serie de funciones API desarrolladas en C. Lo anterior significa que una base de datos diseñada para trabajar con ODBC puede ser utilizada independientemente del lenguaje de programación que intenta explotarla.

ODBC está totalmente vinculado con SQL, puesto que la conexión con la base de datos en cuestión se hace a través de este lenguaje[2]. Esto conduce a

[1] Las funciones que conforman la API de Windows pueden dividirse en varias categorías, de acuerdo a su campo de acción: ventanas, mapas de bits, el portapapeles, colores, cursores, controles, menús, el ratón e impresoras, entre otras.

[2] Las bases de datos pueden ser secuenciales, navegacionales y relacionales. En las secuenciales, un caracter sigue a otro sin un orden impuesto. Las navegacionales representan información en términos de filas (registros) y columnas (campos); a las bases de datos navegacionales se les puede imponer un orden basado en uno o más campos y dos o más de ellas pueden relacionarse haciendo coincidir uno o más de sus campos. Las bases relacionales presentan la información en términos de grupos que cumplen la misma condición; para ello, se interroga a la base de datos (compuesta de varias tablas o ficheros) y se extraen aquellos datos que cumplen las condiciones de la pregunta. El Lenguaje de Interrogación Estructurado (SQL) manipula bases de este tipo, y es el que usa VO para manejar ficheros de Oracle, Informix, etc.

la filosofía cliente-servidor: se tiene una aplicación desarrollada con un lenguaje cualquiera que es cliente de otra aplicación que funge como servidora de bases de datos. La ventaja de trabajar de esta manera es que las operaciones del cliente y del servidor están independizadas, y cada una puede ejecutarse en máquinas distintas, aumentando así el rendimiento.

VO provee al desarrollador de controladores ODBC para las bases de datos más conocidas del mercado: Oracle, Informix SQLServer, Sybase y Paradox, entre otras. Pero los controladores son sólo el nexo, de manera que se necesita tener las bases de datos para poder trabajarlas; esto significa que se necesita comprar Oracle, Informix, etc., para poder utilizar sus bases desde una aplicación de VO.

Se ha mencionado que las aplicaciones que produce VO pueden manejarse bajo Windows. La siguiente sección tiene por objetivo proporcionar un panorama que permita comprender la naturaleza de este tipo de aplicaciones.

3.2. EL ESTILO DE PROGRAMACIÓN WINDOWS[1].

Los programas diseñados para trabajar en ambiente DOS comienzan generalmente por presentar un menú de opciones, y una vez elegida una de ellas, el usuario no puede regresar al menú hasta no haber cerrado dicha opción; en una pantalla de captura es el programa quien decide el orden en el que han de ser introducidos los datos. Por éstas y otras características se dice que este tipo de programas tienen el control de la secuencia de operaciones, limitando así el accionar del usuario.

En una aplicación Windows, por el contrario, es el usuario quien tiene el control de la secuencia de operaciones; es libre de activar la misma opción del

[1] Se proporcionan a continuación algunos datos sobre el desarrollo de sistemas basados en ventanas. Xerox creó en los años setenta un sistema operativo basado en ventanas, íconos y ratón, que es el antecesor del actual MS-Windows. Una versión especial de este sistema operativo fue incorporada a uno de los proyectos más ambiciosos de la compañía Apple en los años ochenta: la computadora Macintosh. Más tarde, Microsoft distribuyó su primera versión de Windows para las computadoras XT, muy poco funcional; la siguiente versión, llamada Windows 286, era ya capaz de ejecutar más de un programa a la vez; fue hasta la versión 3.0 que las compañías de software más conocidas comenzaron a desarrollar aplicaciones para ser manejadas bajo este ambiente. A partir de entonces han surgido las versiones 3.1, 3.11 (para trabajo en grupo), 95 (que pretende ser un sistema operativo independiente del DOS) y NT (sistema operativo para redes). *Hacia Visual Objects (1994; pp. 419-422).*

menú las veces que quiera, de tener abierta más de una opción; incluso, es libre de cambiar de aplicación sin tener que cerrar la que está activa (esta capacidad de mantener funcionando al mismo tiempo más de una aplicación se llama multitarea[1]). Se dice que un programa con estas características está conducido por eventos[2].

El hecho de sentir que tiene el control del programa, hace que la mayoría de los usuarios se sientan más cómodos utilizando una aplicación Windows que una para DOS. Otras características que generalmente le agradan al usuario de las aplicaciones Windows son:

- Los programas que operan bajo Windows son consistentes con su aspecto y su manejo, lo que evita que el usuario pase largos periodos de tiempo aprendiendo a utilizarlos. La presentación en ventanas[3] de los resultados de una acción que tienen siempre la misma forma y los menús que pueden manejarse con ratón o con teclado exactamente de la misma manera en todos los programas son ejemplos de esta consistencia.

- El ambiente es amistoso con el usuario, pues le proporciona una gran cantidad de ayuda visual para realizar sus operaciones.

- La presentación del programa es atrayente, ya que se trata de un ambiente gráfico[4], además de que se puede personalizar cambiando el color de las ventanas, así como el tipo y el tamaño de letras, entre otras cosas.

---

[1] En realidad, el procesador dedica pequeñas fracciones de tiempo de proceso a cada una de las tareas, pero debido a la rapidez con que trabaja da al usuario la impresión de que todas están funcionando a la vez. En otras palabras, se utiliza la técnica del tiempo compartido igual que se usa en una red de computadoras donde hay un solo procesador central con varias terminales. Las aplicaciones que se pueden manejar de esta forma deben ser capaces de cuidar que las demás que se encuentran activas al mismo tiempo no interfieran en su funcionamiento, además de que deben estar diseñadas para funcionar en la memoria extendida de la máquina (a esto se le llama funcionamiento en modo protegido; su contraparte, el modo real utilizado por las aplicaciones DOS, sólo es capaz de gestionar el primer MB de memoria RAM).

[2] Un evento es todo aquel suceso que puede ocurrir en una ventana. Son ejemplos de eventos un clic del ratón, la selección de una opción de menú, la activación de un botón, el redimensionamiento de una ventana y un mensaje.

[3] Una ventana puede definirse como una pantalla en la que se ejecuta una tarea. Las ventanas surgen de la necesidad de visualizar en el mismo monitor los resultados de más de una tarea (de lo contrario se tendría que contar con más de un monitor).

[4] En el modo gráfico la pantalla de la computadora se maneja en pixeles, no en caracteres como ocurre en el modo texto. A manera de ejemplo, supóngase que se quiere dibujar un círculo por medio de puntos: en modo texto, se dispone de un total de 80 x 25 celdas (caracteres) en cada una de las cuales se puede colocar un punto para formar la figura; en

- Las aplicaciones Windows pueden comunicarse entre sí para intercambiar información, lo cual se logra a través de herramientas como DDE[1] y OLE[2].

Una vez descritas las características básicas de una aplicación Windows, conviene mencionar ahora cómo es que el programador podrá dotar a sus programas con estas cualidades. Si se asimiló que las aplicaciones Windows están conducidas por eventos, no habrá mucho problema para comprender que el desarrollador debe adoptar una técnica de programación orientada a eventos.

Cuando un evento ocurre debe ser transformado en un mensaje para que pueda ser manejado por la aplicación; es tarea de Windows realizar esta transformación. De este modo, un programa conducido por eventos debe estar diseñado para recibir y procesar mensajes en cualquier momento; no se sabe qué mensajes se producirán ni en qué orden, puesto que el control lo tiene el usuario. Se debe responder a esos mensajes generando acciones que deben realizarse lo más rápidamente posible y terminarse sin dejar rastro.

Cuando un programa necesita que el usuario introduzca información por medio del teclado se dice que se encuentra en estado de espera (por ejemplo lectura de datos y elecciones de menú). Si el programa no está conducido por eventos, cada estado de espera le impone al usuario la tarea que debe ejecutar, y no le permite realizar otra diferente hasta que no cierra la actual; en otras palabras, los únicos eventos que reconoce el programa son los que están asociados con este estado de espera en particular.

En un programa conducido por eventos se puede entrar a un nuevo estado de espera sin haber terminado con la operación del estado de espera actual; la manera de programar esto no es tan complicada como en un principio pudiera parecer. La idea es tener un solo estado de espera capaz de conocer todas las opciones disponibles del programa en cualquier momento, los eventos asociados con cada opción y el código que se debe llamar para gestionar cada

---

modo gráfico, se dispone de 800 x 600 celdas (pixeles) para colocar un punto con el mismo fin. La diferencia entre las imágenes obtenidas en ambos casos es abismal, pues en el segundo existe menor distancia entre los puntos que forman la figura, dando así una apariencia de trazo continuo que no puede lograrse en el primero.

[1] DDE (Intercambio Dinámico de Datos) es una herramienta de Windows que permite el establecimiento de canales de comunicación entre las aplicaciones para que puedan intercambiar información.

[2] OLE (Enlace e Incorporación de Objetos) es una forma de compartir datos entre las aplicaciones Windows, y se basa en el concepto de documento compuesto, es decir, un documento que contiene datos de dos o más fuentes. Uno de los ejemplos más simples de esto es un escrito en Word que incluya una imagen de Paintbrush y una hoja de cálculo de Excel.

evento. Este estado de espera es un bucle (un ciclo) con la siguiente estructura:

```
Mientras...
        oEvento: = EsperaEvento()
        GestionaEvento(oEvento)
Fin.
```

Es claro lo que realiza la función EsperaEvento, mientras que GestionaEvento determina la parte de código que atenderá al evento ocurrido, el cual es enviado como parámetro. La aplicación es la encargada de informar al gestor qué eventos espera y cuál es el código que va a procesar a cada uno de ellos[1]. El gestor, por su parte, almacena la información en una lista que crece y decrece dependiendo de las necesidades de la aplicación en cada momento de su ejecución.

Para ejemplificar la gestión de eventos descrita anteriormente, supóngase un programa que inicia con una ventana que contiene el menú principal. El gestor es alimentado con los eventos de la ventana y del menú, así como con el código que los procesa y entonces se da inicio al bucle de eventos. Supóngase también que se elige una opción del menú principal; el gestor reconoce el evento y llama al código correspondiente, que a su vez genera por resultado una ventana con cierta información. Ahora la lista del gestor se ve incrementada con los eventos de la nueva ventana y su respectivo código, regresando el control al bucle de eventos, que, como puede observarse, se ha constituido en el único estado de espera de la aplicación. De este modo, es factible activar una nueva opción del menú principal sin tener que cerrar la ventana activa, puesto que es un evento incluido en la lista del gestor.

El no saber qué evento se puede producir fuerza a los gestores de eventos a ser independientes de su entorno; por el contrario, la interfaz de una aplicación (su presentación al usuario) conducida por eventos debe estar diseñada para ser sensible al contexto. Por ejemplo, si se abre una ventana sin cerrar la actual, los mensajes de la barra de estado, los indicadores de menú y los controles deben pertenecer a la nueva ventana, mientras que la anterior es desplazada a un segundo plano y debe esperar su turno para ser considerada.

[1] Desde que se crean las ventanas, menús y controles que conforman una aplicación, se define para ellos el código que procesará sus eventos, el cual se denomina procedimiento de ventana. La posición de este código es informada al gestor de eventos por medio de Windows para que recurra a él en caso de ser necesario. En VO el programador no tiene que preocuparse por hacer este código, puesto que se genera automáticamente por medio de los editores visuales de ventanas y menús.

Hasta aquí se ha presentado información acerca de cómo se manejan los eventos, pero no se ha mencionado algo acerca de cómo es que el programa se entera de que ha ocurrido uno de ellos. Windows es el encargado de comunicar a una aplicación la ocurrencia de eventos colocando mensajes en la cola de mensajes de dicha aplicación, la cual es mantenida por el propio Windows. A partir de aquí es responsabilidad del programa comprobar frecuentemente la existencia de eventos en la cola y procesarlos debidamente. Supóngase que un usuario ha cambiado el tamaño de la ventana actual; a medida que el usuario redimensiona la ventana, Windows envía mensajes a la aplicación informando del nuevo tamaño y las coordenadas de la ventana; el procedimiento de ventana permite que Windows cambie el tamaño de la ventana y una vez que éste envía el mensaje que indica el término de su tarea, dicho procedimiento actualiza la información presentada en la ventana acorde al nuevo tamaño[1].

Que las aplicaciones sean conducidas por eventos no es la única característica novedosa del ambiente de desarrollo Windows para aquellos que realizan sus primeras incursiones por él:

- Windows proporciona una gran cantidad de memoria a las aplicaciones, pues las hace funcionar, como ya se ha dicho, en modo protegido, permitiéndoles el acceso directo a 16 MB de memoria. Si no se tiene tal cantidad de memoria disponible en la máquina y la aplicación ha hecho que se agote, Windows activa un gestor de memoria virtual que intercambia código y datos de manera transparente para la aplicación desde y hacia el disco duro.

- Al desarrollar en Windows, el programador no es responsable de controlar las impresoras, ni la tarjeta de vídeo ni ninguno de los componentes de hardware de la máquina en la que va a ejecutarse su aplicación; Windows y sus controladores de dispositivos se encargan de ello.

- Los programas se comunican con Windows a través de librerías que deben enlazarse a las aplicaciones en tiempo de ejecución, por lo cual se denominan Librerías de Enlace Dinámico (DLL)[2]. Este enlace de las librerías con la aplicación ocurre cada vez que ésta se ejecuta; el código de la DLL es

[1] Los mensajes que envía Windows son ignorados por la aplicación hasta que aparece el mensaje indicando el fin del redimensionamiento; a este tipo de mensajes se les denomina mensajes por defecto y no son procesados por el procedimiento de ventana, sino por una función API llamada DefWindowProc().

[2] Las librerías de enlace estático (como las de Clipper) son vinculadas al programa una sola vez, cuando el enlazador crea el programa ejecutable y determina las dependencias entre la aplicación y ellas.

independiente del de la aplicación, de manera que el usuario requiere tanto del archivo ejecutable como de la DLL para poder utilizar el programa. Las funciones que la aplicación requiere de una DLL se almacenan en una sección del archivo .EXE, llamada lista de funciones importadas, que contiene una entrada para cada función y la referencia de la DLL que la contiene. Cuando la aplicación se ejecuta, las DLL requeridas son localizadas, inicializadas y ligadas a la misma. Las rutinas que conforman una DLL están enlazadas entre ellas mismas, de modo que si una función DLL llama a otra, el vínculo se hace desde que se crea la librería, no en tiempo de ejecución. Como comentario final acerca de las DLL, una de ellas puede ser utilizada por varias aplicaciones a la vez, pero Windows sólo carga una copia de ella.

Es común que las aplicaciones conducidas por eventos sean desarrolladas en un lenguaje orientado a objetos; ese es el caso de las que se generan por medio de VO. Para comprender la naturaleza de esta relación, en la siguiente sección se hará un recorrido por los conceptos básicos de la Programación Orientada a Objetos (OOP).

3.3. A CAMBIAR LA FORMA DE DESARROLLO: OOP[1].

3.3.1. INTRODUCCIÓN.

Las diferentes técnicas de programación a lo largo del tiempo han pretendido ser cada vez más semejantes a la forma de pensar del hombre; incluso, los lenguajes de programación están compuestos por estructuras similares a las del lenguaje hablado.

La manera de resolver un problema por medio de la programación estructurada, que es la de mejor aceptación por parte de los programadores, es dividirlo en partes suficientemente pequeñas como para ser instrumentadas cada una en un módulo de programa. Actualmente existen dos vertientes de programación estructurada ampliamente aceptadas: la programación procedural y la OOP.

En la programación procedural, la aplicación se divide en un conjunto de tareas cuya ejecución depende de funciones o subrutinas articuladas que toman y ceden el control según un esquema predefinido. La lógica de ejecución

[1] Los orígenes de la OOP se remontan a 1967 en Noruega, donde se desarrolló un lenguaje con las características de la orientación a objetos llamado Simula67. Posteriormente nació SmallTalk, el primer lenguaje totalmente orientado a objetos, creado en el Centro de Investigación de Palo Alto de Xerox. *Hacia Visual Objects (1994; p. 604).*

del programa es lineal, es decir el orden de ejecución es el mismo en el que se encuentran escritas las instrucciones que lo conforman. Los datos de este tipo de aplicaciones no se encuentran en ningún orden y la única jerarquía que existe en ellas es la que describe el hecho de que las subrutinas son llamadas por otra, considerada como principal. Este modo de organización de código es muy efectivo, pero dificulta el crecimiento posterior de las aplicaciones, ya que la red de interrelación entre las subrutinas puede ser tan densa como para complicar la identificación de la ruta necesaria para alterar un procedimiento.

Con la aparición de las interfaces gráficas de usuario se dio paso a las aplicaciones conducidas por eventos, en las cuales, como ya se explicó anteriormente, el programador no sabe con certeza el flujo de ejecución del programa, puesto que el control lo tiene el usuario. De este modo, la programación orientada a eventos impone nuevos retos, tanto al programador como a la programación basada en descomposición procedural.

La respuesta a los requerimientos de la programación conducida por eventos es la OOP, en la cual se descompone el sistema según los datos que gestiona. La OOP pretende ser el esquema de programación más próximo a la forma de pensar del hombre: su estructura de datos básica es el objeto, que queda definido por sus características (datos) y por su comportamiento (métodos). El mecanismo de ejecución de las aplicaciones de OOP es el envío de mensajes a los objetos, los cuales invocan al método asociado a dicho mensaje.

Después de esta breve introducción, en los próximos apartados se describen con mayor profundidad los conceptos fundamentales de la OOP.

3.3.2. ABSTRACCIÓN.

Al clasificar un ente se crea una abstracción de él que lo define completamente, pues se están identificando sus propiedades esenciales. En términos de OOP, esa abstracción recibe el nombre de clase y es la base del enfoque orientado a objetos. Recordando lo dicho acerca de la semejanza de la orientación a objetos con la forma de pensar del hombre, un ejemplo del concepto de clase puede ser algo con motor, cuatro llantas, volante, pedales y palancas que sirve para trasladarse de un lugar a otro; la clase aludida se denomina coche.

Una clase es el resultado de modelar las características (datos) y el comportamiento (acciones) del ente en cuestión. En OOP cada uno de los datos que una clase define se llama variable, y su comportamiento queda regido por fragmentos de código denominados métodos (cada uno de ellos corresponde a una acción). Volviendo al ejemplo de la clase coche, sus

variables serían motor, cuatro llantas, volante y pedales, entre otras; sus métodos serían arrancar, caminar y frenar, por citar algunos.

Tener en mente la abstracción de un coche no significa que con ella se pueda ir de un lugar a otro; se requiere la materialización de esa clase, es decir, se debe tener un vehículo para poder hacer funcionar sus partes y efectuar un traslado. La materialización de las clases en la OOP se denomina instanciación, y su resultado es un objeto (también llamado instancia) perteneciente a la clase. Así, un Shadow en el estacionamiento es una instancia de la clase coche.

Vale la pena abundar un poco más sobre los conceptos de clase y objeto. Un objeto es un componente de la aplicación que existe en tiempo de ejecución, y está integrado por datos y código, los cuales representan, respectivamente, a sus características y a su comportamiento[1]. Por su parte, una clase es únicamente código, una plantilla de la cual se obtienen objetos que comparten las mismas propiedades.

Las variables que define una clase pueden ser de instancia o de clase. Una variable de instancia es aquella de la cual cada objeto de la clase tiene una copia; una variable de clase tiene como característica que es compartida por todos los objetos instanciados de la clase.

Los mensajes son el mecanismo de ejecución de los lenguajes de OOP: un mensaje se envía a un objeto para indicarle cuál es el método que debe ejecutar. Los objetos tienen una identidad única, lo que permite que al mismo tiempo puedan existir varios objetos de la misma clase, los cuales tienen, cada uno, una copia de las variables de instancia definidas por la clase, pero comparten el código de los métodos. Entonces, al invocar un método, se debe hacer referencia explícita al objeto que debe ejecutarlo, es decir, debe enviársele un mensaje[2].

3.3.3 TIPOS DE MÉTODOS.

De acuerdo a la función que realizan, se pueden identificar los siguientes tipos de métodos:

- Constructor. Es un método que crea una nueva ocurrencia del objeto.

[1] El código del objeto no es otra cosa que los métodos definidos por la clase. Se puede hacer referencia a ambos, características y comportamiento, como las propiedades del objeto.

[2] Para el envío de mensajes, los lenguajes de OOP incorporan un operador llamado send; en VO este operador es el caracter : . Así, la instrucción oVentana:Mostrar() envía un mensaje al objeto oVentana para que ejecute su método Mostrar.

- Destructor. Elimina de la memoria un objeto cuando ha terminado su función[1].
- De acceso. Permite conocer el contenido de una variable de instancia o de clase.
- De asignación. Modifica el contenido de una variable de instancia o de clase.

3.3.4 ENCAPSULACIÓN.

La encapsulación es una característica de la OOP que impide que los usuarios de una clase vean su funcionamiento interno; ellos saben lo que puede hacer, pero no cómo lo hace[2]. La encapsulación también implica que una clase debe estar diseñada para contener todos los elementos necesarios para su funcionamiento, condición que le permite ser totalmente independiente de las demás clases del sistema.

La encapsulación permite a los desarrolladores cambiar el funcionamiento de una clase sin que lo sepan los usuarios, siempre y cuando no cambie su interfaz (la forma en que se debe hacer el llamado a la clase: nombre y parámetros); si los usuarios conocieran el interior de la clase y se apoyaran en él para programar, tendrían que cambiar sus códigos en caso de que el desarrollador hiciera un cambio en dicha clase.

El soporte de la encapsulación en OOP es la posibilidad de ocultar las variables de la clase. Para impedir que los usuarios lean o escriban en las variables de la clase, el desarrollador puede declarar sus variables como ocultas (hidden), las cuales sólo pueden ser consultadas y modificadas por métodos pertenecientes a la clase[3]. Si desea que los métodos de otras clases lean las variables ocultas, puede declarar un método llamado ACCESS que retorne como resultado su valor; si quiere que también los métodos de otras clases puedan escribir en ellas, debe declarar un método llamado ASSIGN que modifique su contenido[4].

---

[1] En VO no es necesario que el programador utilice un destructor para eliminar los objetos, pues VO incorpora una utilidad llamada recolector de basura que realiza esta función.

[2] En este contexto, usuario es el programador que incorpora una clase a su aplicación; puesto que ya sabe para qué sirve y que funciona correctamente, no debe importarle demasiado cómo está compuesta internamente.

[3] Las variables ocultas también reciben el nombre de variables no exportadas, mientras que las visibles se denominan exportadas (export) y pueden ser leídas y modificadas por métodos ajenos a su clase.

[4] La ventaja de utilizar una variable oculta a la que se le define un método access contra el uso de una visible es que en el primer caso se puede restringir, por medio del método access, el valor que será asignado a la variable, mientras que en el segundo la asignación es totalmente libre.

3.3.5. HERENCIA.

La herencia permite crear nuevas clases (llamadas hijas o subclases) a partir de las ya existentes (llamadas padres o superclases) sin duplicar código: la nueva clase hereda todas las propiedades del padre, y puede definir las que son específicas de ella. De esta manera se establece una jerarquía de clases en donde las de menor nivel heredan todas las propiedades de las de los niveles superiores [1]. En la herencia y en la encapsulación está fincada la gran cantidad de código reutilizable que produce el enfoque orientado a objetos y que constituye uno de sus principales atractivos con respecto a la programación procedural.

Las subclases pueden definir métodos con el mismo nombre que los de su superclase, y variables de instancia con el mismo nombre de las variables de instancia ocultas de su superclase; cuando se envía un mensaje a un método o se accede a una variable que ha sido redefinida por la subclase, se usa su versión en lugar de la de su padre.

En OOP existen dos tipos de herencia: la simple y la múltiple. En la herencia simple o directa la subclase tiene una sola superclase, mientras que en la múltiple se crea una nueva clase a partir de dos o más ya existentes. Son raros los lenguajes de OOP que soportan la herencia múltiple (VO, por ejemplo, no lo hace).

Las variables declaradas como ocultas por una clase no son visibles tampoco por sus subclases; para que una clase hija pueda leer y escribir en las variables de su superclase, pero que las clases que no pertenecen a la jerarquía no tengan ese derecho, dichas variables deben declararse como protegidas (protect).

La herencia no es el único tipo de relación que puede existir entre las clases. Cuando en Windows se muestran dos ventanas, cada una con un menú, y se le da el foco a una de ellas, se muestra el menú de la ventana activa; en este caso se dice que la clase ventana contiene a la clase menú. Por otro lado, esas mismas ventanas comparten el puntero del ratón, es decir, no hay un puntero para cada ventana; en este caso se dice que la clase ventana usa a la clase puntero.

---

[1] Cabe señalar que en OOP existen clases llamadas abstractas que se utilizan como base para la construcción de otras, pero que por sí mismas no son útiles. En otras palabras, no se instancian, sólo heredan sus propiedades a subclases de las que sí pueden obtenerse objetos.

3.3.6. POLIMORFISMO.

En OOP el polimorfismo significa que distintas clases pueden definir métodos con el mismo nombre, aunque no necesariamente dichos métodos realicen los mismos procesos. Por ejemplo, se puede definir un método llamado play para la clase grabadora y un método play para la clase CD; ambos hacen que se escuche algo, pero los procesos que involucran son diferentes.

En un ambiente de desarrollo de aplicaciones de OOP en grupo, el polimorfismo permite que cada programador pueda llamar como desee a sus métodos sin importar que los demás puedan utilizar nombres idénticos para los suyos. Gracias al polimorfismo es posible agregar nuevas clases a una aplicación con repercusiones prácticamente nulas para las clases existentes.

3.3.7. RELACIÓN ENTRE LA OOP Y LA PROGRAMACIÓN ORIENTADA A EVENTOS.

En la programación conducida por eventos se requiere que las aplicaciones realicen tareas breves que desaparezcan sin dejar rastro. La encapsulación de las clases generadas por la OOP produce entidades (objetos) que ejecutan su función y después pueden destruirse, puesto que son independientes unos de otros. Esta característica de la OOP es la que la hace compatible con la programación conducida por eventos, justificando así la gran cantidad de lenguajes orientados a eventos que la utilizan.

3.3.8. VENTAJAS Y DESVENTAJAS DE LA OOP.

Después de conocer los principios básicos de la OOP es posible percatarse de algunas ventajas y desventajas de este enfoque.

Entre las ventajas que ofrece la orientación a objetos se encuentran las siguientes:

- La propiedad que tienen los objetos de un sistema OOP de contener todo lo que necesitan para funcionar los hace independientes unos de otros; esto provoca que el programa pueda adaptarse fácilmente a los cambios, pues el agregar o quitar objetos afecta únicamente a pequeños módulos del mismo (a esto se le conoce como extensibilidad).

- Si el diseño de los sistemas se basa en las tareas que éste realiza (tal como ocurre con la programación procedural) y si dichas tareas cambian, lo hará también la estructura del programa. Si por el contrario el diseño está basado en los datos (como en la OOP) y se requiere que el sistema realice más o diferentes tareas, el impacto sobre la estructura del programa será mucho menor. Esto da por resultado una cantidad de código reutilizable mucho mayor que la que se obtiene con la programación procedural. Características

como la herencia y la encapsulación también contribuyen a la producción de código reutilizable.

Algunas de las desventajas que pueden observarse de la OOP son:

- Debido a la reticencia del ser humano a los cambios, la introducción a la OOP de los programadores acostumbrados a la técnica de desarrollo procedural resulta difícil.

- Cuando se crea una relación de herencia entre las clases, el compilador incluye en el archivo ejecutable tanto el código de la clase padre como el de la hija; en muchas ocasiones no se utilizan todas las propiedades del padre, lo que produce un incremento inútil en el tamaño del programa. Evidentemente, esta situación se agudiza mientras más extensa sea la jerarquía de clases.

Una vez cubierto el objetivo de profundizar brevemente en lo que fueron las herramientas de programación de *SIPRES*, se da paso a la descripción del proceso de análisis, diseño y desarrollo del sistema.

4. *SIPRES*, EL SISTEMA DE ARQUEO AL FONDO FIJO.

En este capítulo se describe la metodología de desarrollo de sistemas empleada para realizar el proyecto *SIPRES*, para lo cual se introducen a continuación algunos conceptos básicos de teoría de sistemas que permitirán identificar a *SIPRES* como un sistema de información.

4.1. CONCEPTOS BÁSICOS DE TEORÍA DE SISTEMAS.

Un sistema es un conjunto de elementos interrelacionados entre sí que persiguen un fin común. Todo aquello que rodea al sistema, es decir, lo que no pertenece a él, se denomina ambiente, y la frontera entre el sistema y su ambiente se conoce como límite del sistema[1]. Cuando un sistema, además de

[1] Como puede observarse, el concepto de sistema en ningún momento maneja la idea de computación ni de software; es el avance tecnológico el que ocasiona que hoy en día,

tener relaciones entre sus miembros tiene relaciones con elementos de su ambiente, se denomina abierto y a dichas relaciones se les conoce como las entradas y las salidas del sistema.

Como ejemplo de un sistema se puede mencionar una computadora cuyos elementos son el monitor, el teclado y el procesador central; el objetivo es procesar datos que el usuario de la máquina introduce por medio del teclado (entradas) y presentar la información procesada por medio del monitor (salida); los límites del sistema son físicos y su ambiente es todo aquello que no forma parte de la estructura de la computadora.

A pesar de que existen ejemplos de sistemas desde épocas remotas[1], fue hasta las décadas de 1930 y 1940 que nació una "teoría general de sistemas", de la cual gran parte se debe al trabajo de Ludwig Von Bertalanffy, a quien se le considera el padre de esta disciplina. Dicha teoría estudia los sistemas concibiéndolos en principio como una caja negra en la que sólo se pueden identificar las entradas y las salidas, pero no su interior; una vez que se distinguen los límites del sistema y la naturaleza de sus relaciones con el exterior, se procede a examinar el contenido de la caja. Se identifican entonces grandes subsistemas y las relaciones que existen entre ellos, considerándolos, a su vez, como cajas negras cuyo contenido se descubre al momento de continuar con este proceso de descomposición, el cual termina cuando los subsistemas son lo suficientemente simples como para ser estudiados con cierta facilidad.

Mientras más complejo sea un sistema se vuelve más importante la comunicación entre sus componentes, puesto que deben trabajar de manera coordinada para lograr el objetivo común. De este modo, la transmisión de datos y de información entre los elementos de un sistema se convierte en un aspecto muy importante de su funcionamiento.

Vale la pena definir los conceptos de dato e información, ya que frecuentemente y de manera equivocada se manejan como sinónimos. Un dato es un hecho, un acontecimiento, una transacción; la información es un conjunto de datos procesados de manera que sean útiles para quien los recibe.

En el ámbito de cualquier organismo, disponer de una información de calidad contribuye a una toma de decisiones con mejores elementos de juicio. Se dice que una información es de calidad si:

---

generalmente, sea mucho más práctico y eficaz un sistema automatizado que uno que no lo es.

[1] El sistema de correos de la cultura azteca es uno de ellos.

- Es relevante para la decisión que se desea tomar.
- Es precisa, es decir, la información coincide exactamente con la realidad.
- Es lo suficientemente completa en lo que se refiere a los elementos clave que influyen en la decisión.
- Se comunica a la persona adecuada.
- Es oportuna.
- Tiene el nivel de detalle apropiado.
- Es comprensible para quien la recibe.

El sistema que se encarga de recopilar (o en su caso producir) y distribuir la información necesaria para los procesos y la toma de decisiones de un organismo con las características de calidad ya mencionadas se denomina sistema de información.

Los elementos que componen un sistema de información son:

- Los procedimientos y las prácticas habituales de trabajo presentes al ejecutar cada una de las tareas involucradas en el funcionamiento del organismo, y que indican qué información se necesita, cuál es el papel de cada persona integrante del organismo y qué equipo se requiere para desarrollar las funciones de una manera eficaz.
- La información.
- Los usuarios.
- El equipo de soporte para la comunicación, el procesamiento y el almacenamiento de la información, que puede incluir desde máquinas de escribir, lápices y archiveros hasta computadoras y discos flexibles.

Como ejemplo de un sistema de información considérese el sistema de arqueo al fondo fijo de la ENEP Acatlán, cuyos procedimientos y prácticas de trabajo han sido dados a conocer en el capítulo 2. Los usuarios del sistema son el Departamento de Presupuesto y la Secretaría Administrativa y, dado que el sistema era manual hasta antes de *SIPRES*, el equipo de soporte para la comunicación, el procesamiento y el almacenamiento de la información consistía en máquinas de escribir, papel y archivos.

El proyecto de automatización del sistema de arqueo implicó cambios en los procedimientos y prácticas de trabajo y, evidentemente en el equipo de soporte. Por otro lado, dicha tarea de automatización requirió seguir una serie de pasos dictados por la teoría general de sistemas, el primero de los cuales fue el análisis del sistema que se expone en la siguiente sección.

4.2. EL ANÁLISIS.

El análisis de sistemas consiste en dividir al sistema en sus componentes para estudiarlos de forma aislada sin perder de vista sus relaciones con los demás. En realidad el análisis de sistemas incorpora en sí mismo un proceso de síntesis, pues es imposible entender un sistema si no se conoce cómo funcionan sus componentes en conjunto. En general, el análisis de sistemas produce un documento que indica lo que el futuro sistema debe hacer y cuáles son sus requisitos, pero no cómo lo hará.

En el DSI el análisis de sistemas involucra una serie de actividades que se describen en los siguientes apartados.

4.2.1. RECOPILACIÓN DE LA INFORMACIÓN.

El proyecto *SIPRES* surgió de la petición del Jefe del Departamento de Presupuesto de automatizar su procedimiento de arqueo al fondo fijo. La entrevista[1] fue el medio utilizado para recopilar la información del procedimiento de arqueo que ya fue descrita en el capítulo 2.

Una primera entrevista realizada por los miembros del DSI a los responsables de los órganos que necesitan un sistema automatizado debe obtener la siguiente información:

- Pasos que involucra el procedimiento a automatizar.
- Las necesidades de información del órgano (datos a capturar, tipos de consulta y reportes).
- La definición de conceptos que eliminen ambigüedades entre lo que entiende por un tópico el responsable del órgano y lo que entiende el desarrollador.
- Tareas específicas relacionadas con el procedimiento que realizan los integrantes del órgano.
- Órganos involucrados con el procedimiento y su participación en el mismo.
- Equipo de cómputo disponible (tipo de procesador, capacidad de memoria, velocidad del disco duro, tipo de impresora, tarjeta de red, nodo de red).
- Habilidad de los integrantes del órgano para manejar programas de computadora.

4.2.2. DISEÑO PROTOTIPO.

Con el material recopilado en la entrevista, el equipo de desarrollo construye un informe que presenta al Jefe del DSI y elabora también un primer diseño,

[1] Una entrevista es un medio para recoger información de otra persona por medio de una conversación interpersonal; es tarea del entrevistador diseñar la estructura de la conversación acorde a la información que desea obtener.

llamado prototipo, de la apariencia del sistema y de las funciones que realizará; este último se expone al futuro usuario para que verifique que el sistema se ajusta a sus necesidades y, en su caso, realice las observaciones pertinentes. Cabe aclarar que el prototipo permite estudiar la factibilidad del proyecto, lo cual, evidentemente, se realiza antes de presentar el diseño inicial al usuario.

4.2.3 ESTUDIO DE FACTIBILIDAD.

Ya se ha hablado acerca de cómo los sistemas solicitados al DSI por los órganos de la Escuela son o se hacen factibles; el caso de *SIPRES* no es la excepción. No obstante, el estudio de factibilidad de cada nuevo proyecto permite al DSI identificar las necesidades de material humano y tecnológico que involucra dicho proyecto y la mejor manera de satisfacerlas.

El estudio de factibilidad de *SIPRES* consideró los siguientes aspectos:

- Factibilidad técnica[1]. Se contaba con el equipo de cómputo, con el software y con el material humano necesarios para el desarrollo. Por otra parte, en el Departamento de Presupuesto se contaba con una máquina Acer con procesador 386 con 2 MB de memoria RAM (que podrían aumentar a 4 MB mediante un préstamo de simms de memoria por parte del DSI) y tarjeta de red; una impresora IBM de matriz de punto; y un nodo de red dentro de las instalaciones del Departamento en buenas condiciones. Además, el personal del Departamento de Presupuesto tenía nociones de computación. En suma, había los elementos suficientes para el desarrollo de una aplicación en VO, que, como ya se dijo, era la herramienta elegida a priori para el proyecto.

- Factibilidad económica[2]. El procedimiento no automatizado de arqueo requería prácticamente de un día completo de labor de los integrantes del Departamento de Presupuesto para emitirlo y de varios minutos al día para el registro de información. El mayor beneficio que *SIPRES* produciría al Departamento de Presupuesto sería el de una información actualizada al día, compartida por medio de la red con varios órganos, que, mediante la práctica constante, requeriría del mismo tiempo de trabajo diario que con el procedimiento manual, pero de unos cuantos minutos para emitirlo en cualquier momento. Otros beneficios serían la eliminación de errores de

[1] La factibilidad técnica de un proyecto consiste en disponer de la tecnología necesaria para su desarrollo y que ésta se pueda utilizar en la organización.

[2] La factibilidad económica se presenta si el beneficio obtenido sobrepasa el costo del proyecto.

mecanografía[1] y la introducción de un catálogo de proveedores. En estas condiciones, podía decirse que el beneficio superaba al costo[2].

- Factibilidad operativa[3]. El equipo de desarrollo estaba seguro de que, una vez que se le mostraran las ventajas de *SIPRES* sobre el procedimiento tradicional de arqueo al Secretario Administrativo, éste aceptaría la implantación del sistema.

Una vez que el equipo de trabajo realizó el estudio de factibilidad, mostró el prototipo al Jefe del Departamento de Presupuesto y fueron ultimados los detalles relativos a las necesidades de procesamiento de información de este órgano. Posteriormente, el equipo procedió a diseñar el plan de actividades que guiaría el desarrollo de *SIPRES*.

4.2.4. PLANIFICACIÓN DE ACTIVIDADES.

El plan de actividades es un documento que orienta el desarrollo de un proyecto y que incluye los siguientes puntos:

- Establecimiento de los objetivos del proyecto.
- Identificación de las actividades involucradas en el proyecto y sus relaciones.
- Asignación de personal a las actividades.
- Estimación del tiempo que ocupará cada actividad (dotándolas de cierta holgura, considerando los imprevistos que puedan surgir durante el desarrollo del proyecto[4]).

En el caso de *SIPRES*, los objetivos fueron expuestos en el capítulo 2 de esta memoria; la información de los últimos tres puntos del plan de actividades del proyecto se muestra en la siguiente tabla.

---

[1] Como ya se dijo, antes de *SIPRES* los datos de cada documento original eran redactados hasta en tres diferentes ocasiones.

[2] En este caso, el costo del software, de la capacitación a los usuarios del sistema y de soporte técnico lo absorbió el salario que paga la Escuela a los desarrolladores, así que, paradójicamente, no se consideraron en el estudio de factibilidad económica cuestiones monetarias. En cuanto al tiempo requerido para la capacitación de los usuarios del sistema, se pensó que no sería muy prolongado debido a que se trataba de una aplicación Windows y éstas, generalmente, reducen el tiempo de aprendizaje.

[3] Un proyecto es factible operativamente si es real la posibilidad de que se pueda implantar en la organización.

[4] A pesar de las holguras, las actividades que realiza el DSI diferentes a la del desarrollo de sistemas (tales como soporte técnico, asesorías a los usuarios, cotizaciones de equipos y mantenimiento de la red, entre muchas otras), imposibilitan en la mayoría de los casos que los avances en el desarrollo de los proyectos se produzcan en los tiempos establecidos.

Actividad	Responsable	Avance	Ponderación de avance con respecto al total	Fecha de inicio	Fecha de término
Investigación Preliminar	Brenda Covarrubias Jaime Vergara	0%	5%	17 Sept. 96	11 Oct. 96
Análisis	Brenda Covarrubias Jaime Vergara	0%	5%	17 Sept. 96	15 Oct. 96
Diseño General del Sistema	Brenda Covarrubias Jaime Vergara	0%	10%	30 Sept. 96	21 Oct. 96
Desarrollo del Software	Brenda Covarrubias Jaime Vergara	0%	70%	14 Oct. 96	10 Ene. 97
Documentación Técnica	Jaime Vergara	0%	2%	6 Ene. 97	20 Ene. 97
Documentación de Usuario	Lourdes Romero	0%	3%	6 Ene. 97	6 Feb. 97
Prueba de Sistemas	Equipo de confiabilidad de software	0%	5%	6 Ene. 97	27 Ene. 97
TOTAL		0%	100%		

Figura 4.1. Calendarización de las actividades del sistema *SIPRES*.

Una tabla como la anterior auxilia al jefe del DSI y al líder de proyecto a fijar las fechas de revisión de los avances alcanzados, que, generalmente, difieren de la fecha de terminación de la actividad en uno o dos días.

El siguiente paso en el análisis es elaborar el diagrama de funciones del sistema.

4.2.5. DIAGRAMA DE FUNCIONES.

Un diagrama de este tipo es una técnica para modelar las funciones que debe realizar el sistema y representar la información que comparten entre sí; el objetivo de construir este diagrama es orientar al equipo de desarrollo en el posterior diseño del sistema. El diagrama de funciones se divide en tantas partes como subsistemas se identifiquen con base en la información recopilada acerca del procedimiento que se está automatizando.

En el caso de *SIPRES*, el diagrama de funciones se muestra en las figuras 4.2 y 4.3.

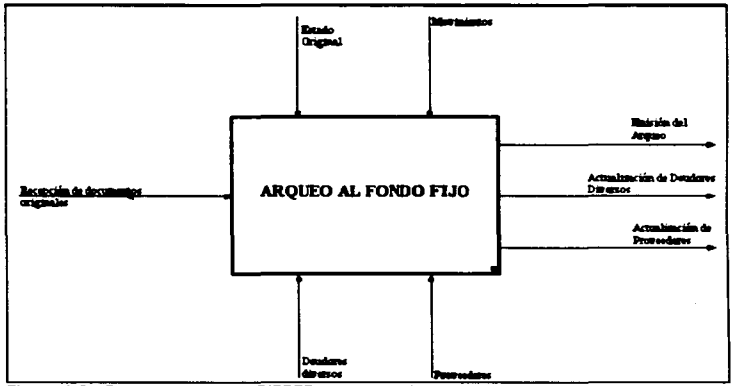

Figura 4.2 . Representación de *SIPRES* como un solo proceso.

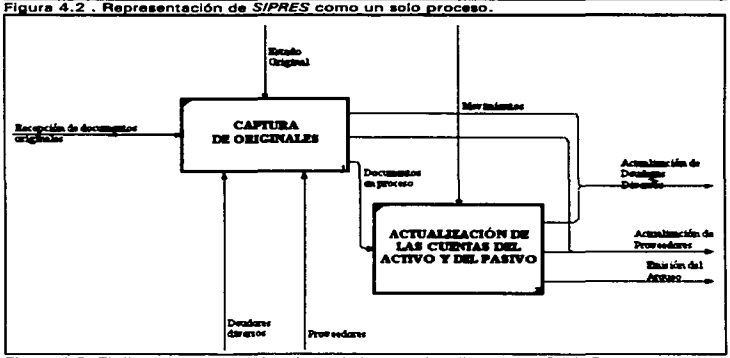

Figura 4.3. El diagrama muestra los dos subsistemas localizados en *SIPRES*.

En las figuras anteriores, los procesos que deberá realizar el sistema se representan por medio de cuadros; las flechas a la izquierda de los mismos indican las entradas del sistema; las flechas de arriba simbolizan acontecimientos que utilizando los datos de entrada provocan una respuesta en el sistema; las flechas bajo el cuadro representan fuentes de alimentación internas (en este caso, catálogos de proveedores y de deudores diversos); finalmente, las flechas a la derecha muestran las salidas del sistema.

4.2.6. DIAGRAMA ENTIDAD-RELACIÓN.

Un diagrama de este tipo auxilia al programador en el posterior diseño de las bases de datos. El diagrama entidad-relación de *SIPRES* se muestra en la figura 4.4.

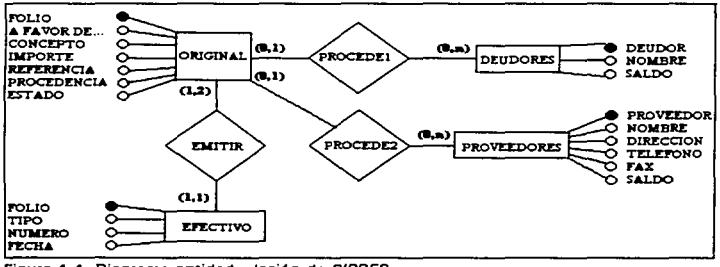

Figura 4.4. Diagrama entidad-relación de *SIPRES*.

En el diagrama anterior se identifican los siguientes elementos:

- Las entidades (rectángulos), que representan a los objetos de los cuales se desea almacenar información en bases de datos: ORIGINAL contendría los datos de los documentos que se reciben en el Departamento de Presupuesto; EFECTIVO listaría los cheques o contrarrecibos emitidos por la cantidad especificada en los documentos; DEUDORES y PROVEEDORES funcionarían como catálogos de deudores diversos y de proveedores, respectivamente.

- Los atributos (círculos) de cada entidad, representan los campos de cada base de datos. Los círculos oscuros indican un campo clave por el cual estaría ordenada la base.

- Las relaciones entre las entidades (rombos) indican lo siguiente: PROCEDE1 relaciona a las entidades ORIGINAL y DEUDORES en caso de que la procedencia del documento sea un deudor diverso; PROCEDE2 es un caso análogo a PROCEDE1, sólo que las entidades en cuestión son ORIGINAL y PROVEEDORES; finalmente, EMITIR relaciona a ORIGINAL y a EFECTIVO, representando el caso en el que un documento llega a los estados de cheque y contrarrecibo.

- Los pares ordenados ubicados sobre las líneas que van de los rectángulos a los rombos representan los números máximo y mínimo de elementos de cada entidad que participan en la relación. Por ejemplo el par (0,1) de ORIGINAL en la relación PROCEDE1 indica que a cada documento puede corresponderle un deudor o ninguno; por el otro lado, el par (0,n) de DEUDORES en la misma relación significa que a cada deudor le puede corresponder ninguno o un número indeterminado de documentos. A cada par ordenado se le llama la cardinalidad de la entidad respecto a la relación en cuestión.

Con los diagramas de funciones y entidad-relación listos, el equipo de desarrollo estaba en condiciones de abordar la siguiente actividad del proyecto: el diseño del sistema.

4.3. EL DISEÑO.

La fase de diseño produce la estructura del sistema (módulos, bases de datos, seguridad de la información, etc.) que posteriormente se materializará en el desarrollo del software

4.3.1. DISEÑO DE LOS MÓDULOS DE *SIPRES*.

En el capítulo 3 se mencionó que resolver un problema por medio de la programación estructurada implicaba dividirlo en partes pequeñas que conformaran los módulos del programa. Así pues, un módulo es un bloque de código que puede modificarse sin repercusiones en el resto, lo que reduce tiempos de depuración y compilación del programa.

Apoyado en el diagrama de funciones ilustrado en las figuras 4.2 y 4.3, el equipo de desarrollo de *SIPRES* decidió dividir el programa en los siguientes módulos:

- *Edición*. Es la fuente de alimentación de *SIPRES*; por medio de éste se capturan los datos de los documentos recibidos por el Departamento de Presupuesto. Responsable: Brenda Covarrubias.

- *Movimientos*. En esta parte se realizan cambios al estado de los documentos vigentes y se asigna número a los cheques y contrarrecibos para facilitar su control. Responsable: Jaime Vergara.

- *Procedencias*. Éste permite imprimir un reporte de documentos vigentes de acuerdo a su procedencia (fondo fijo, ingresos extraordinarios, acreedores diversos, proveedores o cuentas a reserva de comprobar), reporte que será útil para tener impreso el arqueo al fondo fijo. Responsable: Jaime Vergara.

- *Consultas*. Como su nombre lo indica, en esta parte se puede consultar la información de proveedores, deudores diversos, documentos, cheques y contrarrecibos contenida en las bases del sistema. En el caso de proveedores y de deudores, se permite, además de la consulta, la modificación directa de sus datos. Responsable: Brenda Covarrubias.

- *Arqueo*. Aquí se elabora y se imprime el arqueo al fondo fijo a una fecha determinada. Responsable: Jaime Vergara.

- *Administración*[1]. Este módulo presenta las opciones:

    - *Seguridad*. Su función es dar de alta usuarios del sistema o modificar la información de los que ya existen.

    - *Indexar*. Ejecuta el ordenamiento de las bases de *SIPRES*, es decir, genera los archivos índice que permitan realizar consultas más rápidas a la información del sistema.

    - *Generar período*. Por petición del jefe del Departamento de Presupuesto, *SIPRES* se diseñó para realizar un cierre de valores al término de cada año y así comenzar el siguiente con las bases limpias (sólo quedarían vigentes aquellos documentos cuyo trámite no hubiese finalizado el año anterior). La opción generar período crea un nuevo directorio con las bases necesarias para iniciar los trabajos del nuevo año.

[1] Los módulos Edición y Movimientos representan los procesos del diagrama de funciones; Procedencias, Consultas y Arqueo simbolizan las salidas de dichos procesos; finalmente, Administración es un módulo que no se relaciona propiamente con el procedimiento de arqueo, por ello no se encuentra indicado en el diagrama de funciones. Su importancia se verá más adelante cuando se trate el diseño de la seguridad del sistema.

- *Cambio de período*. Considerando la necesidad de consultar la información de cualquiera de los períodos registrados por el sistema, se proporciona al usuario la oportunidad de elegir el período en el que desea trabajar.

Responsable: Jaime Vergara.

Con los módulos ya diseñados, la estructura del menú del programa estaba prácticamente establecida, misma que se muestra en la figura 4.5.

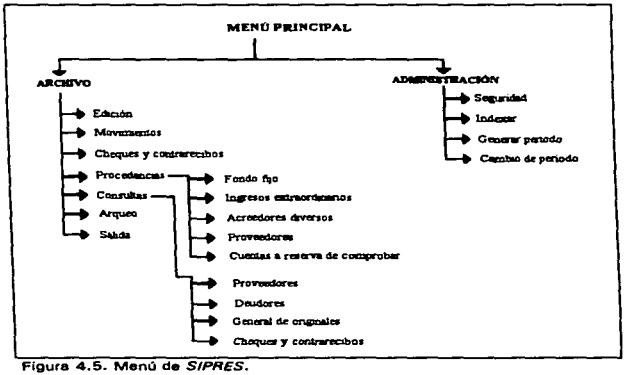

Figura 4.5. Menú de *SIPRES*.

Para darle al sistema la apariencia de una aplicación Windows, se agregaron al menú principal las clásicas opciones Ventana (para ordenar en mosaico o en cascada las ventanas abiertas por el programa) y Ayuda (para activar el sistema de ayuda en línea).

4.3.2. DISEÑO DE LAS BASES DE DATOS.

La decisión de tomar a VO como plataforma de desarrollo del sistema llevaba intrínseca la elección del tipo de bases de datos a utilizar: archivos DBF con índices NTX.

Con base en lo anterior, el equipo de desarrollo de *SIPRES* diseñó las bases de datos que integrarían el sistema. En las siguientes líneas se expone una breve descripción de la utilidad de cada base, así como su estructura y sus índices (en la columna correspondiente al tipo del campo se muestran las letras N, C o D, dependiendo de si se trata de un campo numérico, alfanumérico o de fecha, respectivamente; por lo que respecta a la longitud, se indica en los campos numéricos que estos se componen de dos cifras decimales).

- *Arqueo.* Base temporal que contiene cada una de las cantidades que integran el arqueo al fondo fijo.

CAMPO	TIPO	LONGITUD	DESCRIPCIÓN
BANCO	N	16-2	Cantidad de activo en bancos
CHEQUE	N	16-2	Cantidad de activo en cheques
CONTRARECI	N	16-2	Cantidad de activo en contrarrecibos
ORIGINAL	N	16-2	Cantidad de activo en originales
PORTRAM	N	16-2	Cantidad de activo en documentos por tramitar
ENTRAM	N	16-2	Cantidad de activo en documentos en trámite
RECHAZO	N	16-2	Cantidad de activo en rechazo
DEUDORES	N	16-2	Cantidad de activo en deudores diversos
TOTAL1	N	19-2	Total de activo
FONDOFIJO	N	16-2	Cantidad de pasivo en fondo fijo
INGREX	N	16-2	Cantidad de pasivo en ingresos extraordinarios
ARESERVA	N	16-2	Cantidad de pasivo en cuentas a reserva de comprobar
ACREEDOR	N	16-2	Cantidad de pasivo en acreedores diversos
PROVEEDOR	N	16-2	Cantidad de pasivo en proveedores
TOTAL2	N	19-2	Total de pasivo
FECHA	D	8	Fecha del arqueo

Figura 4.6. Base Arqueo.

- *Proveedo.* Almacena los datos personales de los proveedores de la ENEP.

CAMPO	TIPO	LONGITUD	DESCRIPCIÓN
PROVEEDO	C	4	Clave del proveedor
NOMBRE	C	50	Nombre del proveedor
DIRECCION	C	60	Dirección
TELEFONO	C	12	Teléfono
FAX	C	12	Fax
SALDO	N	19-2	Saldo actual
ARCHIVO ÍNDICE			CAMPO(S) CLAVE
PROVEEDO.NTX			NOMBRE
CLAVPROV.NTX			PROVEEDOR

Figura 4.7. Base Proveedo.

- *Deudores.* Almacena los datos personales de los deudores diversos de la ENEP.

CAMPO	TIPO	LONGITUD	DESCRIPCIÓN
DEUDOR	C	2	Clave del deudor
NOMBRE	C	50	Nombre del deudor
SALDO	N	19-2	Saldo actual
ARCHIVO ÍNDICE			CAMPO(S) CLAVE
DEUDORES.NTX			NOMBRE
CLAVDEUD.NTX			DEUDOR

Figura 4.8. Base Deudores.

- *Periodos*. Contiene las rutas en el servidor de los directorios que representan a los períodos de trabajo.

CAMPO	TIPO	LONGITUD	DESCRIPCIÓN
FPERIODO	C	4	Período
PATH	C	50	Ruta de las bases en el servidor
FONDOFIJO	N	19-2	Fondo fijo para el período
BANCO	N	19-2	Capital en banco en el período
ARCHIVO ÍNDICE			CAMPO(S) CLAVE
PERIODOS.NTX			FPERIODO

Figura 4.9. Base Periodos.

- *Usuarios*. Contiene los datos personales de los usuarios de *SIPRES*.

CAMPO	TIPO	LONGITUD	DESCRIPCIÓN
USER	C	10	User
PASSWORD	C	21	Clave del usuario encriptada
ATRBMGRAL	C	36	Permisos para el sistema (encriptados)
NOMBRE	C	30	Nombre del usuario
ARCHIVO ÍNDICE			CAMPO(S) CLAVE
USUARIOS.NTX			USER

Figura 4.10. Base Usuarios.

- *Efectivo*. Contiene los números de cheques y contrarrecibos.

CAMPO	TIPO	LONGITUD	DESCRIPCIÓN
FOLIO	C	5	Folio del documento al que pertenece el cheque o contrarrecibo
TIPO	C	2	"CH" o "CR", según corresponda
NUMERO	C	6	Número de cheque o contrarrecibo
FECHA	D	8	Fecha de emisión
ARCHIVO ÍNDICE			CAMPO(S) CLAVE
EFECFOL.NTX			FOLIO + TIPO

Figura 4.11. Base Efectivo.

- *Original*. Guarda la información de todos los documentos que ingresan al Departamento de Presupuesto.

CAMPO	TIPO	LONGITUD	DESCRIPCIÓN
FOLIO	C	5	Folio del documento
AFAVORD	C	50	A nombre de quién está el documento (esto en caso de que no se trate de un documento proveniente de un deudor ni de un proveedor, en cuyo caso se utiliza el campo CLAV_DDP)
CONCEPTO	C	50	Motivo del documento
IMPORTE	N	12.2	Cantidad manejada en el documento
REFERENCIA	C	50	Referencia del documento
PROCEDENCI	C	1	I: ingresos extraordinarios; A: acreedores diversos; F: fondo fijo, C: cuentas a reserva de comprobar; P: proveedores
FECHDD	D	8	Fecha en que el documento pasó al estado de "deudor diverso"
FECHOR	D	8	Fecha en que el documento pasó al estado de "original"
FECHPT	D	8	Fecha en que el documento pasó al estado de "por tramitar"
FECHET	D	8	Fecha en que el documento pasó al estado de "en trámite"
FECHRZ	D	8	Fecha en que el documento pasó al estado de "rechazo"
FECHBC	D	8	Fecha en que el documento pasó al estado de "banco"
FECHFIN	D	8	Fecha en que el documento terminó su trámite
ESTADO	C	1	O: original; P: por tramitar; E: en trámite; R: rechazo; C: contrarrecibo; H: cheque; B: banco; D: deudor diverso; T: fin
CLAV_DDP	C	4	Clave del deudor o del proveedor (en caso de que el documento provenga de alguno de ellos)

ARCHIVO ÍNDICE	CAMPO(S) CLAVE
ORFOLIO.NTX	FOLIO
ESTADO	ESTADO

Figura 4.12. Base Original.

Como ya se mencionó anteriormente, las bases de datos de *SIPRES* deberían almacenarse en alguno de los dos servidores de red del DSI con el objeto de que los órganos autorizados para consultarlas lo pudiesen hacer desde su propia terminal. El compartir información por medio de una red fuerza la existencia de niveles de seguridad que impidan que pueda ser consultada o modificada de forma indebida; los niveles de seguridad con que los desarrolladores del DSI dotan a sus sistemas se describen en el siguiente apartado.

4.3.3. DISEÑO DE LA SEGURIDAD.

Es responsabilidad de los desarrolladores del DSI instrumentar mecanismos de seguridad de la información que la protejan contra alteraciones involuntarias o no autorizadas que afecten su precisión.

La seguridad de la información que generan las aplicaciones desarrolladas por personal del DSI se divide en dos niveles: el de sistema operativo de red y el de la propia aplicación.

Para poder usar un sistema desarrollado por el DSI, primero se debe tener cuenta de usuario en el servidor donde están almacenados tanto el sistema en cuestión como su información[1]. Cada usuario dispone de un nombre y de un password para poder iniciar una sesión en el servidor; cuando se crea su cuenta, se le otorga el derecho de acceder a los directorios donde se encuentran los archivos (los cuales puede leer y en su caso modificar) que manipula el sistema que va a utilizar. Es responsabilidad del administrador de la red crear la cuenta del usuario y otorgar el acceso a los directorios que el líder de proyecto le indique para que pueda hacer uso del sistema.

El párrafo anterior indica la forma de instrumentar la seguridad a nivel del sistema operativo que, como puede apreciarse, es responsabilidad compartida del administrador de la red y del líder de proyecto. El diseño del segundo nivel de seguridad, el de la aplicación, es tarea del equipo de desarrollo del sistema..

Los usuarios de un sistema se identifican por medio de un nombre y de un password (que, en principio, no tienen ninguna relación con el nombre y el password de su cuenta de red) y cada uno puede utilizar únicamente aquellas opciones del menú que le sean autorizadas (cuando el programa determina quién lo está utilizando, activa las opciones del menú a las cuales tiene derecho ese usuario).

Generalmente, el responsable del órgano para el que fue elaborado el sistema es el que indica quiénes pueden ser usuarios del sistema y qué opciones están disponibles para ellos; el líder de proyecto se limita a darlos de alta y a especificarles sus permisos.

En el caso particular de *SIPRES*, la Dirección de la Escuela sería la que dictaminaría qué órganos serían usuarios del sistema; lo que sí estaba establecido de antemano era que el único órgano con el derecho de capturar información en el sistema sería el Departamento de Presupuesto, mientras que los demás sólo podrían hacer consultas.

[1] Actualmente el DSI administra dos servidores de red: "DSI" con sistema operativo Netware y "DSI01" con sistema operativo Windows 95.

Estos son los dos niveles de seguridad de sistemas que maneja el DSI. Una consideración final al respecto es que cada usuario es responsable de no divulgar sus passwords, tanto el de red como el del sistema, pues de ello depende que no se haga mal uso de la información.

Con el diseño de la seguridad finaliza la etapa de diseño del sistema y se da paso al desarrollo del mismo.

4.4. DESARROLLO DE *SIPRES*.

El desarrollo del sistema permite materializar las ideas plasmadas en el diseño utilizando un lenguaje de programación adecuado. En el caso de *SIPRES*, dicho lenguaje fue VO, cuyas características principales se explicaron en el capítulo 3.

Habiendo sido diseñados ya los módulos y las bases que conformarían el sistema, la labor del equipo encargado de *SIPRES* se orientó a trabajar con su herramienta visual de desarrollo. Esta actividad, que se planeó para iniciar el 14 de octubre de 1996 y finalizar el 10 de enero de 1997, se concluyó mucho antes de lo planeado: el 22 de noviembre de 1997 se presentó ante el jefe del DSI el funcionamiento completo del sistema[1].

4.4.1. CONSIDERACIONES SOBRE LA PROGRAMACIÓN EN VO.

Aprender a explorar el código que generan de forma automática los editores visuales de VO es, quizá, la clave para aprender a programar en esta plataforma. En dicho código se descubre la forma como deben manipularse las propiedades de las ventanas (principales objetos visuales) y también la forma como deben relacionarse con los servidores de bases de datos.

Una vez que se identifican los nombres de las propiedades de los objetos visuales y su funcionamiento, programar funciones o métodos que los utilicen se convierte en tarea fácil para aquellos desarrolladores familiarizados con Clipper, puesto que el lenguaje de VO es casi idéntico a él; en el caso de programadores para los que VO representa su primer lenguaje de desarrollo, tardarán un poco más en obtener resultados, pero el manual de usuario y la ayuda en línea de VO les será de gran valía.

Los editores de ventanas, menús y reportes de VO disminuyen considerablemente el tiempo de diseño y codificación de las salidas de las

[1] La herramienta visual de desarrollo fue la responsable directa en la reducción del tiempo de codificación del programa.

aplicaciones, lográndose una excelente calidad en cuanto a la apariencia se refiere.

Una consideración final, referente al que posiblemente sea el defecto más grave de VO, es que esta plataforma consume una gran cantidad de recursos de hardware, tanto en el aspecto de desarrollo como en el de ejecución de las aplicaciones. Esto es algo que deberán considerar los fabricantes del producto en futuras versiones.

En los siguientes apartados se explican brevemente las actividades que se llevaron a cabo para construir *SIPRES*.

4.4.2. CONSTRUCCIÓN DE LAS BASES DE DATOS.

Utilizando el DBU.EXE, la utilería de Clipper para construir y manipular bases de datos, se crearon físicamente las bases que conformarían el sistema, cuya estructura ya se explicó en la sección anterior.

Posteriormente, se utilizó el editor visual de servidores de bases de datos para incorporar las bases creadas por DBU a la aplicación. Al construir cada servidor de bases de datos, se les especifican atributos a sus campos, entre los que destacan: longitud mínima y máxima; tipo (numérico, fecha, alfanumérico o lógico); y un segmento de código para validar que no se le asigne un contenido equivocado (esto en el caso de que dicho contenido se introduzca desde una pantalla de captura)[1].

4.4.3. DISEÑO-DESARROLLO DE LAS VENTANAS.

La esquematización de las salidas a pantalla de un sistema es una labor que, generalmente, se realiza durante la etapa de diseño del sistema. Sin embargo, el trabajar con una herramienta visual como VO permite que el diseño de las ventanas de captura y de consulta produzca al mismo tiempo su codificación (de aquí el título del apartado).

Con el objeto de dar al lector una ligera idea de la apariencia de *SIPRES*, se muestran aquí las ventanas que representan las opciones más importantes del sistema.

[1] Un ejemplo de validación sería el siguiente: supóngase una pantalla de captura de los datos personales de los empleados de cierta empresa en la que se impide introducir letras en el dato "código postal". La validación de la información que introduce el usuario desde el teclado es vital para evitar la filtración de datos erróneos a las bases del sistema.

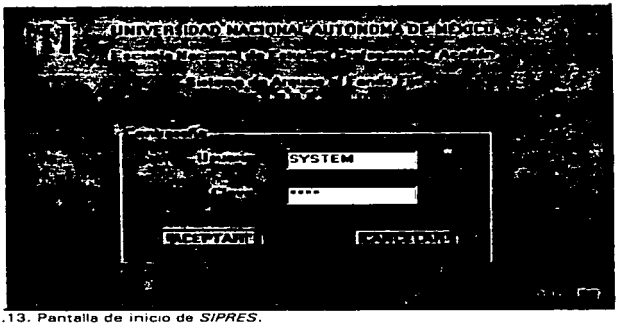

Figura 4.13. Pantalla de inicio de *SIPRES*.

Figura 4.14. Ventana de captura de documentos.

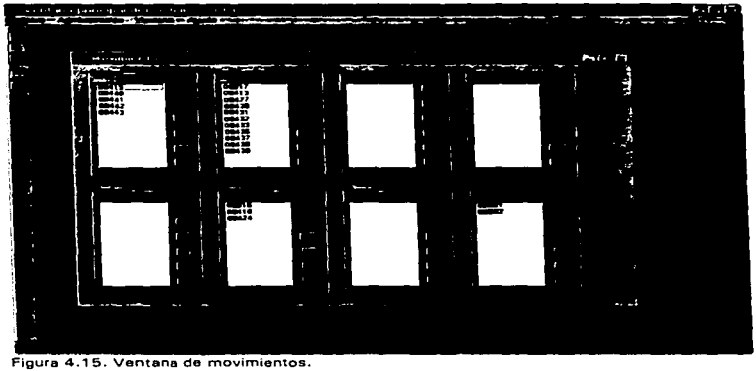

Figura 4.15. Ventana de movimientos.

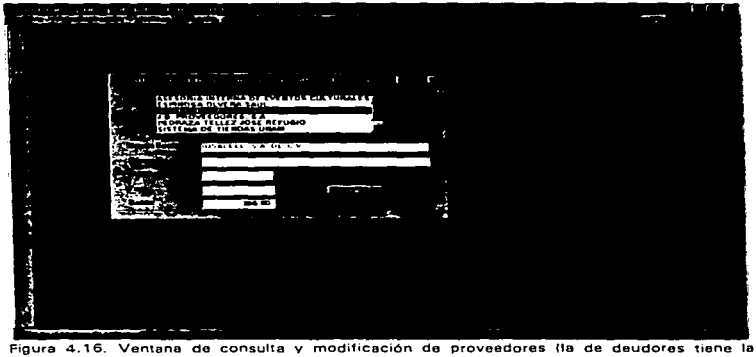

Figura 4.16. Ventana de consulta y modificación de proveedores (la de deudores tiene la misma estructural).

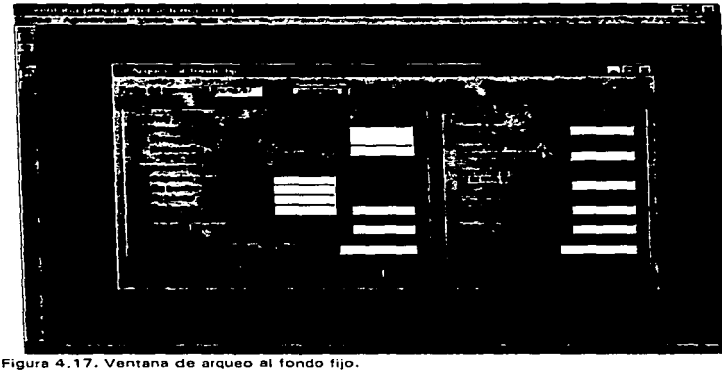

Figura 4.17. Ventana de arqueo al fondo fijo.

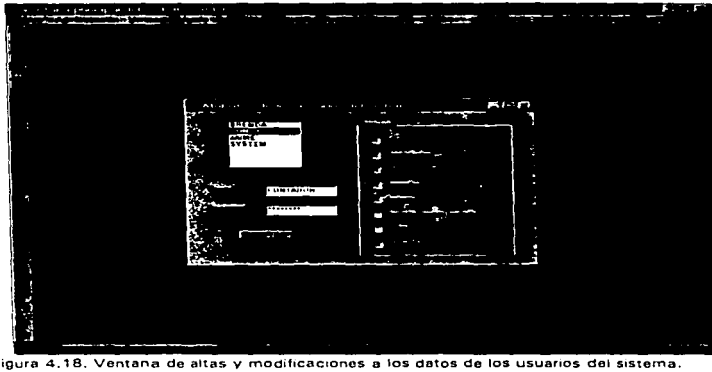

Figura 4.18. Ventana de altas y modificaciones a los datos de los usuarios del sistema.

4.4.4. DISEÑO-DESARROLLO DE LOS REPORTES.

Al igual que en el caso de las salidas a pantalla, diseñar las salidas a impresora en VO es, al mismo tiempo, programarlas. VO incorpora un asistente para el diseño de los reportes de sus aplicaciones llamado CARET, el cual produce documentos con extensión .RET. Cuando el usuario activa una opción correspondiente a la emisión de un reporte, la aplicación busca la ubicación del documento .RET correspondiente, mismo que se visualiza en pantalla y después puede imprimirse. Esto implica que, al igual que las bases, dichos documentos deban ubicarse en algún directorio del disco duro del servidor en el que se ejecuta el sistema en cuestión.

4.4.5. GENERACIÓN DE FUNCIONES Y MÉTODOS.

Ya se ha mencionado cómo los editores visuales de VO generan código por sí mismos. Sin embargo, es evidente que ese código no cubre todas las necesidades de programación del sistema, por lo que el desarrollador tiene que construir sus propias funciones o incorporar otros métodos a las clases ya existentes.

Haber programado anteriormente en Clipper fue una gran ventaja para el equipo de trabajo de *SIPRES*, puesto que pudo incursionar con cierta facilidad en la construcción de procedimientos propios en el lenguaje de VO; incluso, se pudo adaptar a la aplicación código hecho en Clipper para otros sistemas (por ejemplo el que corresponde a la seguridad del sistema).

La aplicación estaba ya desarrollada; ahora tenía que comprobarse si funcionaba correctamente. En la siguiente sección se describe la forma como el equipo de desarrollo y el equipo de confiabilidad del software (ECS) del DSI realizaron las pruebas al sistema para detectar y corregir sus posibles fallas.

4.5. CONFIABILIDAD Y PRUEBAS DEL SOFTWARE.

El funcionamiento de todo sistema debe ser tal que provoque que el usuario confíe en los resultados que le presenta y en la información que le proporciona. Se dice que un sistema es confiable si no produce fallas cuando se utiliza para lo que debe y como debe utilizarse, entendiendo como falla un error del sistema que provoque consecuencias graves para el usuario[1].

[1] Supóngase que el sistema computarizado para la venta de boletos de un espectáculo indica que ya no hay lugares cuando sí los hay; éste es un error que trae como consecuencia –grave– para el usuario la pérdida de ganancias. Supóngase ahora que el mismo sistema deja de funcionar a las 18:00 horas y se necesita que funcione hasta las 19:00 horas; este error no es de consecuencias graves, puesto que se puede alterar la hora de la computadora.

Las fallas no detectadas antes de poner en marcha al sistema ocasionan la necesidad de darle mantenimiento[1]. Al respecto puede señalarse que un buen análisis del sistema reduce la posibilidad de fallas, y que un buen diseño debe dotar al sistema de la flexibilidad suficiente como para que permita que se le dé mantenimiento.

Antes de poner en marcha cualquier sistema debe probarse su funcionamiento, es decir, debe ejecutarse buscando condiciones propicias para que falle. A continuación se describen los tipos de prueba más importantes que instrumentan sobre sus sistemas el equipo de desarrollo y el ECS del DSI[2].

- *Prueba de código*. Este tipo de prueba la realiza el desarrollador desde que está programando, con el fin de verificar que sus algoritmos funcionen correctamente. Para llevarla a cabo, el programador se vale del depurador de Clipper o del de VO, según sea el caso.

- *Prueba de validación de la entrada de datos*. En este caso, el ECS verifica que el sistema no acepte entradas de datos erróneas en las pantallas de captura.

- *Prueba de almacenamiento y modificación de la información*. Consiste en corroborar que la información introducida en las pantallas de captura se haya guardado correctamente en las bases de datos y que las modificaciones sobre un registro determinado se almacenen, efectivamente, sobre ese registro.

- *Prueba de consultas y reportes*. Se comprueba que en las pantallas de consulta y en los reportes aparezcan correctamente todos los datos incluidos en su diseño.

- *Prueba del sistema en red*. Se realizan las pruebas anteriormente descritas pero con el sistema funcionando en red y teniendo conectados al mayor número posible de usuarios. En este caso la prueba de almacenamiento es realmente importante.

- *Prueba de la ayuda y de los manuales de usuario y de procedimientos*. Se elige a uno de los integrantes del ECS que no conozca el funcionamiento del sistema para comprobar si puede manipularlo apoyándose en la ayuda en

[1] También son causa de mantenimiento los cambios en las políticas y procedimientos de trabajo y el avance tecnológico.

[2] Debe recordarse que a pesar de que se realicen cientos de pruebas a un sistema antes de ponerlo en marcha, éste nunca estará libre de errores, puesto que es imposible imaginar todas las situaciones bajo las cuales puede funcionar y realizar pruebas al respecto.

línea del sistema, en el manual de usuario y en el manual de procedimientos.

4.6. DOCUMENTACIÓN DEL SISTEMA.

Esta actividad es, sin lugar a dudas, la menos grata para un desarrollador y, sin embargo, resulta tan importante como la programación misma; una buena documentación reduce tiempos en el mantenimiento del sistema y, por tanto, reduce también su costo.

La documentación que prepara un equipo de desarrollo del DSI consiste en:

- *Manual de usuario*. En este documento se indica con el mayor grado de detalle posible el funcionamiento del sistema para facilitar el aprendizaje del usuario o la resolución de problemas que puedan presentarse durante su ejecución[1].

- *Manual de procedimientos*. Este documento informa al usuario los cambios que se presentaron en sus procedimientos y prácticas habituales de trabajo a raíz de la implantación del sistema, así como la nueva forma de realizarlos.

- *Manual del administrador*. Este manual describe las tareas que debe realizar el administrador del sistema (ordenar bases, dar de alta a usuarios, modificar los datos de los usuarios, etc.)[2].

- *Manual técnico*. Este manual es una guía para los programadores que le dan mantenimiento al sistema. En él se incluyen aspectos tales como:

    - La investigación preliminar.
    - Los objetivos del sistema.
    - El diagrama de funciones.
    - La estructura de las bases de datos y sus índices.

[1] Cabe señalar que el usuario cuenta con un sistema de ayuda en línea dentro del mismo programa que, en muchas ocasiones, resulta más práctico que el propio manual.

[2] En el caso de *SIPRES*, el administrador del sistema es el propio líder de proyecto. La razón de lo anterior es que la generación de nuevos períodos de trabajo (que es parte de la administración) implica la modificación de algunas rutas en los reportes, misma que debe hacerse desde VO. En otros sistemas en los que la administración implica la simple ejecución de opciones del menú, el encargado de ella es alguno de los usuarios.

- La estructura de los reportes emitidos por el sistema y un ejemplo de cada uno.
- Una breve descripción de los programas que integran cada módulo, de las funciones que constituyen cada programa y de las variables públicas más importantes de cada programa.
- El directorio en el disco duro del servidor de las bases reales y de las bases de prueba.
- Una guía de instalación.
- Una tabla de problemas más frecuentes reportadas por los usuarios del sistema, su causa y su solución.
- Las conexiones de la aplicación con otros sistemas[1].

El manual de usuario de cada sistema realizado por integrantes del DSI corre a cargo del personal de apoyo del Departamento, bajo la supervisión del líder de proyecto; el resto de la documentación es responsabilidad del equipo de desarrollo del sistema.

Con esto se ha llegado prácticamente al final de la memoria; sólo resta enunciar las conclusiones que su realización inspiró a su autor.

[1] Estas conexiones pueden consistir en compartir una o más bases de datos o en que los procesos que realiza alguno de los sistemas dependan de la información que otro produzca.

CONCLUSIONES.

A continuación se exponen algunas reflexiones motivadas por la elaboración de este estudio.

- Existen varias metodologías de análisis, diseño y desarrollo de sistemas, pero su aplicación es muy flexible, condición que permite omitir algunos pasos y mezclar varios enfoques, dependiendo siempre de la naturaleza del proyecto que se trate.

- Las herramientas visuales de desarrollo resultan de gran utilidad para producir aplicaciones atractivas para los usuarios, reduciendo tiempos de programación y de capacitación. Tal vez la mayor desventaja de dichas aplicaciones es que, en general, resultan ser grandes consumidoras de recursos de hardware.

- Es impresionante la rapidez con que evolucionan tanto la tecnología de la computación como las plataformas de desarrollo de aplicaciones. Sin

embargo, un programador puede estar limitado por los recursos con que en ese sentido cuente la institución para la que trabaje.

- Parece un reto interesante para el egresado de la carrera de Matemáticas Aplicadas y Computación crear sistemas de información que incorporen modelos matemáticos útiles para la toma de decisiones, logrando de esta manera la conjunción de esas dos partes de su formación profesional y contribuyendo así a mejorar la planeación de las actividades de su empresa. En ese sentido, *SIPRES* podría convertirse en un sistema que alimentara de información a otro que fuese capaz de optimizar el empleo de los recursos financieros de la escuela con base en un modelo de asignación.

Agosto de 1997.

BIBLIOGRAFÍA.

1. Senn, James A. *Análisis y diseño de sistemas de información.* Ed. McGraw Hill. México, 1989.

2. Piatini Velthuis, Mario G., et. al. *Análisis y diseño detallado de aplicaciones informáticas de gestión.* Ed. RA-MA. Madrid, 1996.

3. *CA-Visual Objects: getting started.* Computer Associates. E. U. A., 1994.

4. Spence, Rick. *CA-Visual Objects: guía para el programador.* Ed. RA-MA. México, 1997.

5. *CA-Visual Objects: IDE user guide.* Computer Associates. E. U. A., 1994.

6. *CA-Visual Objects: programers guide*, Vol. III. Computer Associates. E. U. A., 1994.

7. *Escuela Nacional de Estudios Profesionales Acatlán: manual de organización 1995.*

8. *Hacia Visual Objects: actas de los encuentros 1994 de usuarios Clipper.* Ed. Addison Wesley Iberoamericana. E. U. A., 1994.

9. *Legislación laboral universitaria*, U.N.A.M., 1992.

10. Boter y Mauri, Fernando. *Nociones fundamentales de contabilidad*, 3a. ed. Ed. Juventud, S.A. Barcelona, 1963.