UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

ELABORACION DE UN PRODUCTO DE SOFTWARE QUE PERMITA INCORPORAR LA TECNOLOGIA DE IMPRESION ELECTRONICA AL AMBIENTE UNIX

T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACION

P R E S E N T A N :

JULIO JESUS BUEYES ARTEAGA
ARACELI CAMACHO CAMPOS
FRANCISCO MIRANDA DEL VALLE
MARIO PIÑA VAZQUEZ
JAIME SANCHEZ PATLAN
RENE VASQUEZ PEREZ

Director: Ing. Aurelio Adolfo Millán Nájera

MEXICO, D.F. **SEPTIEMBRE, 1995**

FALLA DE ORIGEN

A mi **Padre**, *que con sus consejos, amor y esfuerzo me ha legado como herencia, la educación y preparación académica, que he alcanzado, la cual me ha permitido tener bienestar y éxito*

A mi Madre Alicia, por haberme brindado la vida, sin la cual no estaría presente en este instante

A la Memoria de mi Madre Dolores que con amor, consejos y esfuerzo, logró infundirme los valores de la vida e impulsarme para realizarme como persona.

A mis hermanos, como un incentivo para que logren sus metas y vean que con esfuerzo si se puede.

A mi **Esposa**, *que con su amor, comprensión y apoyo me ha impulsado a terminar una preparación profesional.*

A mis hijos, para que les sirva de ejemplo y como un estímulo para que crezca la semilla de la preparación continua y sigan adelante en los estudios.

A mis familiares y amigos, que gracias a su apoyo he logrado alcanzar este importante objetivo de mi vida.

A mis compañeros y amigos en la realización del presente seminario, ya que con su apoyo, tenacidad y esfuerzo hemos logrado esta meta.

Julio

Con cariño a mi madre, en reconocimiento a sus preocupaciones y sacrificios realizados para que mi existencia no fuese vana. Este también es tu éxito.

A mi padre por su esfuerzo y empeño, para lograr sacarnos adelante.

A mis hermanas : Clara y Rocío por ofrecerme siempre y en todo momento su apoyo, comprensión y motivación para lograr la realización de este anhelo.

A mis sobrinos, con la seguridad y deseo de que esto sea una motivación más para que ellos obtengan mayores y mejores éxitos en la vida.

A Becky, Lolita y Coquito por estar en los buenos y malos momentos de mi vida, por sus palabras de aliento y apoyo; pero sobre todo, por ser mis mejores amigas.

En especial a Julio, Francisco, René, Mario y Jaime, por ser mis grandes compañeros de trabajo, por todo su apoyo y empeño para culminar una de nuestras metas y sobre todo por su amistad.

Araceli

A mis padres; Leonor Vázquez y Tomás Piña

A mis hermanos; Ma. Luisa, Dolores, Hilda, Gloria, Herlinda, Guillermo y Claudia

A mis amigos, ...

por su apoyo incondicional, Gracias!

Con especial agradecimiento a Julio, Araceli, Jaime, Francisco y Rene, con quienes tuve la oportunidad de trabajar en la realización de este trabajo.

Mario

A mis padres por haberme entregado su amor, su paciencia y su vida.

A mi esposa e hijo por su amor, comprensión y apoyo.

A mis hermanos por todos los momentos que vivimos.

A mis compañeros y amigos, a todos ellos por su amistad.

A los señores Andres Barrios, Manuel Angulo, Humberto Lujan, José Luis Calderón, Francisco Bahamonde y Luis Quezada por sus Enseñanzas.

A la vida, por todo lo que me ha dado.

Francisco

A mis padres por brindarme la oportunidad de haber terminado una carrera, ya que ***con su apoyo y dedicación me han abierto las puertas para enfrentar la vida. Me han heredado lo más valioso que pudieran haberme dado, la educación. Gracias a ambos por su ejemplo ya que siempre se han esforzado trabajando por nosotros; Antonia y Raymundo los quiero.***

A mis hermanas Beatriz y Guadalupe, como ejemplo de superación y que ellas brinden la misma oportunidad a sus futuros hijos.

A mi hermano Raymundo para que se esfuerce por alcanzar lo que se ha propuesto y triunfe en la vida.

Dulce Karina espero que crezcas sana y feliz, estudia y trabaja con empeño, solo así se llega a ser alguien valioso en este mundo.

A aquellos profesores quienes transmitieron sus conocimientos y consejos, gracias.

A mis amigos y amigas cuyo apoyo siempre me empujo para llegar hasta el final, gracias.

Gracias también a mis compañeros de tesis: Araceli, Francisco, Julio, Mario y René, ya que con su esfuerzo y deseo de superación pudimos realizar este trabajo. Deseo que todos sus anhelos y objetivos que aún no han realizado, se vean alcanzados lo más antes posible y que continúen triunfando en la vida.

Jaime

A Lupita

Que es el motivo de todas las cosas que hago, que con su amor y paciencia me ha ayudado a superar los obstáculos que se nos han presentado.

A Iván

Que desde su llegada ha alegrado todos los momentos que vivimos juntos.

A Miguel

Sin su apoyo, no habría logrado salir adelante.

A mis padres y hermanos

Que me permitieron elegir el camino de mi vida y por todos los esfuerzos que hicieron para que alcanzara todas mis metas.

A mis compañeros de equipo Araceli, Franciso, Mario, Jaime y Julio por su valiosa cooperación.

René

Al Ing. Aurelio Adolfo Millán Nájera, por su apoyo, comprensión y sobre todo por su cooperación en la realización del presente seminario, sin él no hubieramos alcanzado esta importante meta de nuestras vidas, por todo lo anterior, muchísimas gracias.

Julio, Araceli, Jaime,
Francisco, Mario, René

PROLOGO

A todos los hombres les gusta creer que pueden hacer las cosas solos, pero el hombre de verdad sabe que no hay nada como el apoyo, el estímulo y un equipo.

Tim Allen.

Elaboración de un producto de software para incorporar la tecnología de impresión electrónica en el ambiente UNIX.

El objetivo de este trabajo es el de diseñar un lenguaje de creación de formas electrónicas, un compilador y una interfase de impresión para el ambiente UNIX, que permita mezclar los archivos de impresión con formas electrónicas en una impresora láser.

A continuación se da una breve descripción de lo que se verá en cada uno de los capítulos que forman este trabajo.

El capítulo I, Introducción, muestra los antecedentes históricos de la página escrita, el funcionamiento de una impresora láser, la historia de la aparición de la primera impresora láser por Xerox, así como algunas de las características más sobresalientes de los sistemas UNIX, terminando con la justificación para la realización del producto en cuestión, tiempo de desarrollo y personal requerido para el desarrollo.

El capítulo II, Análisis de Requerimientos, especifica cual será el ambiente de desarrollo, operación y mantenimiento del producto, los flujos de datos encontrados en el proceso utilizando la metodología de Yourdon, los requerimientos funcionales del producto y los de operación del mismo. Se proporcionan los lineamientos que deberán seguirse para las prioridades de instrumentación, una lista de las modificaciones y mejoras previstas para el producto en desarrollo, los criterios que se adoptaron para la aceptación del producto y las guías y sugerencias para el diseño.

El capítulo III, Diseño del Producto, muestra en primer lugar el lenguaje de diseño de formas que se desarrolla, utilizando una notación de Backus y de Railroad muy utilizada por Unisys, siguiendo el diseño de un módulo de rutinas que se encontraron como comunes, el diseño del compilador de formas, generado a partir del lenguaje de diseño de formas previamente establecido y la interfase de impresión del producto, junto con dos módulos adicionales, que son de soporte para el funcionamiento del producto. Termina el capítulo con el diseño de las pruebas de aceptación.

El capítulo IV, Instrumentación, proporciona los estandares de programación establecidos para la programación con el lenguaje "C", una relación entre los programas y los archivos que intervienen en el desarrollo del producto y unos ejemplos de los programas realizados.

El capítulo V, Pruebas, indica los resultados obtenidos al realizar pruebas de funcionamiento del producto, siguiendo los lineamientos establecidos en el capítulo Diseño del Producto.

El capítulo VI, Conclusiones, permite indicar los comentarios relacionados con el desarrollo del producto, como corolario del uso de la metodología de desarrollo de sistemas usado, los problemas que se presentaron y como se resolvieron, entre otros aspectos relacionados al desarrollo de sistemas en México.

Finalmente el Anexo A muestra una ampliación de los aspectos de análisis de requerimientos que no fueron cubiertos en el capítulo correspondiente.

Por último es necesario mencionar que el manual de usuario del producto desarrollado se encuentra en un documento aparte, la razón de ello es no hacer más voluminoso el presente trabajo.

CONTENIDO

INTRODUCCION

1.1 Antecedentes

La llegada de los Sistemas de Impresión Electrónica como dispositivos periféricos para obtener la salida de un computador, han modificado la forma en que la misma salida se obtiene, ya que estos sistemas ofrecen mayor calidad y velocidad.

Pero hablando de velocidad, se observa que la unidad de medida también ha sufrido cambios, pues de líneas y caracteres por minuto con que se acotaba la velocidad de las impresoras de impacto, con la llegada de los Sistemas de Impresión Electrónica la velocidad se acota en páginas por minuto, lo cual indica que estos sistemas son capaces de imprimir una página de información en un solo paso.

Por lo anterior es importante conocer brevemente la historia de la impresión hasta nuestros días.

1.1.1 Bosquejo histórico de la página escrita

Se estima que la página escrita aparece en el mundo alrededor del año 3500 A. C. en la Costa Oriental del Mar Mediterráneo, según se ha podido constatar en las evidencias encontradas en las ruinas de las antiguas ciudades. Este hecho es bastante significativo, ya que las causas que originaron este suceso fueron el crecimiento económico, la fundación de instituciones, como el gobierno y la iglesia, así como la necesidad de designar propiedades y mantener un registro de los mismos. La escritura que se aprecia fue una combinación de pictografías utilizadas como ideogramas y signos silábicos.

Los signos tuvieron un desarrollo en los siguientes dos milenios y medio para aparecer como signos en forma de cuña: la estructura de la escritura de los Asirio-Babilónicos. Este tipo de escritura influyó en el desarrollo de los jeroglíficos escritos en Egipto y siglos después, en una escritura similar en Asia Central y China. Este tipo de escritura no describía simplemente los objetos que representaban, muchas de las veces como caricaturas, sino por el contrario, representaban valores fonéticos y elementos gramaticales.

Lo más importante de este tipo de escritura es la forma en que fueron escritos, la escritura cuneiforme Babilónica fue ampliamente escrita de izquierda a derecha, algunas veces en renglones de diferentes tamaños y longitudes, así como todas las formas de escritura alrededor de objetos circulares.

Los jeroglíficos Egipcios fueron escritos de derecha a izquierda, de izquierda a derecha, de arriba a abajo y aún alrededor de círculos concéntricos. Algunas de las inscripciones fueron escritas en un estilo denominado "boustrephedon Griego" por la semejanza en "la forma de un buey arando un campo", que significa que cada línea va en el sentido opuesto a la línea precedente, aún en el antiguo Egipto o entre las antiguas tribus de Canaan, donde la escritura fue inscrita en piezas de cerámica (cascos), piel de animal (vitela) o papel primitivo (papiro).

En las magníficas inscripciones observadas en los rollos del Mar Muerto, que datan del año 200 A.C. al 70 D.C., hay amplias variaciones en el número de caracteres en una línea, el ancho y número de líneas por página y ciertamente en el tamaño de los caracteres en un estilo manuscrito individual.

Mientras tanto el pueblo Semítico redujo las molestas silabarias (que tienen un signo separador para cada sílaba), que típicamente incluían 440 ó más signos separadores, contrariamente a lo que está comúnmente establecido, sin embargo, ellos nunca desarrollaron un alfabeto. (Aún los 22 signos de la escritura Hebrea, no son un alfabeto). El primer alfabeto donde todas las consonantes y vocales fueron representadas gráficamente por un y sólo un caracter, fue desarrollado por los Griegos alrededor del año 1200 A.C.

En el plazo de otros 400 años, los griegos a lo largo de la costa de lo que ahora se conoce como Líbano y aquellos que vivieron en Egipto, donde escribieron su alfabeto en el papiro, aparentemente aún usaban rollos para escribir libros. Después de la era cristiana, alrededor del año 65 D.C., los libros en forma de folio común con páginas cuadradas unidas en el margen izquierdo, comenzaron a aparecer.

Existen varias teorías acerca de este hecho, pero la más común de todas es probablemente aquella que relata que los primeros cristianos buscaban hojear para adelante y para atrás sus libros para la comparación de textos. Cualquiera que sea la razón, el libro en un tamaño que pudiera fácilmente ser llevado en las manos con páginas que pudieran ser volteadas, fue por el año 100 D.C.

El formato de esas páginas inició el desarrollo de lo que hoy se conoce como bloque de caracteres a 6 x 9 pulgadas y líneas escritas equitativamente hasta el final del tamaño de página, pero sin altos entre palabras, sólo al final de las sentencias y con una letra inicial o alargada al comienzo de los párrafos o secciones.

Con otros dos siglos, la longitud de línea fue desarrollada y se quitaron los espacios al final de las sentencias, con otras facilidades, tales como los nombres divinos siendo escritos en un estilo diferente. La página de papiro Griega del siglo II es muy familiar hasta nuestros días. En los siguientes 200 años, las páginas de dos y tres columnas empezaron a aparecer.

Una muy interesante facilidad es la justificación del texto del lado derecho del bloque escrito. Esto no fue realizado en las imprentas sino hasta que Caslon lo realizó en el siglo XVII. Un magnífico ejemplo de formato de página del Código Griego es visto en el manuscrito Bíblico llamado SINAITICUS, debido a su origen en el Monasterio de Santa Catalina en el Desierto Judío (Sinai), ahora en el Museo Británico.

Hasta mediados del siglo XV, los autores, quienes escribían los libros y los escribas, quienes los copiaban, todos pensaban en términos gráficos de la página, pero cuando Gutenberg empezó a experimentar con el primer tipo movible, tuvo que concentrarse en letras individuales y líneas simples.

En el siglo XIX cuando la energía eléctrica fue introducida a las imprentas y la máquina Linotipo fue inventada, pero haciendo aún el énfasis en el caracter, la palabra y la línea. No fue sino hasta la introducción de la imprenta ayudada por computador que el énfasis se regreso a donde había estado hace 500 años; la página. El embate de la impresión electrónica ha llevado un nuevo crecimiento de interés en diseñar la página.

1.1.2 El advenimiento de las computadoras

La historia de las computadoras va desde los antiguos egipcios del período medio, alrededor del 1800 A.C., pero los griegos clásicos, avanzaron en el arte alrededor del 450 A.C. La innovación de los computadores realmente inició con la búsqueda de un método por el cual los cálculos aritméticos y geométricos pudieran ser representados y procesados por medios mecánicos.

Una larga línea de dispositivos de barras mecánicas, ruedas, deslizamientos y engranes fueron desarrollados con el paso de los siglos. Quizá una de las más exitosas fue la Máquina de Pascal, inventada por el matemático francés Blaise Pascal (1623-1662) y versiones de su ingenioso dispositivo manejado por engranes estuvieron en continua producción hasta mediados de los años 60's de nuestro siglo.

Por casi 75 años después de la introducción de la electricidad como instrumento para proporcionar potencia alrededor de 1880, máquinas sumadoras mecánicas y comptometros (una máquina que podía multiplicar, dividir y hacer problemas de interés compuesto) fueron alimentadas por electricidad para su funcionamiento. Pero el uso de la electricidad para transmitir una señal analógica apareció con el exitoso telégrafo de Samuel F.B. Morse (1791-1872) quien inició experimentando con un dispositivo magnético en 1832 y patentado en 1837.

Muchos inventores buscaron desarrollar algún dispositivo para producir la salida de la señal analógica transmitida por el telégrafo como un mensaje impreso a la velocidad en que fue transmitida. En 1874 Thomas A. Edison (1847-1931) llevo tres aplicaciones de patente intituladas "Telegrafía Química". Estas patentes describen métodos para producir marcas sobre papel tratado químicamente por medio de una aguja y un electrodo conductor de soporte que proporcionaba un camino a través del medio de grabación, entonces causaba que una marca roja o azul apareciera por efecto electromecánico en el papel tratado.

En 1876, la Pluma Eléctrica patentada por Edison, que usaba corriente eléctrica directa para vibrar un punto de aguja metálico sobre una pluma para hacer agujeros microscópicos en papel y así formar un stencil, el cual era usado con anilinas que teñían para producir numerosas copias de un documento. Este instrumento fue el predecesor del moderno electrógrafo (que no usaba luz), para el proceso de impresión de no impacto.

Iniciando 1884, un hombre llamado H. Van Hoevenbergh de la compañía Baltimore & Ohio Telegraph Company, recibió cerca de 22 patentes para imprimir señales telegráficas en una variedad de configuraciones, de esta manera fue el telégrafo el que promovió la invención de un método para convertir una señal analógica transmitida en forma eléctrica a imágenes impresas.

En los primeros 75 años del siglo XX, un sistema para imprimir señales transmitidas fue desarrollado y construído, más comúnmente usado en Europa que en América, denominado "Sistema Telex". Pero este hecho fue más aparente con la invención y aplicación de la computadora.

La primera computadora analógica no apareció hasta la Segunda Guerra Mundial y la primera computadora digital al final del mismo período. Las computadoras crecieron en capacidad conforme reducían su tamaño y precio desde los 50's y comenzaron a moverse de las aplicaciones de ciencia y tecnología a los negocios en la década de los 60's.

Para los 70's, la actualización continua en las máquinas de escribir electrónicas y la adición de memoria de almacenamiento electrónico, en paralelo al desarrollo de pequeñas, rápidas y más interactivas máquinas de composición de tipografía, finalmente proporcionó el moderno procesador de palabras basado en computadora. Las versiones de hoy de los procesadores de palabras basados en microcomputadoras son capaces no solo de proporcionar tipografía sino también de procesar imágenes.

Cientos de años después de Gutenberg, hubo un esfuerzo por inventar algún sistema de composición instantáneo de la entrada directa de palabras. La computadora finalmente ofreció la forma en que esto podía ser realizado, pero el problema fue como encontrar una salida o sistema de imágenes, que tomara ventaja de la velocidad de las computadoras y la habilidad de producir textos y gráficos. La respuesta fue encontrada con la invención de una forma para convertir la imagen electrónica virtual o latente producida por la computadora a una imagen visual.

Un gran número de métodos fueron inventados, cada uno con un grado variable de éxito. El trabajo fue iniciado después de la Primera Guerra Mundial y O. Van Bronk patentó un proceso usando corriente fotoeléctrica para activar papel fotosensible. Esta invención fue patentada en Bretaña en 1922.

La aplastante característica de la salida de la computadora fue su inconstancia, esto es, cada línea o caracter puede ser impreso en tal forma que el texto podría variar. Desde finales de los 50's, muchas innovaciones se hicieron presentes en las cada vez más rápidas máquinas de impresión analógica, mecánica, o de impacto. Muchos dispositivos probados usaban múltiples cabezas, ruedas rotantes y aún bandas metálicas en un esfuerzo por escribir en papel el torrente de caracteres alfanuméricos producido por las rápidas computadoras.

Estos dispositivos usaban papel continuo y toda la salida de computadora impresa durante aproximadamente 22 años de 1955 a 1977, fueron realizadas en impresoras de impacto. Pero un gran número de fabricantes de computadoras y máquinas para negocios iniciaron trabajos para imprimir la salida de las computadoras en dispositivos alternativos.

En 1938, Chester Carlson inventó el proceso de copiado electrofotográfico a partir de dos fenómenos naturales ya conocidos: aquellos materiales con cargas opuestas se atraen y ciertos materiales se convierten en mejores conductores de electricidad cuando son expuestos a la luz. Con una combinación única de estos fenómenos se creó un nuevo proceso para elaborar copias en papel.

Algunas modificaciones al proceso de copiado electrofotográfico inventado por Carlson, denominado xerografía apareció como una alternativa para la impresión. De esta manera, la xerografía resultó ser la solución más viable de impresión y la más promisoria técnica para proyectar una imagen fue ciertamente el rayo láser (siglas de Light Amplification by Stimulated Emmision for Radiation).

1.1.3 Los primeros dispositivos de impresión electrónica

Muchos de los grandes fabricantes de equipo electrónico y de oficina experimentaron con varias formas en que una imagen electrónica pudiera ser capturada y fijada en papel. Mientras IBM había empezado a mirar el potencial antes de la Segunda Guerra Mundial, fue Xerox quien empezó a tomar ventaja del desarrollo de Chester Carlson de la electrofotografía. Después de años de desarrollo de la copiadora electrofotográfica en el Instituto Batelle en Cleveland, Ohio.

Xerox presentó el Sistema de Impresión Electrónica 9700 en 1977, equipado con un DEC 11/34 como controlador y corría estrictamente fuera de línea, con cintas magnéticas con densidad de grabación de 1600 bits por pulgada. Estos desarrollos han sido generados para la mayor parte de los computadores de IBM, sin embargo muchos desarrolladores de dispositivos de conexión han aparecido para permitir que los Sistemas de Impresión Electrónica sean conectados en todas las plataformas de cómputo existentes del mercado.

Los Sistemas de Impresión Electrónica utilizan para su funcionamiento la tecnología de rayo láser para proyectar una imagen, el proceso básico de impresión láser (Figura 1.1) se describe a continuación:

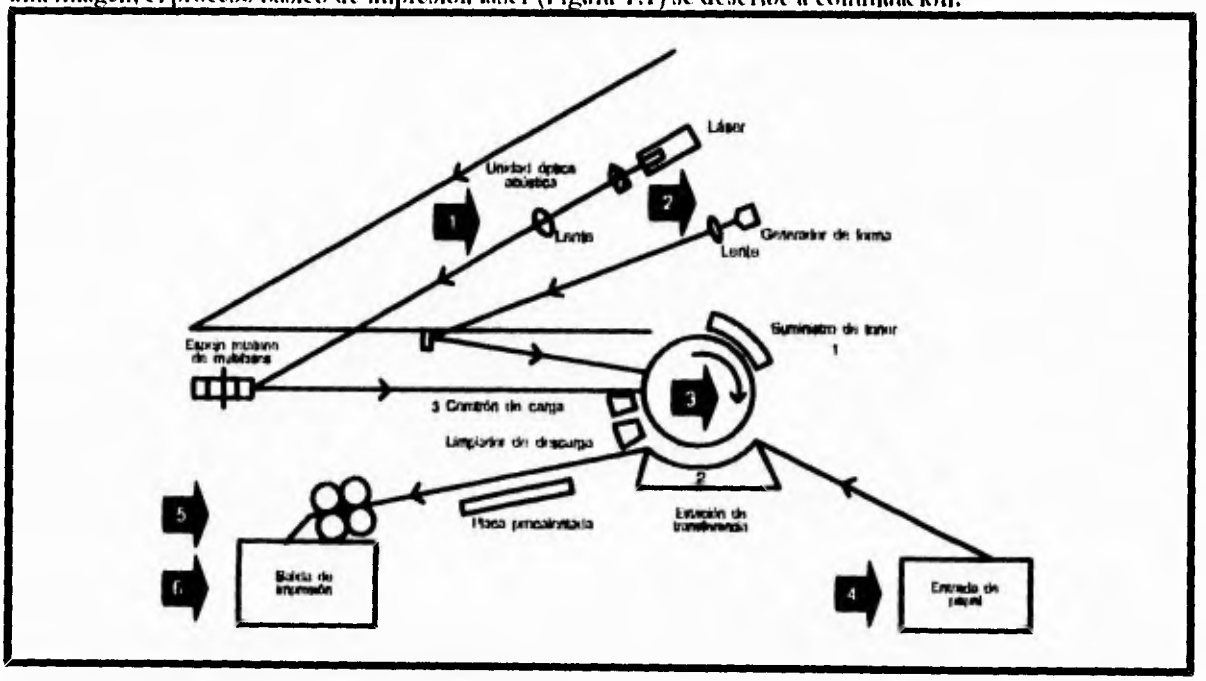


Figura 1.1 Proceso de Impresión con Láser

1.- La superficie de un tambor xerográfico es sensibilizada mediante carga electrostática negativa.

2.- Se hace incidir un rayo láser en un espejo, el cual refleja la imagen de forma invertida en el tambor (selenio), esto hace que se genere una imagen electrostática virtual, la imagen virtual generada por el tambor está formada por cargas electrostáticas positivas.

3.- Se "revela" la imagen virtual para obtener una imagen real, esto se logra con finas partículas de "toner" (tinta seca en polvo) cargadas negativamente, las cuales son atraídas por las cargas de la imagen virtual del tambor.

4.- Se transfiere la imagen revelada a una hoja de papel, cuya superficie previamente ha sido cargada positivamente; es decir, la imagen cargada negativamente con el "toner" es atraída por las cargas contrarias del papel.

5.- Finalmente la imagen es fijada en el papel por medio de un proceso de fusión, generalmente mediante presión y calor.

Una vez terminado el proceso, el tambor es limpiado; es decir, se descarga y carga la superficie para volver a repetir el proceso.

1.2 Justificación del producto

Los datos o información proveniente de un equipo de cómputo debe ser impresa en papel, este papel puede estar o no pre-impreso, lo cual se realiza en una imprenta o algún otro método de impresión de gran escala. Esto significa que contiene un formato previo; donde la información proveniente es colocada.

Además la impresora debe contar con el adecuado espaciamiento entre renglones, para que los datos coincidan con el formato; a este espaciamiento se le conoce como VFU (Vertical Format Unit). De esta forma el operador de la impresora debe colocar la forma pre-impresa y la cinta de control de carro, si es que se usa en el dispositivo de impresión, antes de que el reporte o la información sean impresos.

Esto ocasiona desperdicio en las hojas que tienen las formas, ya que muchas veces se deben realizar pruebas de alineación antes de obtener la información impresa de producción. Por otro lado, si la información debe obtenerse con copias, la información que se obtiene después de la 3ra copia resulta con muy mala calidad y poco legible.

La llegada del primer Sistema de Impresión Electrónica a los ambientes de cómputo, trajo consigo un cambio en la operación de los centros de procesamiento, particularmente en el subcentro de impresión, ya que este nuevo sistema permitía eliminar la compra de papel pre-impreso con las formas necesarias.

Lo anterior fue posible debido a que el primer Sistema de Impresión Electrónica cuenta con un lenguaje para describir las formas, propio del sistema, las cuales serán impresas junto con la información procedente del computador, realizando una mezcla electrónica de forma y datos. Además la forma descrita con el lenguaje es almacenada en el disco interno del Sistema de Impresión Electrónica.

De igual manera, el control del trabajo de impresión de datos que provienen del computador se describe con un lenguaje descriptor de impresión, propio del sistema, donde se definen las características, incluyendo la forma necesaria para el reporte y cómo se desea que la información aparezca. Asimismo el control de trabajo se almacena en el disco interno del sistema.

Así el operador del Sistema de Impresión Electrónica necesita indicar en la consola de operación propia del sistema, el nombre del descriptor de impresión que se debe utilizar con los datos que provienen del sistema de cómputo.

Sin embargo, lo anterior es válido sólo para los Sistemas de Impresión Electrónica que cuentan con un módulo controlador (el cual es un sistema de cómputo dedicado a la impresión) y módulo impresor; ya que en otro tipo de dispositivos de impresión, el módulo controlador no es proporcionado, de tal manera que la descripción de formas y de impresión debe realizarlas el procesador central.

En algunos ambientes de cómputo existen productos de software para la descripción de formas y de impresión, que deben adquirirse cuando se adquiere un dispositivo de impresión sin módulo controlador, los que deben residir y competir por recursos de cómputo del sistema central.

Es importante mencionar en este momento que los Sistemas de Impresión Electrónica pueden clasificarse de acuerdo al volumen y capacidad de impresión en:

- Alto Volumen con velocidad de impresión de más de 50 páginas por minuto
- Medio Volumen con velocidad en el rango de 20 a 50 páginas por minuto
- Bajo Volumen con velocidad abajo de las 20 páginas por minuto.

En los Sistemas de Impresión Electrónica de Alto Volumen podemos encontrar en el mercado dispositivos sin módulo controlador o con él, que permiten utilizar sus recursos para la descripción de formas e impresión, tal es el caso de Xerox.

En los Sistemas de Medio Volumen podemos encontrar dispositivos que tengan capacidad de almacenamiento, pero no cuentan con un verdadero lenguaje de descripción de formas, ya que es posible realizarlas con secuencias de escape, tal es el caso de Xerox y Hewlett Packard, o sin la capacidad de almacenamiento y sin la posibilidad de describir formas.

En los Sistemas de Bajo Volumen, que prácticamente son de escritorio, al igual que en el Medio Volumen, la descripción de formas se realiza por medio de secuencias de escape, como en el caso de Xerox y Hewlett Packard, pero no es posible la descripción en algunos otros sistemas.

En el caso de los Sistemas de Medio Volumen que cuenten con capacidad de almacenamiento, disco interno, la descripción de formas puede ser almacenada en los mismos, así como los fonts que se utilizan en las formas, y el sistema cuenta con la capacidad de adquirir el control de la impresión, sin necesidad de que el computador lo realice.

Sin embargo en los sistemas de Bajo Volumen, los fonts que son usados y la descripción de formas en secuencias de escape, deberán ser cargados en la memoria RAM del equipo antes de enviar los datos que serán impresos y en este caso, los datos deberán contener las secuencias de escape que controlarán la impresión. Como se puede observar, cuando se tiene un sistema de Bajo Volumen el operador debe realizar trabajo previo al envío de datos de impresión, esto se observa del siguiente proceso:

Programador:

Realiza un archivo con comandos en Secuencias de Escape, indicando las posiciones en puntos donde deben ir los textos y las líneas necesarias de la forma, si se incluyen logotipos o imágenes, estas deben ser colocadas en la forma, de tal manera que el equipo de impresión pueda imprimir la forma.

Realiza cambios en los programas que generan impresión, para incluir las Secuencias de Escape necesarias que permitan imprimir la forma con los datos.

Genera un programa o proceso batch que permita cargar los fonts o tipografía que utiliza la forma, uno por uno y la forma en el equipo de impresión y los datos de impresión conteniendo las Secuencias de Escape.

Usuario:

Cuando desea imprimir los archivos que requieren de una forma específica, debe correr el programa o proceso batch para cargar la impresora con los fonts y la forma así como los datos de impresión. (Ver Figura 1.2).

La utilización de Secuencias de Escape varía de acuerdo a la marca y modelo del Sistema de Impresión Electrónica, ya que cada fabricante genera diversas maneras de realizar las formas y la tipografía, generando con ello que si en una instalación se tienen varios Sistemas de Impresión Electrónica de diversos fabricantes y los programas deben generar salida indistintamente en cualquier equipo de impresión, se deben generar varios archivos con diseño de forma, para el equipo de impresión adecuado, así como el proceso de carga de tipografía y forma, también deberá generarse para el equipo de impresión donde se desea.

Ahora bien, si se desea realizar alguna modificación a la forma, el trabajo es sencillo en un sistema de Alto Volumen, porque el lenguaje de descripción de formas es fácil de interpretar y de editar, sin embargo en los sistemas de Bajo Volumen, la interpretación de los comandos en secuencias de escape no es fácil, sobre todo si el que desea realizar las modificaciones no es la misma persona que la creó, llegando muchas veces a redescribir la forma con las modificaciones pedidas, creando con ello falta de oportunidad y haciendo menos productivo el proceso.

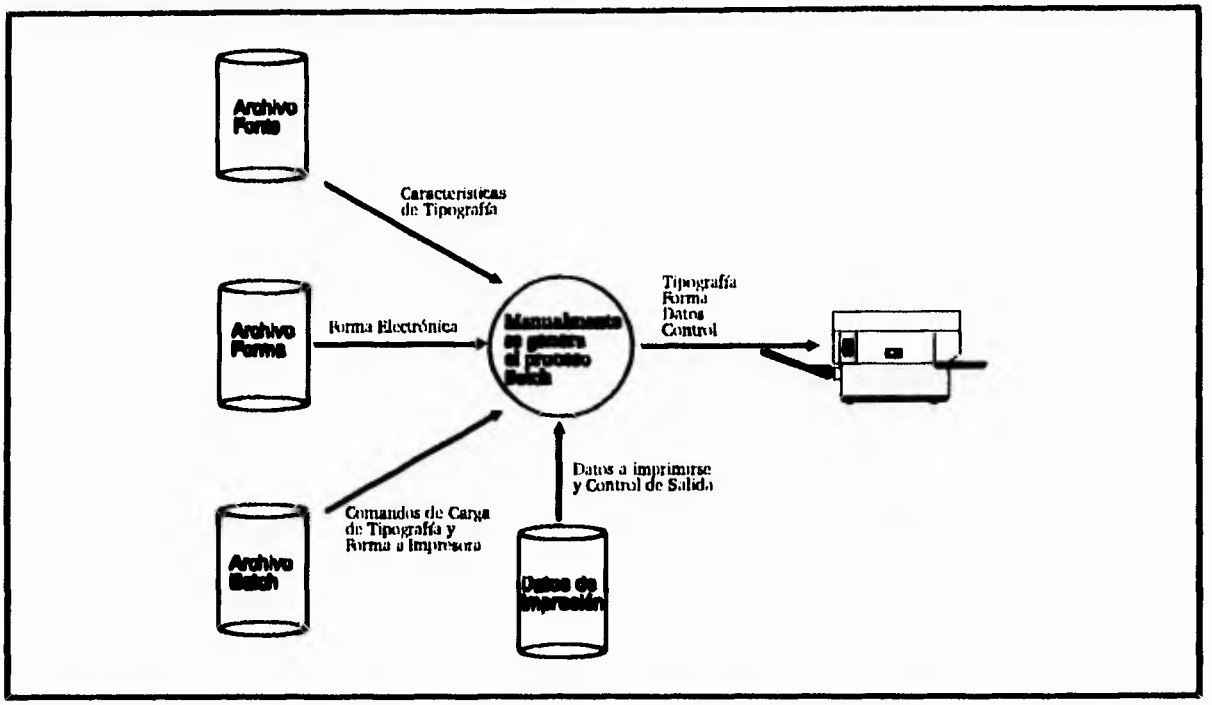


Figura 1.2 Proceso manual de envío a impresión

1.2.1 Características del sistema UNIX

El Sistema Operativo UNIX fue desarrollado por Bell Laboratories de los Estados Unidos, como una respuesta a las necesidades internas, para los proyectos de investigación que ahí se manejaban.

Los sistemas UNIX estándar son sistemas operativos multiprogramables de tiempo compartido, los cuales proporcionan un sistema de archivos jerárquicos con protección total, volúmenes desmontables, independencia de dispositivos y características que facilitan la sencillez de programación, se pueden observar las siguientes características principales de estos sistemas:

- Tendencia hacia la normalización (es un sistema abierto); es decir, es escalable, extendible, interoperable y compatible para uso en aplicaciones de la industria.
- Poseé comandos más elementales que un sistema operativo propietario.
- Interoperable con otros sistemas operativos.
- Fortaleza en comunicaciones.
- Tendencia como ambiente de desarrollo para herramientas de Ingeniería de Software.
- Facilidad para manejo de dispositivos.

- La documentación propia del sistema no es adecuada para los principiantes.
- No existe un procesador de palabras adecuado.

Sin embargo, algunos fabricantes de computadoras se han fijado en el creciente número de empresas que están adoptando los Sistemas Abiertos basados en UNIX, y que éstas buscan soluciones para transacciones en línea, basadas en estándares reconocidos por la industria.

Los programas del sistema UNIX se clasifican en:

- Nucleo

  Es el responsable de planificar las tareas y administrar el almacenamiento de datos

- Shell

  Es el programa que relaciona e interpreta las órdenes del usuario. Llama los programas a memoria y los ejecuta al mismo tiempo o en acoplamientos denominados "tubo".

- Programas de utilería

  Son los que ejecutan una variedad de rutinas y funciones especiales de mantenimiento del sistema.

Debido a que el sistema UNIX se diseño por y para programadores, ofrece diversas características que son fundamentales para el desarrollo de programas. Debido a este enfoque se puede observar que UNIX carece de características esenciales para el desarrollo de aplicaciones comerciales, principalmente porque las órdenes muchas veces son obscuras en su nombre y si son usadas sin saber su acción, fácilmente pueden dañar o eliminar archivos.

Actualmente los fabricantes de computadoras que ofrecen el sistema operativo UNIX estan agregando características de ayuda que permitan mejorar la relación sistema/usuario.

1.2.2 Definición del problema

El ambiente del Sistema Operativo Unix, un ambiente con amplio crecimiento debido a su utilización en los sistemas de arquitectura abierta, específicamente se ha enfocado en computadores de rango medio o minicomputadores, donde las necesidades de impresión son distribuidas y no se requiere de dispositivos de impresión de Alto Volumen; se considera que los sistemas de Medio y Bajo Volumen pueden cubrir esos requerimientos.

En este ambiente actualmente no existen productos de software que permitan describir las formas de una manera sencilla, sino que deben usarse secuencias de escape, y los programas que generan la salida de impresión deben modificarse para incluir las secuencias que controlan la impresión.

Todo el tedioso proceso mencionado anteriormente, pone las bases para que se implante un producto de software que permita describir las formas de una manera sencilla y fácil, además de poder ser editadas y modificadas sin perder oportunidad. Asimismo el producto debe eliminar la necesidad de modificar los programas que generan salida de impresión, aumentando la productividad y permitiendo el uso casi inmediato de los dispositivos de impresión.

Es importante mencionar que el producto que se desea desarrollar debe cumplir con los siguientes requisitos:

- Portabilidad

  El producto deberá tener la capacidad de ser transferido de un equipo de cómputo a otro dentro del ambiente del Sistema Operativo UNIX, no importando la versión que pudiera encontrarse en los equipos.

- Confiabilidad

  El producto debe realizar las funciones requeridas en los puntos principales que se identificaron.

- Escalabilidad

  El producto se debe realizar en una base modular, que permita agregar más funcionalidad en la etapa de mantenimiento.

- Versatilidad

  El producto deberá realizar las funciones en una manera que compita por muy pocos recursos computacionales con un mínimo de errores y si es posible sin ellos, y continuar operando, aún cuando los datos proporcionados por el usuario sean inválidos.

- Facilidad de operación

  El producto deberá ser fácil de usar, sin requerir un entrenamiento exhaustivo, ya que el manual de usuario deberá ser suficiente herramienta de apoyo.

1.2.3 Estrategia de solución

Se propone realizar un producto de software que resida en el sistema computador con Sistema Operativo UNIX, que permita manejar eficientemente los recursos de impresión en un ambiente de desarrollo y producción, que proporcione aumento en la productividad y flexibilidad en el cambio, de tal manera que su uso sea sencillo y ágil, con las siguientes capacidades:

- Adquisición de fonts

  Los cuales serán utilizados en la descripción de formas y sean descargados al Sistema de Impresión Electrónica cuando la forma sea utilizada.

- Describir formas

  De una manera sencilla, utilizando lenguaje convencional, en lugar de secuencias de escape, facilitando su edición y modificación, que permita incrementar la productividad.

- Mantenimiento de impresoras

  Que permita al usuario del producto establecer las características de los dispositivos de impresión, como: el nombre del dispositivo, tipo y puerto de comunicación y algunas otras características importantes relacionadas con los dispositivos.

- Controlar la impresión

  Que evite la modificación de los programas de usuario que generan salida impresa, el controlador debe insertar los adecuados comandos de control.

- Interfase de usuario

  Que permita al usuario del producto especificar qué archivo será impreso, la forma que será utilizada en la impresora destino y el número de copias requeridas.

- Mantener registro

  De los archivos de fonts que se encuentran en el dispositivo de almacenamiento del computador, así como de las formas que han sido desarrolladas y algunos otros parámetros del sistema.

Sin embargo, es importante mencionar que existen ciertas restricciones que se deberán tomar en cuenta para realizar el producto, éstas son:

- La operación del producto y la descripción de formas deberán ser proporcionadas para cubrir los Sistemas de Impresión Electrónica de bajo Volumen, específicamente Xerox y Hewlett Packard.
- El producto deberá operar en un Ambiente de Sistema Operativo UNIX
- El lenguaje para descripción de formas deberá proporcionar las facilidades necesarias para realizar formas sencillas que incluyan únicamente líneas, cajas y texto.

- Deberá ser capaz de adquirir inicialmente fonts para ser usados con Sistemas Xerox.
- Se deben proporcionar características de impresión simplex (una sola cara de la hoja).

De acuerdo a la definición del problema y a las características que el producto debe cubrir, mencionadas anteriormente, se propone sea desarrollado un producto que proporcione una solución integral al problema planteado, de esta forma la solución debe comprender el diseño y construcción de:

- Un lenguaje de descripción de formas
- Un compilador para el lenguaje de descripción de formas
- Un programa e interfase de usuario para definir impresoras
- Un programa e interfase de usuario para el envío de la impresión y control para la mezcla de formas con datos de impresión
- Un programa e interfase para la adquisición de fonts que se almacenarán en el sistema de cómputo.

1.2.4 Plan de proyecto

El modelo del ciclo de vida que se sugiere sea llevado en este proyecto es el tradicional, que consta de las siguientes fases:

- Análisis de requerimientos

  Logrando con ello integrar un documento que permita especificar los requerimientos necesarios para la producción del producto y un manual de usuario preliminar

- Diseño del producto

  Al final de esta etapa deberemos obtener el diseño estructural, el diseño detallado y el manual del usuario.

- Instrumentación o codificación

  En esta etapa deberemos tener el producto listo para las pruebas del mismo.

- Pruebas

  Para verificar que el producto sea consistente y completo respecto a los requisitos y al diseño.

El grupo de trabajo se estructurará en la forma Democrática con Guía de Desarrollo, ya que esta estructura brinda la facilidad de los productos que sean realizados en cada fase se discutan abiertamente y sean examinados por todos los miembros o participantes. Esto proporciona a que cada uno de los miembros contribuya y aprenda de los otros, además de que el grupo trabajará mejor en un ambiente de abierta comunicación. El miembro que se denomina Guía de Desarrollo, es la persona que sabe y conoce más del tema.

Los requerimientos de personal que debe intervenir en el proyecto se obtiene de las características de complejidad del mismo, si tomamos en cuenta las funciones nos inclinamos por determinar que este producto está clasificado como Programa de Apoyo, y se espera que se obtengan alrededor de 5000 líneas para su realización (KDSI), por lo tanto para determinar el número de meses de programador (PM) requeridos para este producto, lo obtenemos de la ecuación de Boehm.

$$PM = 3.0 * KDSI^{1.12}$$

$$\mathbf{PM = 3.0 * 5^{1.12}}$$

PM = 18.19 meses de programador

El tiempo de desarrollo (Td) para este producto se calcula en

$$\mathbf{Td = 2.5 * PM^{0.35}}$$

$$\mathbf{Td = 2.5 * 18.19^{0.35}}$$

Td = 6.9 meses de desarrollo

Los resultados de las anteriores ecuaciones nos sirve para calcular el número de programadores o participantes para este proyecto de:

$$\#\ Prog. = \frac{PM}{Td}$$

$$\#\ Prog. = \frac{18.19}{6.9}$$

 # Prog = 2.6 programadores o participantes del proyecto

Sin embargo el tiempo de desarrollo (Td) que se tiene negociado para este proyecto está restringido a 3 meses, no se considera que el proyecto sea realizado en los 7 meses de desarrollo que se calculan, de esta manera debemos obtener el número de programadores o participantes en el mismo de acuerdo con el tiempo establecido:

$$\#\ Prog. = \frac{18.19}{3}$$

 # Prog. = 6.06 participantes en el proyecto

Es importante mencionar que el diseño e instrumentación del proyecto pueden ser realizados en forma modular, debido a las características de la estrategia de solución planteada.

El esfuerzo relativo que se tendrá en este proyecto, por cada uno de los participantes en el proyecto, indicará el esfuerzo que los mismos deberán trabajar en forma continua, lo obtenemos de la ecuación de Putnam, que se menciona a continuación:

$$E = \frac{K}{Td^4}$$

Donde:

K = meses de programador calculado
Td = tiempo de desarrollo

por lo tanto tenemos que el esfuerzo relativo total de los integrantes del proyecto es:

$$E = \frac{18.19}{3^4}$$

E = 23.93 meses de programador

De esta forma el esfuerzo relativo individual de cada uno de los participantes del proyecto será:

E_I = 3.98 meses de programador

Los recursos materiales necesarios para el desarrollo de este proyecto en las fases de análisis, diseño y desarrollo o instrumentación, es una computadora personal por cada integrante del equipo, 2 Sistemas de Impresión Electrónica (Xerox y Hewlett Packard) y un Sistema de Ambiente UNIX.

Las herramientas y lenguajes de programación que deberán usarse en el proyecto son: herramienta de análisis CASE, lenguaje de programación C.

La documentación necesaria para tener éxito en el logro del producto son; manuales de XES (Xerox Escape Sequences), manual de PCL 4 (Print Control Language versión 4), manuales del Sistema Operativo Unix y documentación referente a Ingeniería de Software, Compiladores, etc.

Página dejada en blanco intencionalmente

ANALISIS DE REQUERIMIENTOS

Para la creación de cualquier producto de software, se requiere como parte fundamental el análisis de requerimientos. Dicho análisis consta de una serie de fases, las cuales indican en forma evolutiva los aspectos necesarios para la realización del sistema; en esta parte del proyecto se explicará en forma profunda cada una de las fases que se requieren para la conformación del producto.

2.1 Ambientes de desarrollo, operación y mantenimiento

Los ambientes de desarrollo, operación y mantenimiento son importantes de definir, ya que en éstos se instrumentará el proyecto, por tal motivo se explicará cada uno con la finalidad de que el producto que se desea realizar, tenga un sustento sólido y permita en las etapas posteriores una facilidad en su realización.

Actualmente existen microcomputadoras denominadas "Compatibles con IBM", dichos equipos se han generalizado enormemente, por lo que es relativamente fácil contar con uno de ellos, asimismo la operación de estas máquinas no representa un obstáculo para su utilización, aunado a lo anterior, existe una amplia gama de software desarrollado para que opere en estos equipos; básicamente son estas características las que han permitido decidir que se empleé durante la etapa de desarrollo equipos compatibles con IBM. El software utilizado en el desarrollo y en el equipo serán: el Sistema Operativo MS-DOS versión 5.0 y el Lenguaje de Programación C estándar de Microsoft.

Como se mencionó anteriormente, el lenguaje para desarrollar el producto es "C", que es un lenguaje de programación de propósito general muy bien adaptado a la programación estructurada. La idea de un lenguaje de programación de propósito general, no se refiere sólo a la orientación del lenguaje hacia algún campo de aplicación, sino a la ausencia de instrumentos específicos para realizar determinadas tareas y lo que consideramos más importante, a la ausencia de las limitaciones impuestas por los lenguajes de programación tradicionales. Lo anterior hace que "C" tenga una gran potencialidad: por una parte presenta un juego de instrucciones muy reducido, pero suficiente y fácil de aprender; por otra ofrece muy pocas limitaciones en cuanto a su utilización. Es por lo anterior que "C" es un lenguaje utilizado tanto para la instrumentación de sistemas operativos y lenguajes de alto nivel, como para la realización de utilerías y programas de aplicación.

Esto implica una vital independencia del software con respecto al hardware, y disminuye los costos de desarrollo de software, al ser amplio el espectro de equipos sobre los que puede funcionar un determinado paquete de aplicación.

El producto deberá operar bajo ambiente operativo Unix, y la razón principal, es que no existe un producto para el diseño y emisión de formas en impresoras láser de bajo volumen que opere bajo Unix, asimismo el proyecto deberá ajustarse a las definiciones y características enunciadas en el capítulo uno.

Como se mencionó anteriormente, el proyecto se realizará en microcomputadoras tipo PC, por la facilidad que ofrecen estos equipos; sin embargo, una vez desarrollado el proyecto, éste migrará a una plataforma de mayor dimensión para su operación. Cabe mencionar que como el desarrollo se realizará en "C", es muy probable que en la migración no se requiera de casi ninguna modificación, ya que una de las grandes características de "C" es su portabilidad.

Una vez operando el producto, se deberá realizar el mantenimiento conforme al ciclo de vida; este mantenimiento podrá efectuarse independientemente en cada uno de los módulos, de tal forma que facilite dicha actividad. Al ser modular el producto en su totalidad, el mantenimiento será fácil de realizar y podrá llevarse a cabo en cualquier momento .

2.2 Flujo de datos

El flujo de datos del proyecto planteado, es un requisito indispensable para un diseño bien fundamentado. La importancia de este punto repercute en todo el proyecto, por tal motivo, se desarrollarán a continuación los diagramas de flujo de datos y diccionarios de cada uno de los módulos que conforman el proyecto. La notación que se ocupará para los flujos de datos está basada en el método de Yourdon-DeMarco, el cual es el más conocido para el análisis estructurado y que se aplica fundamentalmente a sistemas que manipulan datos, principalmente con actividades de procesamiento en lugar de estructuras de datos, por lo que es a menudo referenciado como "modelado de procesos".

Como el producto a desarrollar es básicamente para el procesamiento de información que será enviada a una impresora, se decidió utilizar la metodología de Yourdon-DeMarco.

La figura 2.1 muestra la simbología utilizada por este método.

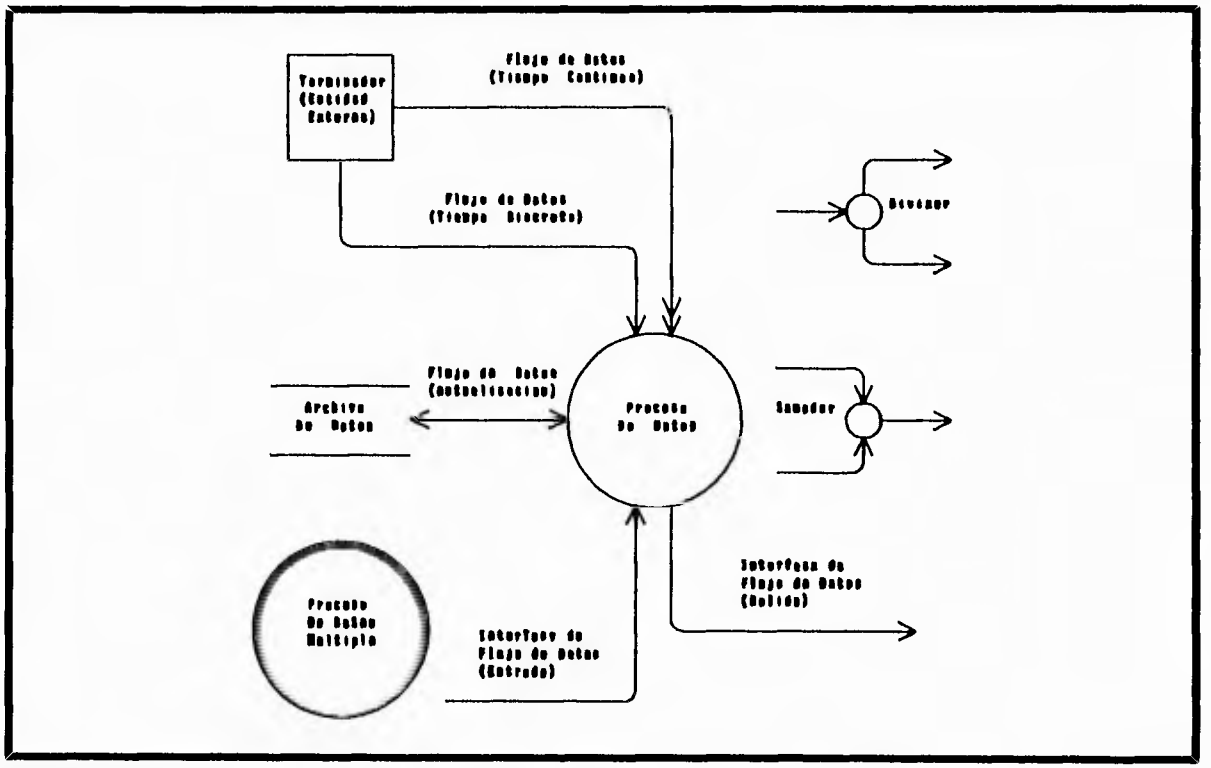


Fig 2.1 Simbología Diagramas de Flujo (Yourdon/DeMarco)

Como se describió en el capítulo I "Introducción", el presente trabajo estará constituido por los siguientes cinco módulos:

1.- Interfase del usuario para mantener impresoras

2.- Interfase del usuario para definir fonts.

3.- Compilador de formas.

4.- Interfase de impresión.

5.- Instalación del producto.

El proyecto integral consta de los módulos antes enlistados. La figura 2.2 constituye el diagrama de flujo de todo el proyecto y la expansión de cada módulo estará detallada en el anexo A.

2.2.1 Interfase del usuario para mantenimiento de impresoras.

El ojetivo principal de este módulo, es el mantener actualizado el archivo de impresoras en el sistema; el usuario podrá anexar una nueva impresora, eliminar alguna de las ya existentes o modificar los datos característicos de cada una de ellas, además podrá emitir un listado de las impresoras registradas en el archivo de impresoras.

La figura 2.3 muestra el módulo de impresión en un primer nivel, este diagrama indica todas las acciones que el módulo deberá realizar.

La primera acción que realizará este módalo, será presentar una lista de opciones en la línea de comandos, donde el usuario indicará la acción que desea realizar, si por alguna razón el usuario desea abandonar el módulo, existirá una opción de salida sin que se afecte la información contenida en el sistema. Si se desea anexar o agregar los datos de una nueva impresora, el módulo solicitará las características de la impresora a anexar, solicitando posteriormente al usuario la confirmación para anexarla; si por el contrario el usuario desea eliminar una impresora existente, el módulo mostrará las características de la impresora seleccionada y preguntará si en verdad quiere eliminarla, si la respuesta es positiva, se eliminará el registro apropiado en el archivo de impresora, en caso de que la respuesta fuese negativa no se borrará ningún registro y se terminaría la acción del módulo de interfase de impresoras. El usuario también podrá efectuar modificaciones a los datos característicos de las impresoras, así como emitir un listado de los equipos y características de los mismos.

Los datos que la interfase de impresoras deberá manejar son: nombre de la impresora, tipo (Xerox o Hewlett Packard), emulación de la impresora, velocidad de impresión, capacidad en la memoria, puerto de comunicaciones, velocidad de transmisión. Los elementos descritos, se consideran suficientes y necesarios para una buena operación del módulo.

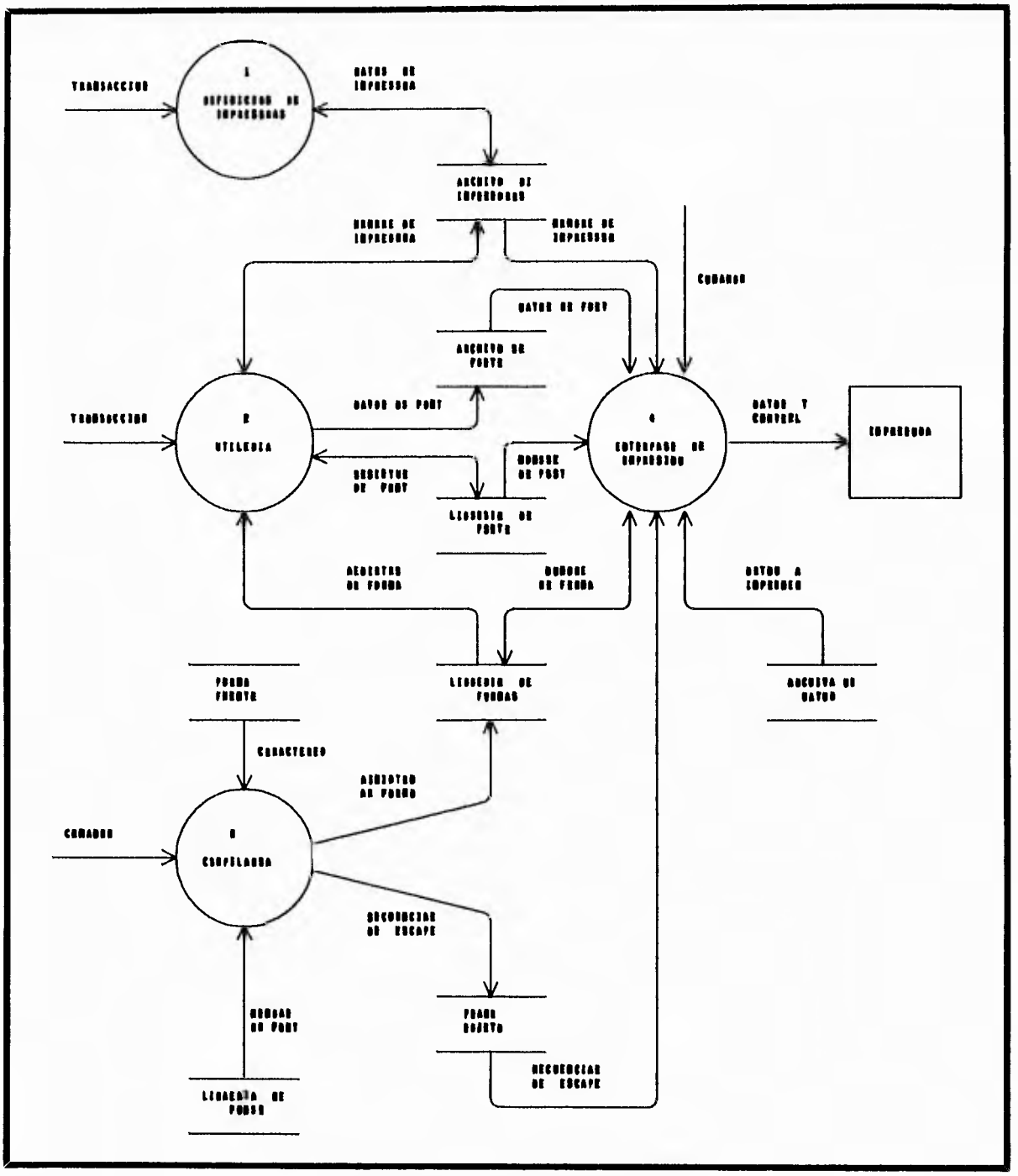


Fig. 2.2 Producto de Software

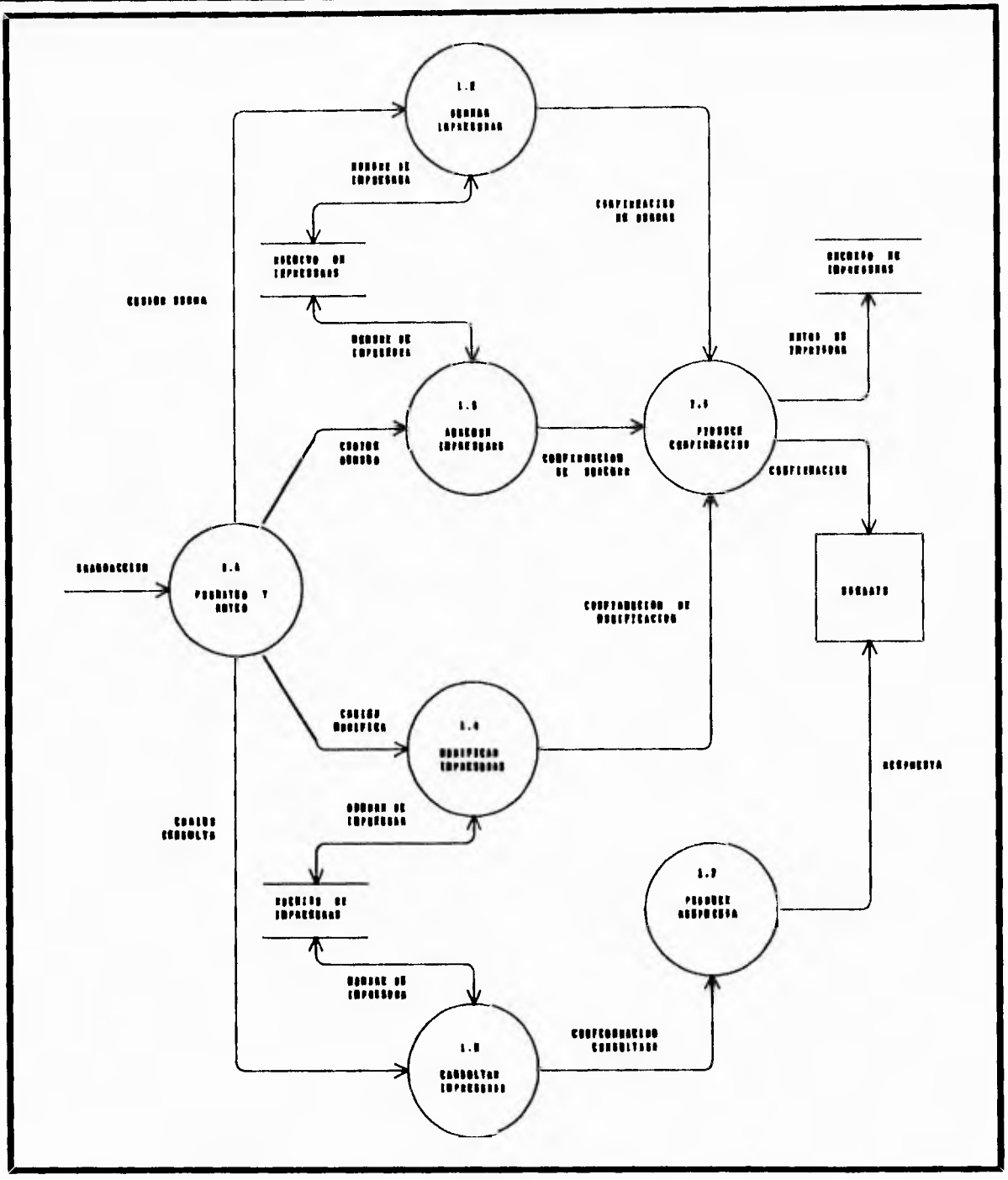


Fig. 2.3 Mantenimiento de Impresoras

2.2.2 Interfase de fonts.

La interfase de fonts tendrá como finalidad el anexar fonts en el sistema para su posterior uso en las impresoras, asimismo podrá proporcionar un listado de los fonts y formas existentes. La figura 2.4 muestra el diagrama de flujo en un primer nivel, de las acciones que deberá realizar la utilería de fonts.

El primer paso será solicitar al usuario la acción que desea realizar, la cual podrá ser: anexar un font, listar los fonts existentes en la librería de fonts o las formas de la librería de formas y salir del módulo de fonts.

Para poder llevar a cabo la primera acción, se debe contar con el disco que contiene los fonts comprados; el disco será leído por este módulo y se actualizarán los registros tanto del archivo de fonts como del archivo de librería de fonts.

Una vez leído el registro del disco, el programa actualizará los datos del registro de la librería de fonts, los cuales son: el tipo o familia de font, tamaño y peso; asimismo copiará en el archivo de fonts los diferentes datos que conforman la generación de un font. Si en la lectura del disco ocurriese un error como el que no estuviera ningún font, entonces el módulo mandará un mensaje de error.

La opción de listar los fonts existentes, sólo tendrá que leer del archivo de librería de fonts cada uno de ellos y listarlos por medio de la impresora. Se considera que dichas acciones son suficientes en este módulo, ya que no existe posibilidad de modificar los datos de un font existente, tampoco existe la posibilidad de eliminar fonts del archivo de fonts, ya que entre más fonts existan mejor será la amplitud en el diseño de la forma.

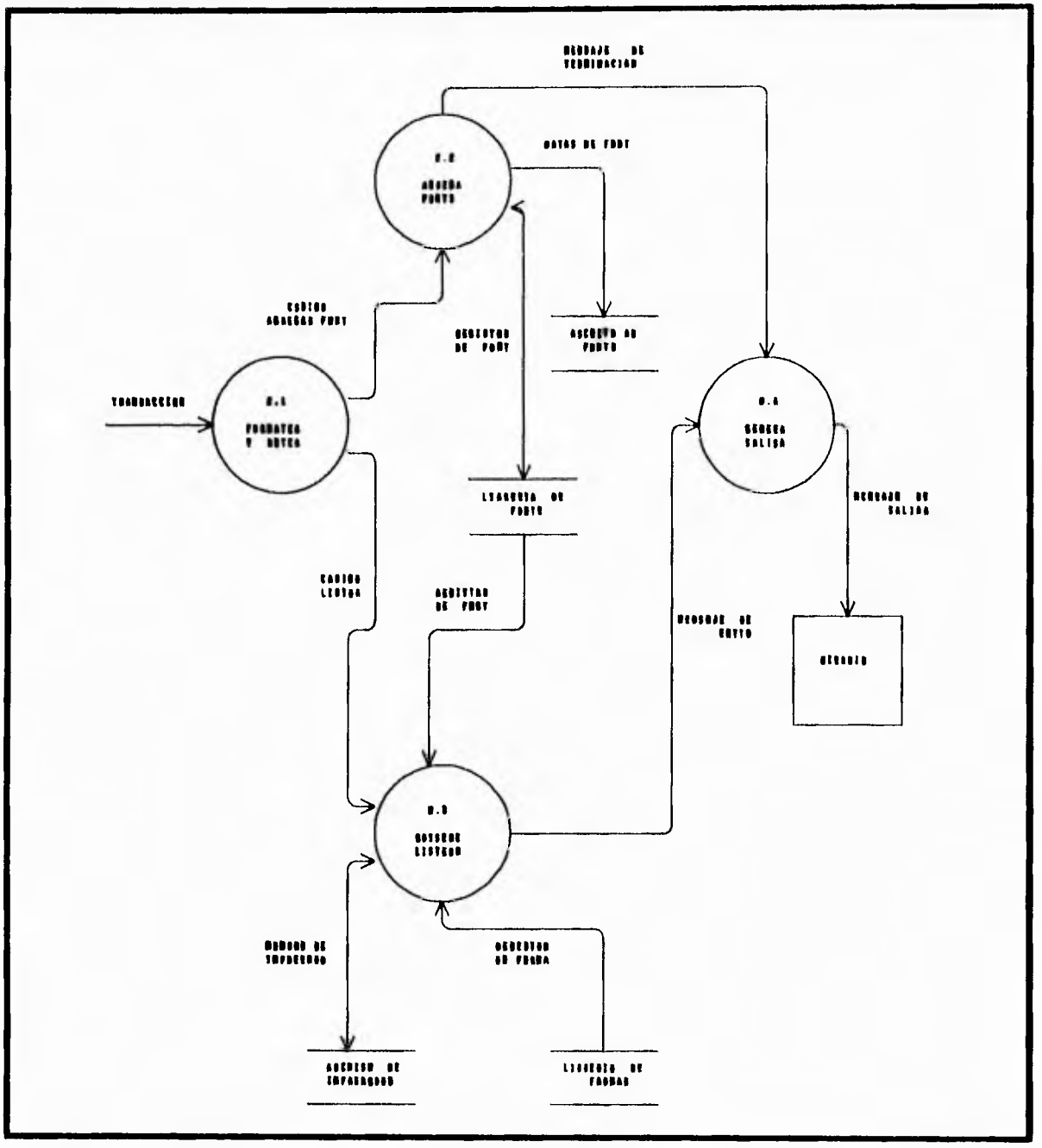


Fig. 2.4 Interfase de fonts

2.2.3 Compilador de formas

En esta sección se describe el diagrama de flujo del compilador, la figura 2.5 muestra cada una de las fases en que está constituido el compilador.

La primera fase es la del análisis léxico, es decir, analizar el programa fuente según las normas sintácticas adecuadas para realizar esta función; el analizador léxico (scanner) lee el programa fuente como una secuencia de caracteres y lo convierte en una secuencia de tokens, los cuales son comparados con una tabla interna que contiene las palabras reservadas, llamada Tabla de Palabras Reservadas. El resultado de la comparación antes mencionada puede definir que el token es una palabra reservada, si no lo fuera, se trata de un identificador o de una constante numérica.

Este procedimiento se realiza cíclicamente hasta que finaliza el programa fuente, lo que indica que la fase de análisis léxico ha concluido. Al terminar la fase se tendrá como resultado nuevas tablas, las cuales son: tabla de símbolos uniformes, que contiene todo el programa fuente representado numéricamente; la tabla de constantes numéricas utilizadas en el programa y la tabla de variables.

La fase siguiente es la del analizador sintáctico (parser), el cual tiene como objetivo verificar que la sintaxis este de acuerdo a las normas establecidas en el lenguaje. Esta fase se apoya en las tablas generadas en la fase anterior, además verifica la existencia de los fonts utilizados.

Una vez realizadas las fases anteriores, el compilador mediante la fase de analizador semántico, realiza las últimas acciones para crear una forma intermedia del programa fuente y agregar información complementaria a las diferentes tablas generadas.

En cada una de las etapas anteriormente descritas puede ocurrir un error, en tal caso el compilador deberá finalizar la etapa en que ocurrió el error, mandar un mensaje y terminar con el proceso de compilación.

La última fase del compilador, es la de generar el código. Dicha acción se realiza mediante las tablas generadas en las fases anteriores y con apoyo de una matriz de código para cada instrucción generada y verificada por las fases anteriores. Al término de todas las fases, el compilador debe generar el programa objeto o forma objeto, la cual estará constituida por una secuencia de instrucciones que sólo reconoce la impresora láser, asimismo debe registrarse en la librería de formas la existencia de una nueva forma.

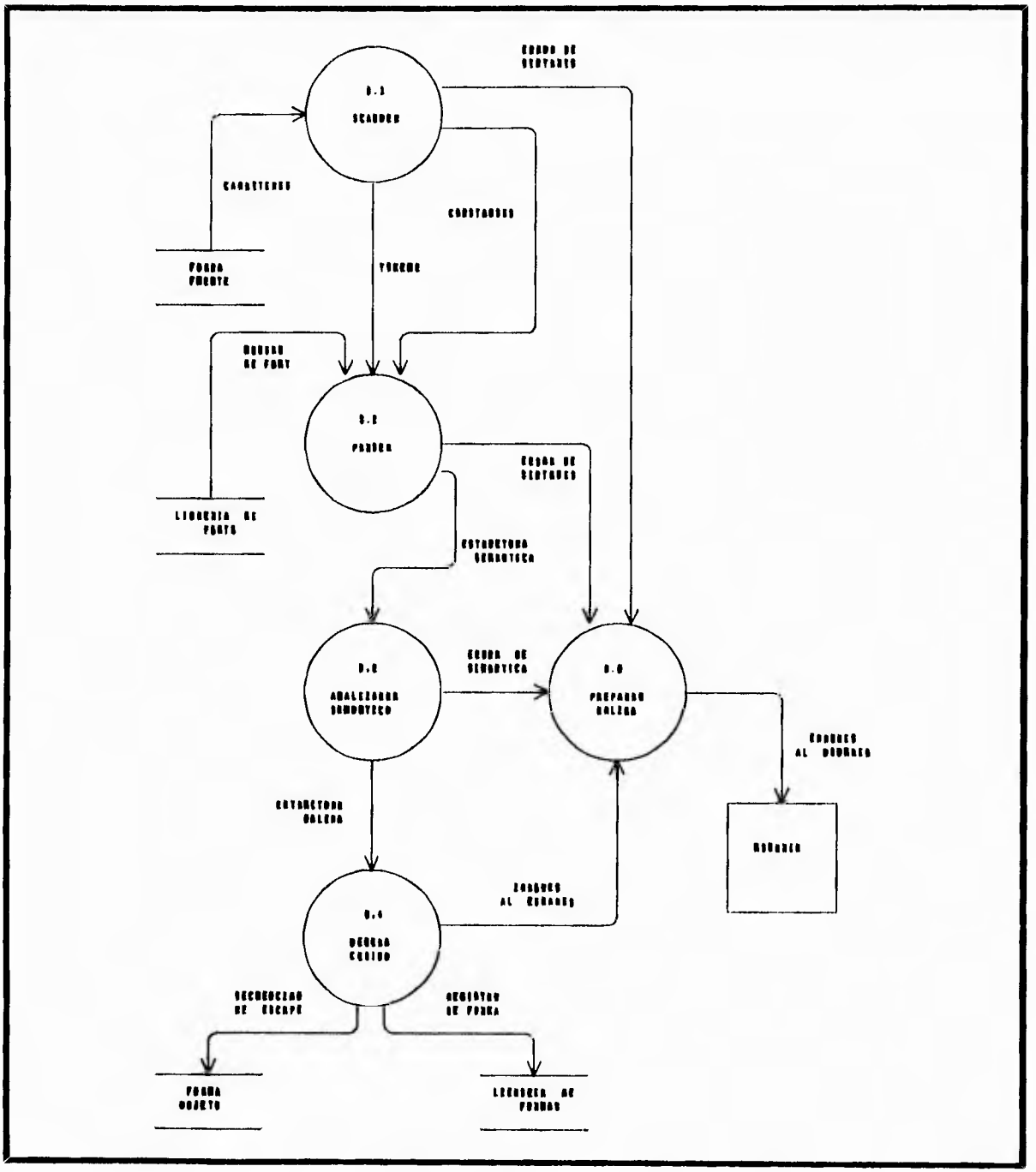


Fig. 2.5 Compilador de Formas

2.2.4 Interfase de impresión

La parte que deberá reunir a cada uno de los módulos antes descritos, será la interfase de impresión. La figura 2.6 muestra la forma en que este módulo realizará la acción.

El objetivo de la interfase de impresión, será enviar la forma diseñada por el usuario, con o sin datos variables, para lo cual deberá teclear o introducir un comando, el que será validado para verificar que esté correctamente escrito y destinado al módulo correspondiente, es decir impresión de archivo o ejemplo de forma. Posteriormente se verificará la existencia de la forma y la impresora en la cual se desea descargar la misma junto con los datos variables.

Cada uno de los módulos tomará las acciones necesarias para indicar el éxito o el error de las operaciones a través de mensajes al usuario y mediante la interacción con las librerías y archivos de fonts y formas. La forma se encontrará residente en memoria del equipo si es una impresora Xerox, o se generará cada vez que se imprima, si es que la impresora es Hewlett Packard.

2.2.5 Instalación del producto

Todo sistema debe ser instalado apropiadamente en el equipo o plataforma en donde se va a operar, por lo que es necesario generar un módulo que facilite dicha tarea.

La figura 2.7 muestra las acciones que realizará el módulo de instalación; la primera acción será dar una bienvenida y desplegar los derechos de autor al usuario, posteriormente serán solicitados los datos necesarios para personalizar el producto, estos datos constan del nombre del responsable de la instalación del producto, la fecha actual del sistema, la fecha de vigencia del producto y el número de serie del equipo en donde se instalará el producto. Lo anterior permitirá generar un archivo de control, en el que se grabarán los datos solicitados y permitirán generar una llave de acceso al sistema, asimismo se grabará el producto en el equipo deseado.

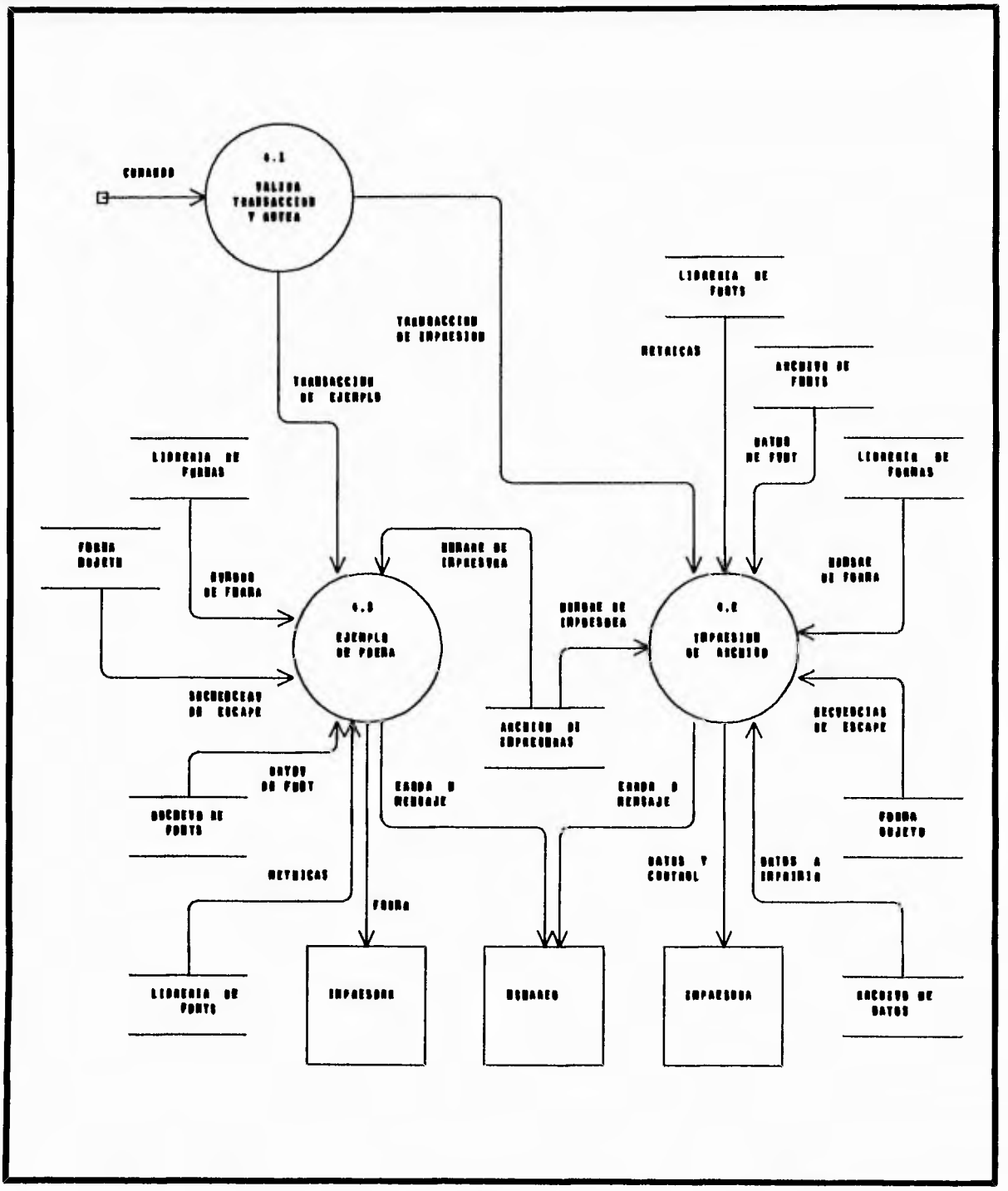


Fig. 2.6 Interfase de Impresión

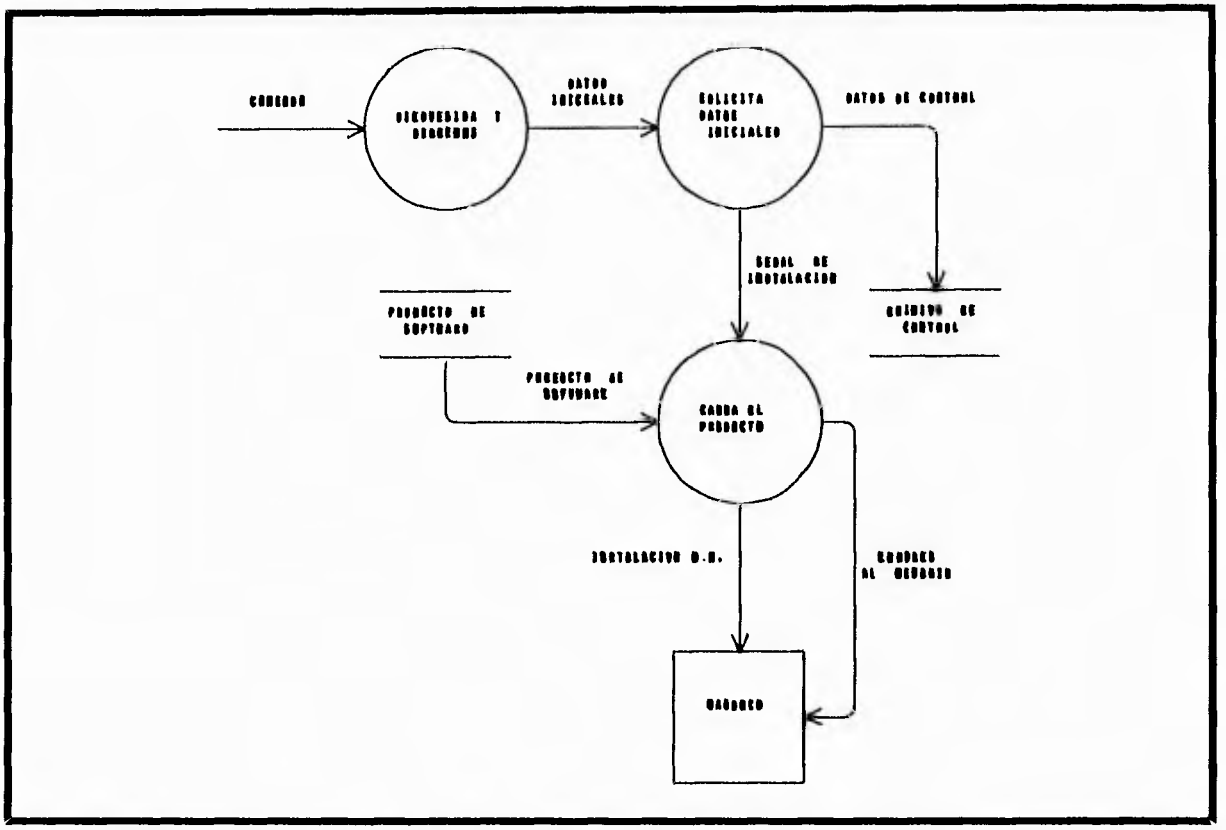


Fig. 2.7 Instalación del Producto

2.2.6 Diccionario de datos

Los objetos o datos que se obtienen después de haber realizado el análisis del producto con la herramienta de case EasyCase, se observan en la siguiente tabla:

IDENTIFICADOR DEL OBJETO	TIPO	ETIQUETA DEL OBJETO
ARCHIVO DATOS VARIABLES EXISTE	Data Flow	ARCHIVO O.K.
BORRAR ARCHIVO DE IMPRESION	Data Flow	BORRAR ARCHIVO DE IMPRESION
CARACTERES	Data Flow	CARACTERES
CODIGO AGREGA	Data Flow	CODIGO AGREGA
CODIGO AGREGAR FONTS	Data Flow	CODIGO AGREGAR FONT
CODIGO BORRA	Data Flow	CODIGO BORRA
CODIGO CONSULTA	Data Flow	CODIGO CONSULTA
CODIGO DE COPIAR FONT	Data Flow	CODIGO DE COPIAR
CODIGO DE ENVIO	Data Flow	CODIGO DE ENVIO

Tabla 2.1 Diccionario de Datos

IDENTIFICADOR DEL OBJETO	TIPO	ETIQUETA DEL OBJETO
CODIGO DE TERMINACION	Data Flow	CODIGO DE TERMINACION
CODIGO LISTAR	Data Flow	CODIGO LISTAR
CODIGO MODIFICA	Data Flow	CODIGO MODIFICA
COMANDO	Data Flow	COMANDO
CONFIGURACION A AGREGAR	Data Flow	CONFIGURACION A AGREGAR
CONFIGURACION A BORRAR	Data Flow	CONFIGURACION A BORRAR
CONFIGURACION A MODIFICAR	Data Flow	CONFIGURACION A MODIFICAR
CONFIGURACION CONSULTADA	Data Flow	CONFIGURACION CONSULTADA
CONFIRMACION	Data Flow	CONFIRMACION
CONFIRMACION DE AGREGAR	Data Flow	CONFIRMACION DE AGREGAR
CONFIRMACION DE BORRAR	Data Flow	CONFIRMACION DE BORRAR
CONFIRMACION DE MODIFICACION	Data Flow	CONFIRMACION DE MODIFICACION
CONSTANTES	Data Flow	CONSTANTES
DATOS A IMPRIMIR Y CONTROL	Data Flow	DATOS A IMPRIMIR Y CONTROL
DATOS DE CONTROL	Data Flow	DATOS DE CONTROL
DATOS DE FONT	Data Flow	DATOS DE FONT
DATOS DE FORMA	Data Flow	DATOS DE FORMA
DATOS DE IMPRESORA	Data Flow	DATOS DE IMPRESORA
DATOS INICIALES	Data Flow	DATOS INICIALES
DATOS VARIABLES	Data Flow	DATOS A IMPRIMIR
DATOS Y CONTROL	Data Flow	DATOS Y CONTROL
EOF	Data Flow	EOF
ERROR DE SEMANTICA	Data Flow	ERROR DE SEMANTICA
ERROR DE SINTAXIS	Data Flow	ERROR DE SINTAXIS
ERROR O MENSAJE	Data Flow	ERROR O MENSAJE
ERROR: ARCHIVO DATOS NO EXISTE	Data Flow	ERROR:ARCHIVO DE DATOS NO EXISTE
ERROR: FORMA NO EXISTE	Data Flow	ERROR:FORMA NO EXISTE
ERROR: IMPRESORA NO EXISTE	Data Flow	ERROR:IMPRESORA NO EXISTE
ERROR: IMPRESORA YA EXISTE	Data Flow	ERROR IMPRESORA YA EXISTE
ERROR:TRANSACCION DEF_IMP NO OK	Data Flow	ERROR TRANSACCION DEFINICIONIMPRESORAS NO OK
ERROR:TRANSACCION_UTILERIA NO OK	Data Flow	ERROR:TRANSACCION UTILERIA NO O.K.
ERRORES AL USUARIO	Data Flow	ERRORES AL USUARIO
ESTRUCTURA SEMANTICA	Data Flow	ESTRUCTURA SEMANTICA
ESTRUCTURA SINTACTICA	Data Flow	ESTRUCTURA SINTACTICA
ESTRUCTURA VALIDA	Data Flow	ESTRUCTURA VALIDA
FONTS O.K.	Data Flow	FONTS O.K.
FORMA	Data Flow	FORMA OBJETO
FORMA O.K.	Data Flow	FORMA O.K.
FORMATO DE PANTALLA	Data Flow	FORMATO DE PANTALLA
IMPRESION O.K.	Data Flow	IMPRESION O.K.
INSERTA FORM FEED	Data Flow	INSERTA FORM FEED
INSTALACION O.K.	Data Flow	INSTALACION O.K.
LINEAS	Data Flow	LINEAS
LINEAS=LINEAS X PAGINA	Data Flow	LINEAS=LINEAS X PAGINA
MENSAJE DE ENVIO	Data Flow	MENSAJE DE ENVIO
MENSAJE DE SALIDA	Data Flow	MENSAJE DE SALIDA
MENSAJE DE TERMINACION	Data Flow	MENSAJE DE TERMINACION
METRICAS	Data Flow	REGISTRO FONTS
NO BORRAR ARCHIVO DE IMPRESION	Data Flow	NO BORRAR ARCHIVO DE IMPRESION
NOMBRE ARCHIVO DATOS VARIABLES	Data Flow	NOMBRE FISICO ARCHIVO_DATOS
NOMBRE ARCHIVO FISICO DE FONT	Data Flow	NOMBRE ARCHIVO FISICO DE FONT
NOMBRE ARCHIVO FISICO DE FORMA	Data Flow	NOMBRE ARCHIVO FISICO DE FORMA
NOMBRE DE FONT	Data Flow	NOMBRE DE FONT

Tabla 2.1 Diccionario de Datos (Cont.)

IDENTIFICADOR DEL OBJETO	TIPO	ETIQUETA DEL OBJETO
NOMBRE DE FORMA	Data Flow	NOMBRE DE FORMA
NOMBRE DE IMPRESORA	Data Flow	NOMBRE DE IMPRESORA
O.K. IMPRESORA EXISTE	Data Flow	O.K. IMPRESORA EXISTE
PRODUCTO DE SOFTWARE	Data Flow	PRODUCTO DE SOFTWARE
REGISTRO ARCHIVO DATOS EOF	Data Flow	REGISTRO ARCHIVO_DATOS EOF
REGISTRO DE FONT	Data Flow	REGISTRO DE FONT
REGISTRO DE FONT PROCESADO	Data Flow	REGISTRO DE FONT PROCESADO
REGISTRO DE FORMA	Data Flow	REGISTRO DE FORMA
REGISTRO DE FORMA EOF	Data Flow	REGISTRO_FORMA EOF
REGISTRO IMPRESORA EOF	Data Flow	REGISTRO_IMPRESORA EOF
REGISTROS IMPRESORA/FONTS/FORMAS	Data Flow	REGISTROS DE LISTADO
RESPUESTA	Data Flow	RESPUESTA
SECUENCIAS DE ESCAPE	Data Flow	SECUENCIAS DE ESCAPE
SEÑAL DE INSTALACION	Data Flow	SEÑAL DE INSTALACION
TIPO DE LIBRERIA	Data Flow	TIPO_LIBRERIA
TIPO_LIBRERIA NOMBRE_IMPRESORA	Data Flow	TIPO_LIBRERIA NOMBRE_IMPRESORA
TOKEN	Data Flow	TOKENS
TRANSACCION	Data Flow	TRANSACCION
TRANSACCION DE EJEMPLO	Data Flow	TRANSACCION DE EJEMPLO
TRANSACCION DE IMPRESION	Data Flow	TRANSACCION DE IMPRESION
TRANSACCION O.K.	Data Flow	TRANSACCION O.K.
ACEPTA CONFIGURACION	Data Process	1.3.3 ACEPTA CONFIGURACION
ACEPTA MODIFICACION	Data Process	1.4.4 ACEPTA MODIFICACION
ACEPTA NOMBRE DE IMPRESORA	Data Process	1.2.1, 1.3.1, 1.4.1, 1.5.1 ACEPTA NOMBRE
ACEPTA TIPO_LIB NOMBRE_IMPRESORA	Data Process	2.3.1 ACEPTA TIPO LIBRERIA Y NOMBRE DE IMPRESORA
ACEPTA TRANSACCION	Data Process	1.1.2, 2.1.2 ACEPTA TRANSACCION
AGREGA FONT EN LIBRERIA DE FONTS	Data Process	2.2.4 AGREGA FONT EN LIBRERIA
AGREGA FONTS	Data Process	2.2 AGREGA FONTS
AGREGAR IMPRESORAS	Data Process	1.3 AGREGAR IMPRESORAS
ANALIZADOR SEMANTICO	Data Process	3.3 ANALIZADOR SEMANTICO
BIENVENIDA Y DERECHOS	Data Process	BIENVENIDA Y DERECHOS
BORRA ARCHIVO DE IMPRESION	Data Process	4.2.6.4 BORRA ARCHIVO DE IMPRESION
BORRAR IMPRESORAS	Data Process	1.2 BORRAR IMPRESORAS
CARGA EL PRODUCTO	Data Process	CARGA EL PRODUCTO
COMPILADOR	Data Process	3 COMPILADOR
CONSULTAR IMPRESORAS	Data Process	1.5 CONSULTAR IMPRESORAS
COPIA FONT DE MEDIO A ARCHIVO	Data Process	2.2.5 COPIA FONT DE MEDIO
DEFINICION DE IMPRESORAS	Data Process	1 DEFINICION DE IMPRESORAS
DESPLIEGA MENSAJES	Data Process	2.4.2 DESPLIEGA MENSAJES
DESCARGA FONTS	Data Process	4.2.4, 4.3.3 DESCARGA FONTS
DESCARGA FORMA	Data Process	4.2.5, 4.3.4 DESCARGA FORMA
DESPLIEGA CONFIGURACION	Data Process	1.4.3, 1.5.3 DESPLIEGA CONFIGURACION
DESPLIEGA INFORMACION	Data Process	1.2.3 DESPLIEGA INFORMACION
DESPLIEGA PANTALLA	Data Process	1.1.1, 2.1.1 DESPLIEGA PANTALLA
EJEMPLO DE FORMA	Data Process	4.3 EJEMPLO DE FORMA
ENVIO A IMPRESION DE LISTADO	Data Process	2.4.3 ENVIO A IMPRESION
ENVIO DE LINEAS	Data Process	4.2.6.1 ENVIO LINEAS
FORMATEA Y RUTEA DEF. IMPRESORAS	Data Process	1.1 FORMATEA Y RUTEA
FORMATEA Y RUTEA UTILERIA	Data Process	2.1 FORMATEA Y RUTEA
GENERA CODIGO (SECS_ESCAPE)	Data Process	3.4 GENERA CODIGO
GENERA SALIDA	Data Process	2.4 GENERA SALIDA

Tabla 2.1 Diccionario de Datos (Cont.)

IDENTIFICADOR DEL OBJETO	TIPO	ETIQUETA DEL OBJETO
IMPRESION DE ARCHIVO	Data Process	4.2 IMPRESION DE ARCHIVO
INCLUSION DE CONTROL	Data Process	4.2.6, 4.2.6.2 INCLUYE CODIGO CONTROL
INTERFASE DE IMPRESION	Data Process	4 INTERFASE DE IMPRESION
LEER MEDIO (DISKETTE)	Data Process	2.2.1 LEER MEDIO
MANTIENE ARCHIVO DE IMPRESION	Data Process	4.2.6.5 MANTIENE ARCHIVO DE IMPRESION
MODIFICAR IMPRESORAS	Data Process	1.4 MODIFICAR IMPRESORAS
MUESTRA DATOS DE FONT	Data Process	2.2.3 MUESTRA DATOS
OBTIENE LISTADO	Data Process	2.3 OBTIENE LISTADO
PARSER	Data Process	3.2 PARSER
PREPARAR SALIDA	Data Process	3.5 PREPARAR SALIDA
PROCESA REGISTRO FONT	Data Process	2.2.2 PROCESA REGISTRO DE FONT
PROCESA LIBRERIA	Data Process	2.3.3 PROCESA LIBRERIA
PRODUCE CONFIRMACION	Data Process	1.6 PRODUCE CONFIRMACION
PRODUCE RESPUESTA	Data Process	1.7 PRODUCE RESPUESTA
RUTEA DEFINICION DE IMPRESORAS	Data Process	1.1.4 RUTEA DEFINICION DE IMPRESORAS
RUTEA SALIDA	Data Process	2.4.1 RUTEA SALIDA
RUTEA TRANSACCION	Data Process	4.1.3 RUTEA TRANSACCION
RUTEA UTILERIA	Data Process	2.1.4 RUTEA UTILERIA
SCANNER	Data Process	3.1 SCANNER
SCANNER DE COMANDO DE IMPRESION	Data Process	4.1.1 SCANNER
SOLICITA DATOS INICIALES	Data Process	SOLICITA DATOS INICIALES
UTILERIA	Data Process	2 UTILERIA
VALIDA EXISTENCIA ARCHIVO_DATOS	Data Process	4.2.3 VALIDA EXISTENCIA DE ARCHIVO DE DATOS
VALIDA EXISTENCIA DE FORMA	Data Process	4.2.1, 4.3.1 VALIDA EXISTENCIA DE FORMA
VALIDA EXISTENCIA DE REGISTRO	Data Process	1.2.2, 1.4.2, 1.5.2 VALIDA EXISTENCIA DE REGISTRO
VALIDA IMPRESORA	Data Process	2.3.2, 4.2.2, 4.3.2 VALIDA IMPRESORA
VALIDA NO EXISTENCIA DE REGISTRO	Data Process	1.3.2 VALIDA NO EXISTENCIA DE REGISTRO
VALIDA NO. DE COPIAS	Data Process	4.2.6.3 VALIDA No. COPIAS
VALIDA TRANSACCION	Data Process	1.1.3, 2.1.3 VALIDA TRANSACCION
VALIDA TRANSACCION Y RUTEA	Data Process	4.1 VALIDA TRANSACCION Y RUTEA
ARCHIVO DE CONTROL	Data Store	ARCHIVO DE CONTROL
ARCHIVO DE DATOS DE FONTS	Data Store	ARCHIVO DE FONTS
ARCHIVO DE DATOS VARIABLES	Data Store	ARCHIVO DE DATOS
ARCHIVO DE IMPRESORAS	Data Store	ARCHIVO DE IMPRESORAS
ARCHIVO DE LISTADO	Data Store	LISTADO
ARCHIVO PROGRAMA FUENTE DE FORMA	Data Store	FORMA FUENTE
ARCHIVO PROGRAMA OBJETO (FORMA)	Data Store	FORMA OBJETO
LIBRERIA DE FONTS	Data Store	LIBRERIA DE FONTS
LIBRERIA DE FORMAS	Data Store	LIBRERIA DE FORMAS
MEDIO FISICO (DATOS DE FONT)	Data Store	MEDIO
MEDIO FISICO FAT (ARCHIVO_DATOS)	Data Store	MEDIO (DISCO) FAT
PRODUCTO DE SOFTWARE	Data Store	PRODUCTO DE SOFTWARE
IMPRESORA	External Entity	IMPRESORA
USUARIO	External Entity	USUARIO

Table 2.1 Diccionario de Datos (Cont.)

Las estructuras de los archivos de datos que se utilizarán en el producto se muestra en la siguiente tabla:

DATOS DEL REGISTRO DE LIBRERIA DE FONTS		
Nombre	**Longitud**	**Tipo**
NOMBRE DE FONT	25	CHARACTER
TAMAÑO DE FONT	4.2	NUMERIC
TIPO DE FONT	4	CHARACTER
ORIENTACION DE FONT	1	CHARACTER
ESPACIAMIENTO DE FONT	1	CHARACTER
ALTO DE CARACTER	4	NUMERIC
ANCHO DE CARACTER	4	NUMERIC
TAMAÑO DE ARCHIVO	6	NUMERIC
NOMBRE ARCHIVO FISICO	20	CHARACTER
FECHA DE INCLUSION	8	CHARACTER
DATOS DEL REGISTRO DE LIBRERIA DE FORMAS		
Nombre	**Longitud**	**Tipo**
NOMBRE DE FORMA	10	CHARACTER
IMPRESORA DE FORMA	4	CHARACTER
ORIENTACION DE FORMA	1	CHARACTER
TAMAÑO DE PAPEL	10	CHARACTER
LINEAS POR PAGINA	2	NUMERIC
FECHA DE CREACION	8	CHARACTER
FONTS USADOS [1..10]	25	CHARACTER
DATOS DEL REGISTRO DE IMPRESORAS		
Nombre	**Longitud**	**Tipo**
NOMBRE DE IMPRESORA	10	CHARACTER
TIPO DE IMPRESORA	10	CHARACTER
EMULACION DE IMPRESORA	4	CHARACTER
TIPO DE CONEXION	1	CHARACTER
PUERTO DE CONEXION	15	CHARACTER
VELOCIDAD_TRANSMISION	6	NUMERIC
CANTIDAD DE MEMORIA	6	NUMERIC
DATOS DEL REGISTRO DE CONTROL		
Nombre	**Longitud**	**Tipo**
FECHA DE INSTALACION	8	CHARACTER
LENGUAJE DE MENSAJES	1	CHARACTER
LICENCIA (VIGENCIA)	8	CHARACTER
NOMBRE DE COMPAÑIA	30	CHARACTER
NUMERO DE SERIE	15	CHARACTER
DATOS DEL REGISTRO DEL ARCHIVO DE FONT		
Nombre	**Longitud**	**Tipo**
DESCRIPCION DE FONT	80	BYTE
ESPECIFICACIONES DE FONT	230	BYTE
DATOS DE FONT	VARIABLE	BYTE

Tabla 2.2 Estructuras de Archivo

2.3 Requisitos funcionales y de operación

La fase de análisis del desarrollo de productos de software implica, entre otras cosas, la definición de los requisitos funcionales y de operación, que el producto deberá cumplir. Estos requisitos especifican las características y capacidades que el producto tendrá, empleando para ello una notación formal.

Las notaciones formales se caracterizan por ser concisas y no ambiguas, esto facilita el razonamiento formal de las especificaciones funcionales y ayudan a verificar los resultados del producto de software a desarrollar. Una de las notaciones formales más comúnmente usadas son las notaciones orientadas a estados, las tablas de transición son un buen ejemplo de éstas.

Las tablas de transición se utilizan para indicar los cambios en el estado de un sistema como respuesta a una entrada específica, además determinan las acciones que se realizarán y los resultados que se producirán al ir de un estado a otro. El estado de un sistema determina la forma en que interactuan todas sus entidades en un momento determinado.

A continuación se muestran las tablas de transición de cada uno de los módulos que componen el producto de software.

Tabla 2.3. Proceso de Instalación del Producto.

Estado Presente	Entrada	Acciones	Salida	Estado Siguiente
S0	Comando	Presenta pantalla		S1
S1	Datos iniciales	Acepta datos	Password generado	S2
S2	Password usuario	Valida password	Archivo Control	S3
			Señal error	SE
S3	Señal de instalación	Instala producto	Instala OK	S0
			Señal error	SE
SE	Señal de error	Despliega error		S0

Tabla 2.4. Interfase de mantenimiento de Impresoras.

Estado Presente	Entrada	Acciones	Salida	Estado Siguiente
S0	Comando	Presenta pantalla		S1
S1	Transacción	Valida transacción		S2 | S3 | S4 | S5 | S0 | SE
S2	Datos	Acepta / valida	Código confirmación	SC1
			Código error	SE1
S3	Dato	Acepta / valida	Código confirmación	SC2
			Código error	SE2
S4	Dato	Acepta / valida	Código despliegue	S6
			Código error	SE2
S5	Dato	Acepta / valida	Código respuesta	S7
			Código error	SE2
S6	Datos	Acepta / despliega / valida		SC3
			Código error	SE1
S7	Datos	Desplegar	Registro de Impresora	S1
SC1	Código "S"	Grabar	Archivo de Impresoras	S1
	Código "N"	Ignorar		S1
SC2	Código "S"	Borrar reg.	Archivo de Impresoras	S1
	Código "N"	Ignorar		S1
SC3	Código "S"	Regraba reg.	Archivo de Impresoras	S1
	Código "N"	Ignorar		S1
SE		Desplegar error		S1
SE1	Código "R"	Reintentar		S2|S6
	Código "C"	Ignorar		S1
SE2		Desplegar mensaje		S3|S4|S5

Tabla 2.5. Interfase de fonts

Estado Presente	Entrada	Acciones	Salida	Estado Siguiente
S0	Comando	Presenta pantalla		S1
S1	Transacción	Valida transacción		S2|S3|S0|SE
S2	Código agrega	Valida medio	Código OK	S4
			Código error	SE
S3	Código listar	Acepta / valida	Tipo listado	S5 | S6
			Código error	SE
S4	Código OK	Leer medio y Desplegar info.	Código confirmación	S7
			Mensaje	S0
S5	Lib. Fonts	Lee archivo y genera reporte	Reporte	S0
S6	Lib. Formas	Lee archivo y genera reporte	Reporte	S0
S7	Código "S"	Actualiza y copia archivo	Lib. Fonts y Archivos Font	S0
	Código "N"	Ignora		S0
SE	Código error	Despliega Mensaje		S0

Tabla 2.6. Compilador de Formas

Estado Presente	Entrada	Acciones	Salida	Estado Siguiente
S0	Comando	Valida		S1
			Código error	SE
S1	Forma fuente	Scanning	Tablas	S2
			Archivo error	S0
S2	Tablas y Lib. Fonts	Parsing	Estructura Semántica	S3
			Archivo error	S0
S3	Estructura Semántica	Análisis Semántico	Estructura válida	S4
S4	Estructura válida y tablas	Genera código y Actualiza Lib.	Forma objeto y Lib. Formas	S0
			Archivo error	S0
SE	Código error	Despliega mensaje		S0

Tabla 2.7. Interfase de impresión.

Estado Presente	Entrada	Acciones	Salida	Estado Siguiente
S0	Comando	Scanning	Tablas	S1
			Código error	SE
S1	Tablas	Parsing	Comando válido	S2 | S3
			Código Error	SE
S2	Comando válido Archivo y/o Lib. Formas e Impresoras	Valida recursos		S3
			Código Error	SE
S3	Archivo Fonts Lib. Fonts	Descarga fonts	Impresora	S4
S4	Forma Objeto	Descarga forma	Impresora	S5 | S6
S5		Liberar forma de impresión	Mensaje	S0
S6	Archivo datos	Envío e inclusión de control	Impresora	S0
SE	Código error	Despliega mensaje		S0

2.4 Prioridades de instrumentación.

La propuesta va enfocada a las facilidades que se le pueda dar al personal encargado de la elaboración de formas electrónicas de impresión en ambiente Unix.

Primeramente, se procederá a elaborar un Lenguaje de Diseño de Formas que debe ser entendible para cualquier persona que quiera definir alguna forma. Este lenguaje deberá tener la capacidad de poder traducir términos coloquiales como caja, línea, centra, etc., a secuencias de escape entendibles para la impresora. Esta es una tarea que requiere del empleo de un pseudo compilador que realice las funciones de scanning y parsing, con la variante de que en vez de generar código objeto para computadoras, se generará un archivo con secuencias de escape que se puede llamar código objeto para impresoras.

En base a lo anterior, es de suma importancia determinar cuales serán las palabras reservadas que formarán parte de dicho lenguaje, además de definir si este lenguaje contará con una estructura rígida (por ejemplo: el nombre de la forma, la definición de los fonts a usar, etc.) o si se pretende que nuestro compilador maneje una estructura dinámica.

Adicionalmente, se debe definir una librería para el manejo de los fonts. Esta librería deberá ser capaz de almacenar registros que le indiquen las especificaciones de los fonts (nombre, estilo, peso, tamaño), así como la ubicación de los archivos que contendrán todos los caracteres correspondientes a cada font. Esta librería permitirá obtener listados de todas los fonts con que cuente el sistema y tendrá una relación directa con el lenguaje de definición de formas.

Por otra parte, se debe crear una librería de formas semejante a la librería de fonts, que podrá ser actualizada conforme se requiera de nuevas formas.

Se pretende con lo anterior, garantizar en la medida de lo posible, la existencia de una relación directa entre los archivos de definición de formas y los archivos de fonts con que cuenta el sistema, ya sea en la computadora o los fonts residentes en la impresora.

Una vez logrado lo anterior, es de vital importancia realizar una interfase de usuario, en la que sea posible determinar qué impresora (si es que en el sistema existe más de una) es el destino final de nuestra impresión. Esto se logra a través del manejo del administrador de impresión propio de los sistemas Unix, pero pretendemos crear una caja de diálogo con el usuario de manera tal, que no se le obligue a recordar todos los comandos necesarios para la ejecución de la tarea expuesta anteriormente.

Estrechamente ligada a esta tarea, se encuentra la interfase de definición de impresoras, donde se pretende que el manejo sea fácil, para que desde un usuario experto hasta un principiante pueda realizarla sin ningún problema.

El producto final es una interfase que permita al usuario final, crear sus formas en un lenguaje coloquial, manejar fonts y agregar datos variables en la forma electrónica a imprimir.

2.5 Modificaciones y mejoras previstas.

Se habló, en el capítulo anterior, de la necesidad de crear un producto de software para impresoras de bajo volumen, que permita generar descripción de formas y de impresión, de manera transparente. Sin duda, el resultado final de todo este proceso deberá ser un producto confiable que cumpla con sus objetivos.

Sin embargo, una mirada retrospectiva muestra que el producto puede aún ser mejorado, pero el hacerlo requiere una inversión superior a la establecida, en cuanto a costo del producto y tiempo de desarrollo. Es por tal motivo, que se propone a continuación algunos de los ajustes que el proceso puede sufrir y que quizá sean de interés para un segundo estudio.

La descripción de la forma podrá ser mejorada, agregando en el lenguaje la generación de líneas dobles, logotipos, firmas y sombreados.

Como el uso de fonts y de los dispositivos de impresión no está limitado más que por costo y tiempo, se propone implementar mayor número de fonts a los dispositivos predefinidos, o bien, agregar otros dispositivos de impresión (IBM, EPSON, etc.) con sus respectivos fonts, mejorando así este proceso.

Por otra parte, no debe existir desbordamiento de información en los sistemas de impresión, por lo que, una posible mejora radica en particionar la información en bloques, de tal manera, que la información enviada al dispositivo de impresión no exceda la capacidad de memoria disponible.

Finalmente se propone mejorar la salida de impresión, agregando en el módulo la capacidad de impresión duplex (ambas caras de la hoja) y TWO-UP (dos páginas lógicas en una página física).

2.6 Criterios de Aceptación.

Uno de los aspectos importantes dentro de la fase de análisis es sin duda el establecimiento de normas o estándares, cuya finalidad es la de determinar que se cumplan los requisitos funcionales y de operación, establecidos en la especificación de requerimientos.

En esta fase del ciclo de vida del producto de software se debe determinar una versión preliminar de los criterios de aceptación; tales criterios deben especificar las pruebas funcionales y de rendimiento que se aplicarán al producto de software, así como los estándares que serán aplicados al código fuente. Pero será hasta la fase de diseño en la que se iniciará un plan de aceptación y verificación el cual se concluirá hasta la fase de instrumentación y pruebas.

Lo primero a establecer serán los factores de calidad del software de los cuales se derivan los atributos o métricas, las que a su vez son mesuradas mediante las pruebas de aceptación del sistema. Esos factores se basan en tres aspectos del producto de software: sus características operacionales, su capacidad de soportar los cambios y su adaptabilidad a nuevos entornos.

Las siguientes métricas que deberá poseer el producto serán las siguientes:

- Correctibilidad.

  El producto final deberá proporcionar una solución efectiva al problema propuesto, por tanto el objetivo del producto final será en este caso un sistema que contemple un lenguaje de diseño de formas, un compilador para el mismo, así como interfases amigables tanto para impresión como para el manejo de la utilería.

- Confiabilidad

  Cada uno de los componentes del producto deberá de realizar sus funciones siempre con la precisión requerida y con resultados válidos, durante todo el tiempo.

  El usuario del producto siempre deberá obtener la forma impresa que describió, haciendo uso del Lenguaje de Descripción de Formas. El compilador de este lenguaje, será capaz de detectar y reportar todos los errores que el usuario pueda cometer al escribir sus programas, estos errores pueden ser sintácticos, semánticos o por querer usar fonts que no estén registrados en la librería respectiva. Si el programa no tiene errores, el compilador deberá generar las secuencias de escape necesarias para obtener la forma impresa descrita en el programa y actualizará la librería de formas.

  El módulo de definición de impresoras proporcionará las funciones de Altas, Bajas, Cambios, Consultas y Listados sobre la librería de impresoras definidas en el sistema, validando los datos que proporcione el usuario.

  El módulo de adquisición de fonts copiará al sistema de cómputo, los archivos de fonts a utilizar durante la impresión de un documento y actualizará la librería fonts, otra función de la utilería será obtener listados de fonts y formas de las librerías respectivas.

  La Interfase de Impresión enviará a la impresora el archivo de datos, el archivo de forma (forma objeto), los archivos de fonts y los adecuados comandos de control, para la impresión del documento que el usuario indique por medio de un comando. La interfase revisará que el comando esté bien escrito y checará que la forma que se pretende usar exista en la librería de formas, emitiendo el mensaje adecuado si se detecta algún error.

- Eficiencia

  Todos los componentes del producto efectuarán sus funciones usando un mínimo de recursos del sistema de cómputo y de tiempo.

  El Lenguaje de Descripción de Formas se diseñará de tal forma, que el usuario pueda crear sus formas utilizando un mínimo de instrucciones, con el fin de ahorrar espacio en disco y tiempo de compilación.

  Para poder instalar el producto se requerirán de 400 a 500 KB de espacio en disco, esto representa una cantidad reducida si se toman en cuenta los beneficios que se pueden obtener con el uso del producto

- Solidez

  A pesar de la introducción de datos inválidos por parte del usuario, los componentes funcionales del producto, serán capaces de continuar operando correctamente y de generar los mensajes que se requieran para dar un informe de los errores detectados. Por ejemplo, el compilador terminará normalmente y emitirá el o los mensajes correspondientes, cuando detecte errores en la forma fuente que se este compilando, o cuando se intente compilar una forma que no exista.

  Como parte de la documentación, el producto contará con un Manual de Mensajes de Error, que contenga una explicación más amplia sobre cada error o problema que se pueda presentar al usar el producto. El objetivo de este manual será ayudar al usuario a tomar las acciones adecuadas para corregir sus errores o solucionar problemas. Para mayor referencia consultar Manual de Usuario.

- Facilidad de uso o usabilidad

  El producto de software que se desarrollará será fácil de usar, el usuario final sólo requerirá del Manual de Usuario y del Manual de Mensajes de Error para poder usar eficientemente todos sus componentes.

  El módulo de definición de impresoras y el módulo de adquisición de fonts, contarán con listas de opciones y pantallas que guiarán al usuario en el uso de estos componentes, haciéndolos de esta forma amigables y fáciles de usar.

  El Compilador y la Interfase de Impresión serán invocados con el uso de sencillos comandos, estos comandos requerirán de unos cuantos parámetros posicionales, que podrán ser consultados en línea de acuerdo a los estandares usados en UNIX.

  ej:

```
sixcdf [cr]
uso: sixcdf <forma fuente> [opciones]
opciones:
-l = genera listado de forma
-s = checar sintaxis de forma sin generar código
```

  El Lenguaje de Descripción de Formas será un lenguaje de alto nivel fácil de leer y escribir, semejante al lenguaje humano, y que se podrá usar en cualquiera de sus dos versiones, español o inglés. La estructura del lenguaje tendrá un diseño modular que permitirá que los programas sean fáciles de entender y modificar, además se contará con la facilidad de incluir comentarios para poder documentarlos.

- Integridad

  Este rubro contempla el grado en que pueda controlarse el acceso al software o a los datos por personal no autorizado, es decir, la seguridad. Debido a las características del uso del sistema no se pondrá restricción al acceso del mismo, excepto en la etapa de instalación del producto; ya que se solicitará un password para poder realizar tal acción.

  Sin embargo es importante mencionar que los archivos de fonts y formas no podran ser eliminados del sistema, asi como el registro de éstos en la base de datos del sistema.

- Facilidad de mantenimiento

  Dado que el producto de software será diseñado de forma modular, el esfuerzo requerido para localizar y arreglar algún error en el programa deberá ser mínimo; con esto se espera que el mantenimiento sea una tarea fácil de realizar.

- Portabilidad

  Con el fin de que su uso no se vea limitado a un solo sistema de cómputo o versión del sistema operativo, el producto podrá ser transferido de un sistema de cómputo a otro, bajo el sistema operativo Unix.

  Esta capacidad que tendrá el producto, se debe al elevado grado de portabilidad que ofrece el lenguaje de programación 'C'. Este lenguaje, permite que aplicaciones desarrolladas en un equipo determinado puedan funcionar sobre equipos de otros fabricantes, con la realización de pocas o incluso sin modificaciones.

- Flexibilidad

  Un aspecto importante que el producto de software deberá tener presente, será que el esfuerzo requerido para modificar el programa o una parte del programa deberá ser mínimo; debido a esto, el trabajo para modificar o en su caso aplicar mejoras al sistema tendrá que ser contemplado en un futuro; esta tarea deberá ser de fácil realización, ya que el diseño del sistema en general se hará de forma estructurada y modular.

- Facilidad de prueba

  Al diseñar el producto de software de forma modular se dará la pauta para que al aplicar las pruebas a cada módulo del sistema, se asegure que cada parte del programa cumple con su función requerida.

Otro aspecto a considerar dentro de la versión preliminar de los criterios de aceptación, es el establecer un formato del plan de prueba de aceptación, éste deberá considerar los siguientes puntos:

- Establecer que requisitos se verificarán.
- Establecer casos de prueba para cada requisito.
- Inferir resultados esperados para cada caso de prueba.
- Determinar las capacidades demostradas por cada prueba.

Para garantizar que se satisfagan los requerimientos funcionales y de operación, es importante definir qué tipo de pruebas se deberán aplicar al código fuente; éstas son las siguientes:

Pruebas de tipo funcional

Este tipo de pruebas es para verificar condiciones operativas, es decir, se definen valores de entrada comunes y resultados esperados a tales valores.

Pruebas de desempeño.

El objetivo de estas pruebas es para verificar tiempos de respuesta, porcentajes de tiempo de ejecución en módulos del programa, el rendimiento, utilización de memoria primaria y secundaria, tasas de tráfico de datos y los enlaces de comunicación.

Pruebas de tensión.

El objetivo de estas pruebas es sobrecargar el sistema y ver el comportamiento del mismo.

Pruebas de estructura.

Se enfocan a la lógica interna de procesamiento del sistema de software, es decir dirigidas a las rutinas de los programas en donde se toman decisiones; el objetivo es probar todos los caminos lógicos posibles.

Algunos ejemplos para la fase de pruebas de aceptación son los siguientes:

Las pruebas de tipo funcional aplicadas al compilador consistirán en proporcionar una serie de instrucciones con errores como estímulo para probar el analizador sintáctico, el requisito a probar será la fiabilidad del compilador, es decir tendrá que entrar en acción el módulo de generación de errores sintácticos. La suposición para esta prueba será que a entradas incorrectas se esperan mensajes de error durante la compilación y por tanto el proceso será abortado mostrando el posible tipo de error.

Una prueba de desempeño para la interfase de impresión podrá tener como estímulo un archivo con las secuencias de escape conteniendo el número máximo de fonts existentes, el requisito a probar puede ser la eficiencia ya que la interfase de impresión deberá hacer muchos accesos a los archivos donde se encuentran las características del los fonts para descargarlos en la impresora; el objetivo es ver como se comporta el sistema ante los accesos a memoria principal y secundaria, así como los enlaces de comunicación entre el módulos de interfase de impresión y los archivos de fonts.

Una prueba de tensión para el compilador podrá ser el diseño de una forma muy elaborada, es decir con un gran número de líneas, cajas y textos, así como del número máximo de fonts permitidos, los requisitos a probar serán la fiabilidad y eficiencia ya que el archivo generado contendrá un número grande de secuencias de escape, el objetivo será ver el desempeño del compilador ante tal situación.

Una posible prueba de estructura para el compilador podrá ser que al diseñar una forma de impresión como entrada se especifique una forma inexistente dentro de la librería de formas, se espera que el módulo de semántica detecte esos posibles errores y envíe mensajes apropiados al usuario para corregir ese error. El objetivo será probar la confiabilidad del analizador semántico.

2.7 Guías y sugerencias de diseño

Para el diseño del lenguaje de descripción de formas se sugiere sea realizado un análisis y estudio de otros lenguajes existentes para otras plataformas de cómputo; por ejemplo el Form Description Language de los sistemas de impresión electrónica Xerox de alto volumen, el cual es parte integral del mismo; otro lenguaje que puede ser analizado es el Form Design Language del producto Spool/LPM, propiedad de Information Systems Corporation, el cual es usado en los sistemas Serie A de Unisys. De esta manera es posible tener una visión más amplia de lo que el lenguaje propuesto deberá cubrir.

Asimismo, se sugiere que el diseño del compilador indique sólo la validación de la existencia de fonts en la librería respectiva, dejando al módulo de envío a impresión la descarga de los mismos. Aunque los fonts pueden ser incluidos en el objeto de la forma, realizarlo de esta manera hará que los archivos objeto crecieran volviendo al sistema ineficiente por el uso de espacio en disco requerido para almacenar las formas.

Es de suma importancia tener en cuenta la estructura de archivos que se maneja en el sistema operativo Unix, de tal suerte que la implantación del producto en el ambiente de operación sea realizada sin efectuar cambios mayores o sustanciales del producto obtenido en la fase de desarrollo. Por tal motivo es importante mencionar que el diseño del producto no debe tomar en cuenta el llamado a rutinas o interrupciones de un determinado procesador, sino que debe ser diseñado sólo con elementos estándar.

Es importante indicar que existen productos que pueden agilizar el diseño y desarrollo del compilador, entre los cuales se encuentran ScanGen que acepta las descripciones de tokens escritos como expresiones regulares y produce tablas que pueden ser usadas por un analizador sintáctico. Así mismo existe el LLGen que acepta especificaciones gramaticales de contexto libre y produce tablas para analizar gramáticas del lenguaje especificado. De esta manera el poder conseguirse estos productos ahorraría tiempo de desarrollo.

En lo referente a los demás módulos del producto, se sugiere se realice código reusable de tal manera que el diseño del producto pueda optimizarse en tiempo, siempre y cuando el mismo código pueda ser usado por los procesos generales. Así el código para la validación de la impresora, la existencia de forma y fonts y la validación de la vigencia de la licencia pueda ser compartido o reusado por varios procesos.

Ya que las definiciones y criterios que se han mencionado durante las secciones Requisitos funcionales y de operación y Criterios de aceptación, son muy similares y de gran importancia en cualquier desarrollo de sistemas, a continuación se indican brevemente algunos conceptos para la garantía de la calidad del software.

2.7.1 Garantía de calidad del software

La garantía de calidad del software (GCS) es uno de los aspectos más importantes dentro de la ingeniería de software, no solo es la clave para lograr que el cliente quede satisfecho sino que tiene un impacto sobre el costo del proyecto, así como en el calendario de actividades del proyecto.

De acuerdo al estándar P730 de la IEEE (The Institute of Electrical and Electronics Engineers) la garantía de calidad del software está definida como:

Un patrón planeado y sistemático de todas las acciones necesarias, para proveer una confianza adecuada en la conformación del software en base a los requerimientos técnicos establecidos.

Tenemos tres puntos que nos pueden dar una idea más clara de que es lo que se busca con la calidad del software.

Primero. Se tiene una visión comprensiva en lugar de una restrictiva, es decir, la GCS no esta restringida a la función de un grupo de calidad de software, mejor aún incluye todas las actividades necesarias que puedan contribuir a la calidad del software durante todo el ciclo de vida del producto.

Segundo. El énfasis está en un plan y su implementación sistemática para lograr los objetivos de la calidad del software. La calidad de un producto de software no se deja al esfuerzo aleatorio de programadores individuales.

Tercero. La noción de calidad es relativa a algunos requerimientos preespecificados en lugar de algún sentido absoluto de calidad; además el propósito es no garantizar al 100% la confiabilidad o cero defectos, es mejor incrementar la confianza para que cada paso razonable que sea tomado durante la fase de desarrollo, asegure la calidad del producto final.

Definiciones básicas y métricas

Una definición de calidad del software es la ausencia de errores o defectos. Un defecto o error es cualquier desviación del comportamiento esperado.

Basados en esta definición la calidad del software es medida o mesurada en término de defectos por miles de líneas fuente no documentadas (KNCSL Thousands of noncommentary source lines).

La ausencia de defectos es condición necesaria pero no suficiente para obtener calidad; la definición implica que el comportamiento esperado este bien y claramente especificado, así que cualquier defecto puede ser fácilmente detectado.

Los errores o defectos en un producto de software son descubiertos durante la etapa de desarrollo y después de liberar el producto al mercado. Pero los errores que realmente cuentan son aquellos que permanecen aún después de liberar el producto para su uso público. Una forma de medir estos errores es en término de decenas de defectos/KNCSL.

La calidad del software puede predecirse utilizando métricas de calidad o métricas de software. En la primera, la predicción de la densidad de errores no detectados está basada en la densidad de errores observada del producto durante su desarrollo. En la segunda, la predicción de la densidad de errores está basada en la observación de las propiedades estructurales del producto.

La predicción de la calidad del software en base a las métricas de calidad consiste básicamente de tres pasos (Según Jones):

1. Mantener un registro histórico de los datos en la eficiencia de eliminación de defectos acumulados (CDRE:Cumulative defect removal efficiency) donde CDRE está definido como el número de defectos detectados antes de liberar el producto/el número de defectos encontrados antes y después de su liberación.

2. Medición de los defectos encontrados antes de la liberación del producto.

3. Estimar la calidad del producto como (1/CDRE-1) X (el número de defectos encontrados antes de la liberación del producto).

Esto es aplicable siempre y cuando las técnicas de eliminación de defectos usadas se mantengan constantes a lo largo del proyecto y si la historia de datos es recolectada de otros proyectos similares para que el CDRE pueda ser inferido de esos datos.

La desventaja de este método es que no se puede mejorar la técnica de efectividad de eliminación de defectos puesto que se asume que el CDRE se mantiene constante. La ventaja es que proporciona un método simple para el cálculo de la calidad del software y es por eso que es usado ampliamente en la industria del software.

Las técnicas basadas en base a las propiedades estructurales de los programas o métricas de software son: las métricas de McCabe o complejidad ciclomática, las métricas de Halstead y las de tamaño del programa.

La complejidad ciclomática se define como una gráfica (G) del programa como:

$V(G)=e-n+2p$ donde:

e=número de aristas o lazos de la gráfica

n=número de nodos

p=número de componentes conectados

La métrica de Halstead se aplica en base a la longitud del programa y esta definida como:

$N1+N2$ donde:

$N1=n1 \log n1$

=uso total de todos los operadores en un programa.

$N2=n2 \log n2$

=uso total de todos los operandos en un programa.

n1 =total de operadores distintos en un programa.

n2 =total de operandos distintos en un programa.

La métrica de tamaño del programa es en términos de las líneas de código o proposiciones. Siendo esta la más fácil de entender y calcular.

Definiciones de calidad del software.

La calidad del software se basa en las 3 dimensiones de un producto (Según McCall).

1. Las operaciones del producto.
2. Las transiciones del producto.
3. La revisión del producto.

La primera trata de correctividad, confiabilidad, eficiencia, integridad y usabilidad.

La segunda trata de portabilidad, reusabilidad e interoperabilidad.

La tercera trata de mantenibilidad, flexibilidad y facilidad de prueba.

Todos estos factores de calidad son completamente subjetivos y difíciles de cuantificar.

Definición de factores de calidad de McCall:

- Correctibilidad.

  Grado en el que el programa satisface las especificaciones y cumple su objetivo principal. (Hace lo que yo quiero?)

- Confiabilidad.

  Grado con el que se espera que el programa desempeñe sus funciones con la precisión requerida. (Lo hace con exactitud todo el tiempo?)

- Eficiencia.

  La cantidad de recursos de cómputo y de código requeridos para que el programa desempeñe su función. (Se ejecutara en mi hardware?)

- Integridad.

  Grado en el cual el acceso al software o a los datos por personal no autorizado pueda ser controlado. (Es seguro?)

- Usabilidad.

  Esfuerzo requerido para aprender, operar, preparar la entrada e interpretar la salida de un programa. (Puedo ejecutarlo?)

- Mantenibilidad.

  Esfuerzo requerido para localizar y corregir un error en el programa. (Puedo arreglarlo?)

- Facilidad de prueba.

  Esfuerzo requerido para probar un programa para asegurar que desempeñe sus funciones. (Puedo probarlo)

- Flexibilidad.

  Esfuerzo requerido para modificar un programa operacional. (Puedo hacerle cambios?)

- Portabilidad.

  Esfuerzo requerido para transferir un programa desde una configuración de hardware y/o sistema de software a otro ambiente. (¿Seré capaz de usarlo en otra máquina?)

- Reusabilidad.

  Grado con el cual el programa pueda ser usado en otras aplicaciones. (¿Seré capaz de re-usar parte del software?)

- Interoperabilidad.

  Esfuerzo requerido para acoplar un sistema con otro. (¿Seré capaz de comunicarlo con otro sistema?)

Aunado a lo anterior es muy importante establecer estándares de programación y desarrollo para el producto, dichos estándares deben ser establecidos por los integrantes del proyecto durante la etapa de diseño, los cuales se indicarán en el Capítulo 4 "Instrumentación".

Página dejada en blanco intencionalmente

DISEÑO DEL PRODUCTO

La primera parte del presente capítulo, se orienta al diseño de un lenguaje de descripción de formas de manera entendible para cualquier persona que desee definir formas electrónicas en su computadora. Posteriormente se hará referencia al diseño de los diferentes módulos que conforman el producto.

Es importante señalar que para la representación de las estructuras del lenguaje de descripción de formas, se seguirá una notación particular que consiste en una combinación de los diagramas de Conway, para especificar por medio de diagramas las combinaciones. Cabe mencionar que dichos diagramas se han utilizado en el ambiente de Unisys/Burroughs como diagramas de ferrocarril (railroad diagram) y la notación de Backus.

Durante este capítulo también se utilizará para la representación del diseño arquitectónico, las cartas de estructura, ya que permiten documentar efectivamente la jerarquía, los parámetros y las interconexiones dentro del producto de software. La principal razón de utilizar estos diagramas, se debe a que no es necesario indicar diagramas de decisión ni secuencia de tareas entre los diferentes módulos del producto, que es lo que se busca en este tipo de diseño.

Adicionalmente, para efectos de diseño detallado, se incluirá inmediatamente después de las cartas de estructura, el pseudocódigo, el cual permitirá indicar paso a paso los algoritmos en esta etapa de diseño; para finalmente obtener el diccionario de datos completo del producto de software.

Cada uno de los módulos que componen el producto de software será diseñado por separado, y al final, tendremos un resumen de todo el producto. Sin embargo, se incluirá una sección dedicada a las rutinas comunes, las cuales formarán una biblioteca de rutinas o funciones, de tal manera que se cumpla con el objetivo de crear código común reutilizable, a fin de simplificar el diseño del producto.

3.1 El lenguaje de diseño de formas

Se busca que el lenguaje de diseño de formas propuesto pueda ser de fácil entendimiento para cualquier persona, por ello se ha puesto especial atención a la estructura y al lenguaje humano, de esta manera el lenguaje de diseño de formas puede ser escrito en español o inglés, facilitando con esto que las personas que están acostumbradas a escribir sus programas en inglés lo puedan seguir realizando. Es así que la estructura del lenguaje es muy parecida a los lenguajes estructurados; ya que escribir un programa que defina una forma se hará en una manera secuencial y en cinco secciones. La definición de una forma deberá ser escrita con las secciones y en la secuencia propuesta que se establece con la siguiente estructura:

<forma>:=
<nombre forma>
<sección impresoras>
<sección atributos>
<sección fuentes>
<sección descripción>

La secuencia de instrucciones deberá darse de la misma manera que se especifica, con motivo de que el compilador pueda resolver los argumentos o parámetros presentados en el cuerpo de la forma. Así la sentencia <nombre de forma> establecerá de una manera sencilla el nombre de la forma, que será usada por el usuario para el envío con la mezcla del archivo de impresión; la <sección impresoras> establecerá la(s) impresora(s) que utilizará(n) la forma, para la generación del código apropiado en cada una de las mismas; la <sección atributos> establecerá los atributos que tendrá la forma, esto es, márgenes, tamaño de papel a ser usado, etc.; la <sección fuentes> permitirá establecer la definición de los fonts que serán utilizados en la forma, los cuales serán descargados a la impresora cuando la forma sea requerida; y la <sección descripción> establecerá la descripción de la forma en sí.

Todas las secciones y sentencias deberán contener al final un caracter de terminación, en este caso, se propone que sea punto y coma ";". La ausencia de este caracter generará un error al momento de compilación.

3.1.1 Comentarios

Se considera la posibilidad de que el programa de la descripción de la forma, pueda ser documentado para facilidad de futuras revisiones o modificaciones que sean requeridas, como se mencionó en el primer capítulo; por lo que se propone la inclusión de comentarios, los cuales deberán ser escritos iniciando con los caracteres /* y finalizando con la secuencia de caracteres */, así el comentario se estructura de la siguiente manera:

<comentario>:=
--- /* --- <cadena> --- */ ---#

Ejemplos:

```
/*  Determinación de los atributos generales de la forma */
/*************************************************************
*                                                            *
* Determinación de los fonts a ser usados en la forma        *
*                                                            *
*************************************************************/
/*  La forma se realiza en orientación Landscape */
```

Otro aspecto a considerar es la posibilidad de realizar comentarios a continuación de las instrucciones o sentencias, para lo cual se propone el uso del caracter "%" despnés del caracter de terminación de sentencia, o bien en la misma línea.

Ejemplo:

```
FORMA  F1031;        % Forma de Estado de cuenta
FORM   BSCM001;      % Forma de Cuenta Maestra 01
```

3.1.2 Sección FORMA

El nombre de una forma deberá empezar siempre con una letra y su longitud máxima permitida es de 10 caracteres con el uso de la siguiente sentencia

<nombre forma>:=

```
----- FORMA --- <cadforma> --- ; ---#
   |-- FORM --|
```

<cadforma>:=

```
                  |------------ <  /9 ------|
---- <alfabeto> ----------- <alfabeto> -----------#
                       |--- <dígito> ---|
```

La sentencia de FORMA/FORM permite establecer el nombre de la forma, solo para efectos de administración; ya que no es necesario establecer el nombre para que la impresora la reconozca; sin embargo, debe ser especificada para que el usuario pueda identificar de una manera eficaz la forma deseada al momento del envío del archivo de impresión. El nombre de la forma siempre deberá iniciar con una letra y su máxima longitud es de 10 caracteres, esto permite ofrecer flexibilidad al usuario para nombrar sus formas. Se recomienda el uso de identificadores de forma no mayores de 8 caracteres, porque es el estandar de PC, y el desarrollo inicial será en este ambiente.

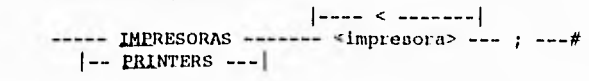
Ejemplos:

```
FORMA FACTURAS;
FORM INVOICES;
FORM PAUTADO;
```

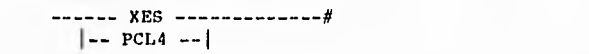
3.1.3 Sección IMPRESORAS

Como el diseño de formas se realizará para dos estilos de impresora diferentes (Xerox y Hewlett Packard), se deberá especificar al compilador la secuencia de escape que deberá generar, de esta manera se incluye en el lenguaje la sentencia que permita definir la(s) secuencia(s) de escape necesarias, de la siguiente manera:

<sección impresoras>:=



<impresora>:=



El compilador al encontrar la sentencia IMPRESORAS/PRINTERS y a continuación el estilo o estilos de impresora generará la secuencia de escape correspondiente, asimismo este dato permitirá a la interfase de impresión saber si la forma generada es para ese estilo o no. Lo interesante es que la forma pueda ser generada para ambos estilos de impresora.

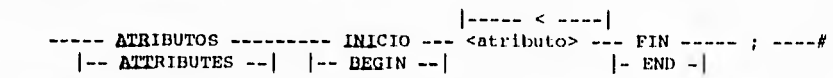
Ejemplos:

```
PRINTERS      XES  PCL4;
IMP    XES;
PRI    PCL4;
```

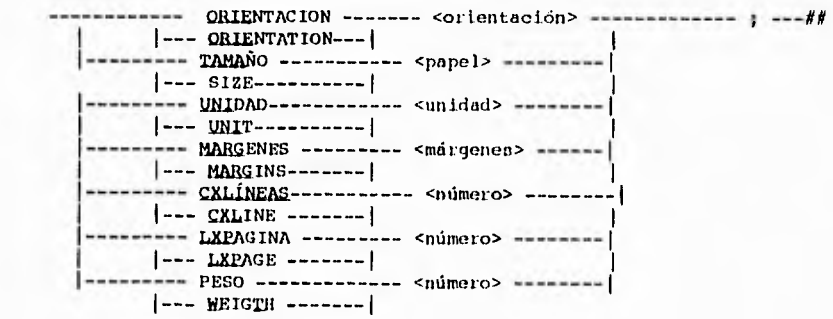
3.1.4 Sección ATRIBUTOS

En toda forma existen ciertos atributos que deben ser especificados, estos indican las características que tiene una forma, lo cual se propone con la sección que se muestra a continuación

<sección atributos>:=

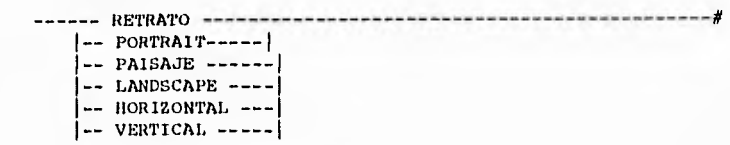


<atributo>:=



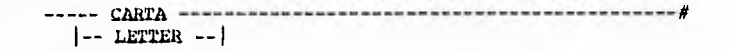
Como se puede observar, los atributos son especificaciones físicas de la forma, ya que incluye la orientación de la página, el tamaño de papel que se utilizará, los márgenes que deberá tener la forma, los caracteres por línea y las líneas por página que se deberán incluir en la página, así como la unidad que será utilizada por defecto al dibujar líneas y el peso o grueso en puntos que éstas últimas tendrán.

<orientación>:=



La orientación de la página se especificará de acuerdo a si se desea la impresión en forma HORIZONTAL/PAISAJE/LANDSCAPE o bien en forma VERTICAL/RETRATO/PORTRAIT, para ello solo deberá aceptarse una definición de orientación en el programa.

<papel>:=

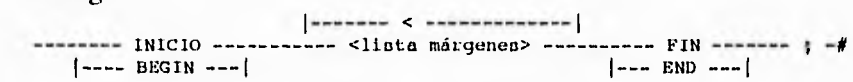


Aunque las impresoras pueden aceptar papel de diferente tamaño, el compilador sólo reconocerá la definición de papel tamaño carta; dejando para futuras mejoras la definición de otros tamaños, como puede ser A4 o LEGAL.

<unidad>:=

La unidad que se especifica en esta parte, tendrá efecto inmediato para la definición de los márgenes y de la longitud de líneas o cajas que puedan ser declaradas más adelante, de esta manera las unidades permitidas en la definición son CM o CENTIMETROS/CENTIMETERS, PULGADAS/INCHES, PUNTOS/DOTS.

<márgenes>:=



En cuanto a márgenes se refiere, pueden ser especificados todos o sólo algunos, de esta manera los márgenes a ser declarados son los siguientes:

<lista márgenes>:=

```
-------- SUPERIOR --------------- <número> ------------------- ; ---#
    |---- INFERIOR ------|
    |---- DERECHO -------|
    |---- IZQUIERDO -----|
    |---- TOP -----------|
    |---- BOTTOM --------|
    |---- LEFT ----------|
    |---- RIGHT ---------|
```

Así los márgenes se pueden especificar como SUPERIOR/TOP, INFERIOR/BOTTOM, IZQUIERDO/LEFT, DERECHO/RIGHT.

<peso>:=

```
---------PESO--------------- <número> ------#
    |-- WEIGHT-|
```

Finalmente la sentencia PESO/WEIGHT indicará el grueso de las líneas, en puntos, que serán dibujadas por defecto en la forma.

Ejemplos:

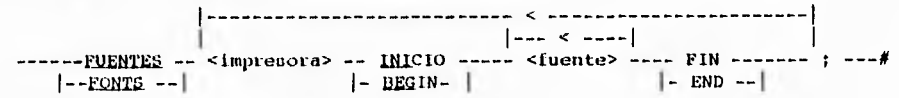
```
ATRIBUTOS
        INICIO
                ORIENTACION     HORIZONTAL;
                TAMAÑO          CARTA;
                UNIDAD          CM;
                MARGENES
                        INICIO
                                SUPERIOR        0.5;
                                IZQUIERDO       0.5;
                                DERECHO         0.5;
                                INFERIOR        0.5;
                        FIN;
                PESO                    2;
        FIN;
ATT
        BEGIN
                ORIE            LANDSCAPE;
                SIZE            LETTER;
                UNIT            INCH;
                MARG
                        BEGIN
                                TOP     0.25;
                                LEFT    0.25;
                        END;
                WT              2
        END;
```

3.1.5 Sección FUENTES

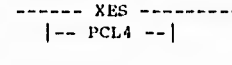
Una de las principales características que tienen las impresoras láser, es el uso de diferente tipografía o tipo de letra y tamaño, por lo que en algunas impresoras se cuenta con una gran diversidad de fonts ya incluidos en los ROM's y que pueden ser referenciados desde la forma; sin embargo, es posible obtener más fonts que son descargados a la impresoras desde el computador cuando se desean usar por esta razón la siguiente sección

servirá para especificar qué fonts se utilizarán en la forma, cuales de ellos ya son residentes y cuales deberán ser descargados en la impresora.

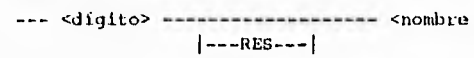
<sección fuentes>:=



<impresora>:=



<fuente>:=



La sentencia FUENTES/FONTS indicará que fonts se utilizarán en la forma y para que impresora se ligarán; ya que el compilador debe manejar los fonts que sean utilizados para el estilo de impresora que se defina, de esta manera si en la sección de definición de impresoras se incluyó sólo a una Xerox y en esta sección se indican fonts para Hewlett Packard deberá indicar el error, además de verificar la existencia de los fonts en la librería correspondiente, si se indica la palabra RES querrá decir que los fonts son residentes en la impresora; esto es, que ya están cargados en el ROM de la misma.

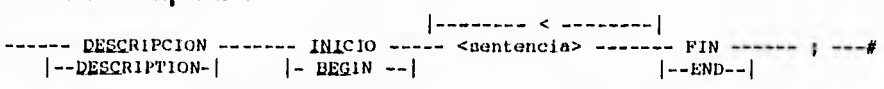
Ejemplo:

```
FONTS
        XES
        BEGIN
                0 RES  Titan10iso-L;
                1      Helve12BI-L;
        END;
```

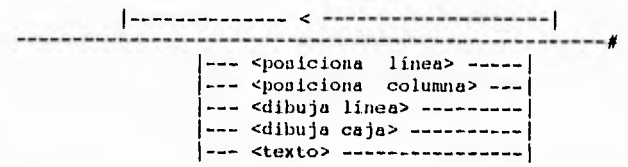
3.1.6 Sección DESCRIPCION

Una vez que se han definido las secciones anteriores, se inicia la descripción de la forma que se va a definir, lo cual es realizado por la última sección. Es aquí donde se le da cuerpo a la forma, lo cual se consigue como se indica a continuación:

<sección descripción>:=



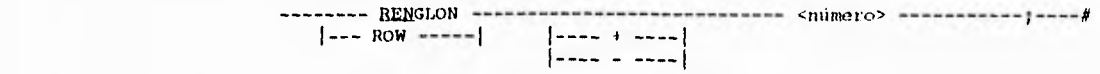
<sencia>:=



Sentencia RENGLON

En esta sección se especificará para el dibujo de líneas y cajas la posición del cursor de acuerdo a línea o renglón, así como el indicar donde se colocará texto en la forma. Es importante mencionar que aún cuando la unidad de defecto que se define en la sección de atributos sea indicada, el posicionamiento del cursor está basado en las características de alto y ancho de caracter que tenga el font 0 de la sección de fonts.

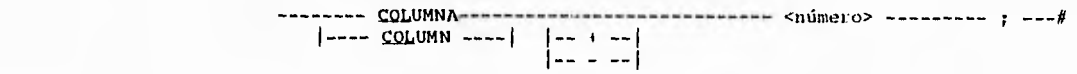
<posiciona línea>:=



La sentencia RENGLON/ROW le indicará al compilador que se posicione en un determinado renglón en un movimiento absoluto o relativo, si le especifica el caracter + o - antes del número, se moverá en forma relativa a la última posición donde se encontraba, en caso contrario lo hará de forma absoluta, esto es, irá directo al renglón especificado con número.

Sentencia COLUMNA

<posiciona columna>:=

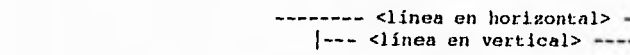


De igual manera la sentencia COLUMNA/COLUMN indicará el movimiento absoluto o relativo del cursor a través de las columnas.

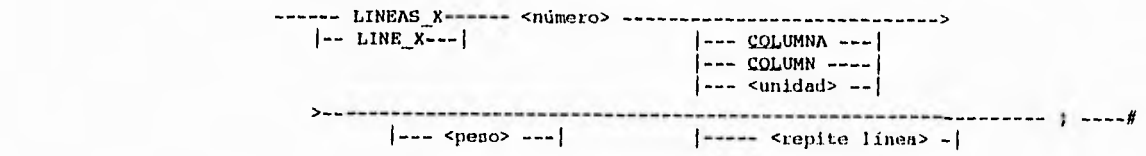
Sentencia LINEAS_X y LINEAS_Y

Ya que el diseño de formas electrónicas incluye el poder dibujar líneas, las cuales pueden ser en forma vertical o en horizontal, se propone la sentencia que se muestra a continuación:

<dibuja línea>:=



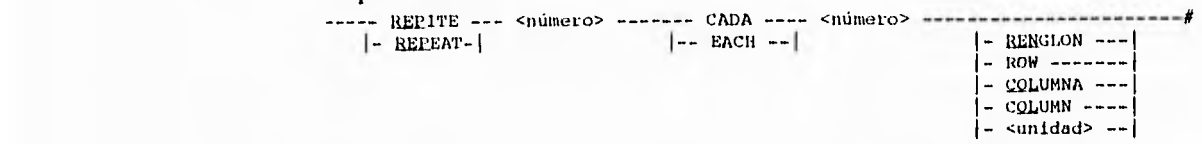
<línea en horizontal>:=



La sentencia LINEAS_X/LINE_X le indica al compilador que se desea una línea en el eje horizontal de la longitud especificada por número, si no se indica una unidad, tomará por defecto la que se especifica en la sección de atributos; pero además, es posible indicar que se quiere en COLUMNA/COLUMN, especificando la línea con el número indicado de columnas. Asimismo es posible definir el peso que deberá tener la línea o

bien definir que repita líneas del mismo tamaño en diferentes renglones, con la especificación de repetir una línea, como se indica a continuación:

<repite línea>:=

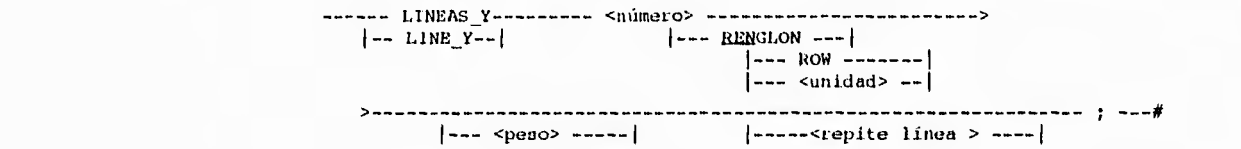


Con agregar al final de la primera parte de la sentencia, la palabra REPITE/REPEAT, se le indica al compilador que repita la línea que se especifica un N número determinado de veces cada Y número definido por COLUMNA/COLUMN o una determinada unidad, en caso de no indicarle lo hará en la unidad especificada en la sección de atributos.

Ejemplo: `LINEAS_X 3 COL PESO 1 REPITE 3 CADA 2 REN;`

Indica que dibuje una línea en el eje horizontal con una longitud de 3 columnas, de peso o grosor de 1, punto repitiendo la línea anterior 3 veces cada 2 renglones.

<línea en vertical>:=

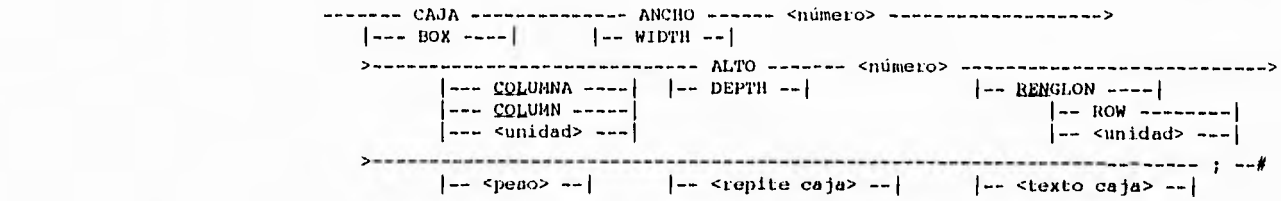


La sentencia para el dibujo de una línea vertical LINEAS_Y/LINE_Y funciona en un sentido idéntico al dibujo de las líneas horizontales, con la diferencia que aquí no se define columna sino RENGLON/ROW para indicar la longitud, en caso de indicarle esta palabra tomará como unidad de defecto la definida en la sección de atributos.

Sentencia CAJA

El dibujo de cajas puede ser realizado por líneas, pero muchas veces se desea tener una estructura que permita definir que lo que se desea dibujar es una caja o un conjunto de cajas, lo cual evita a la persona que esta diseñando una forma elaborar demasiado código, por ello se propone la siguiente sentencia para el dibujo de cajas.

<dibuja caja>:=



La sentencia CAJA/BOX indicará al compilador que se desea una caja de ANCHO/WIDTH determinado por un número N de columnas o la unidad especificada, por un ALTO/DEPTH determinado por un M número de

renglones o unidad especificada, siendo posible establecer el peso o grueso de la línea en esta sentencia o tomar el definido por defecto en la sección de atributos, así como indicarle que se desea repetir la caja un determinado número de veces, con la siguiente declaración, como parte de la misma declaración de caja.

<repite caja>:=

```
----- REPITE ------ <número> ---------------- DERECHA ------------#|
   |- REPEAT -|                              |----- AWAY ----------|
                                             |----- DEBAJO --------|
                                             |----- DOWN ----------|
```

Repetir una caja es posible realizarlo hacia la DERECHA/AWAY o bien hacia ABAJO/DOWN, ya que no tendría mucho sentido especificarle columnas o renglones, porque el ancho de la caja permite determinar cuantas columnas deberá moverse a la derecha y el alto de la misma sirve para determinar los renglones para moverse hacia abajo.

Sin embargo, un punto importante en la declaración de una caja es el poder incluir un texto dentro de la misma, para lo cual se presenta la siguiente declaración, que es parte de la misma sentencia de la caja.

<texto caja>:=

```
----- INCLUYE --- FT --- <dígito> - <justificación> - " - <cadena>- " ---#
   |- INCLUDE-|
```

La declaración INCLUYE/INCLUDE permite indicarle al compilador que se desea colocar un texto dentro de la caja o cajas, donde FT <dígito> indicará el font, previamente definido en la sección de fonts con el número especificado, que se utilizará para el texto. Es posible repetir cajas e incluir texto diferente en cada una de ellas indicando con una coma (,) entre cada texto, lo cual permite realizar cajas del mismo tamaño que van seguidas con diferente texto. Un aspecto importante, es el de poder justificar el texto dentro de la caja, agregando las siguiente palabras antes de la cadena de texto.

<justificación>:=

```
 ----- JUSTIFICA ---------------- IZQUIERDA ---------- ARRIBA --------#
   |- JUSTIFY ------|            |- DERECHA ----|     |-- ABAJO ----|
                                |- CENTRO -----|     |-- MEDIO ----|
                                |- LEET -------|     |-- UPPER ----|
                                |- RIGTH ------|     |-- LOWER ----|
                                |- CENTER------|     |-- MIDDLE ---|
```

Lo anterior permitirá mayor flexibilidad al usuario en la colocación del texto, ya que será posible definir una combinación de las palabras arriba indicadas para justificar el texto.

Ejemplo:

```
BOX WIDTH 3 DEPTH 4;
CAJA ANCHO 4 ALTO 4 PESO 2 INCLUYE FT 2 JUST IZQ MD "NUMERO";
BOX WIDTH 3 COL DEPTH 2 ROW REPEAT 2 AWAY INCLUDE FT 1 JUST LF MD "DIA"  "MES"  "AÑO";
```

Sentencia FT para texto libre

En algunas ocasiones se desea colocar texto en cualquier parte de la forma sin que este deba estar dentro de una caja, para lo cual se ha propuesto la sentencia que permita colocar un texto en el lugar donde se encuentre el cursor, como se puede apreciar continuación:

<texto>:=

```
----- FT ---------- <dígito> -------- " - <cadena> - " ---------- ; ----#
```

De esta manera la sentencia FT indicará al compilador que lo que se desea colocar en el lugar donde se encuentra el cursor, es el texto que se indica a continuación con el font indicado, el cual deberá declararse en la sección de fonts.

Ejemplo:

```
FT  2 "Todo lo que Ud. diga puede ser usado en su contra";
FT 1 "Fecha de creación";
FT 3 "Forma 324u0";
```

3.1.7 Definiciones adicionales del lenguaje

Finalmente es necesario establecer algunas definiciones adicionales para el compilador, de esta manera, se define número, cadena, alfabeto, dígito y símbolos como se establece a continuación:

<número>:=

```
----- <digito> ------------------------------------------------------------#
   |--- / 4 /-----|          |--- . ---------- <dígito> -------|
                                          |--- <dígito> -----|
```

El número para nuestro caso podrá ser entero o real, siendo entero hasta de 4 posiciones y real hasta de 4 posiciones enteras y 2 decimales. Los decimales deberán ser colocados con un cero anterior al punto cuando no se desee manejar enteros.

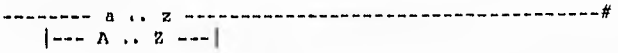
<dígito>:=

```
------- 1 -------------------#
   |---- 2 ----|
   |---- 3 ----|
   |---- 4 ----|
   |---- 5 ----|
   |---- 6 ----|
   |---- 7 ----|
   |---- 8 ----|
   |---- 9 ----|
   |---- 0 ----|
```

Los dígitos comprenden los caracteres del 0 al 9

También es muy importante determinar los caracteres que conforman el alfabeto para armar una cadena de texto o bien un nombre o identificador de forma o de font, para lo cual se define el mismo de la siguiente manera:

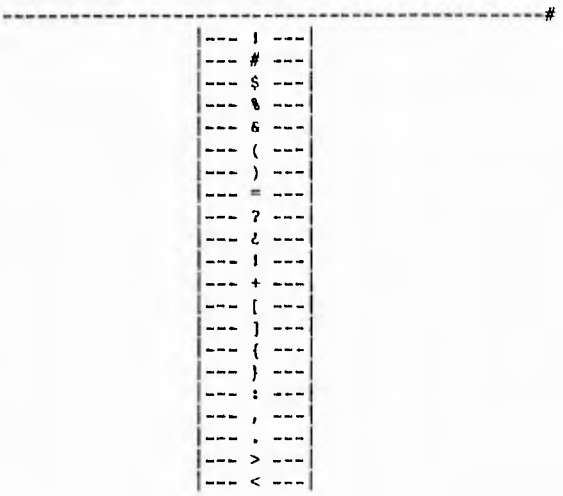
<alfabeto>:=



El alfabeto es cualquier carácter de la letra a a la z, tanto en mayúsculas como en minúsculas.

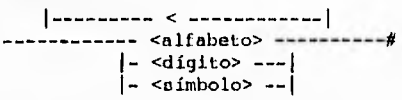
Como último punto esta definir que otros caracteres pueden ser aceptados dentro de una cadena de texto, lo que se conoce normalmente como símbolos, por lo que se define al símbolo como se muestra a continuación:

<símbolo>:=



Finalmente una cadena de caracteres para texto se puede definir como se ilustra en seguida:

<cadena>:=



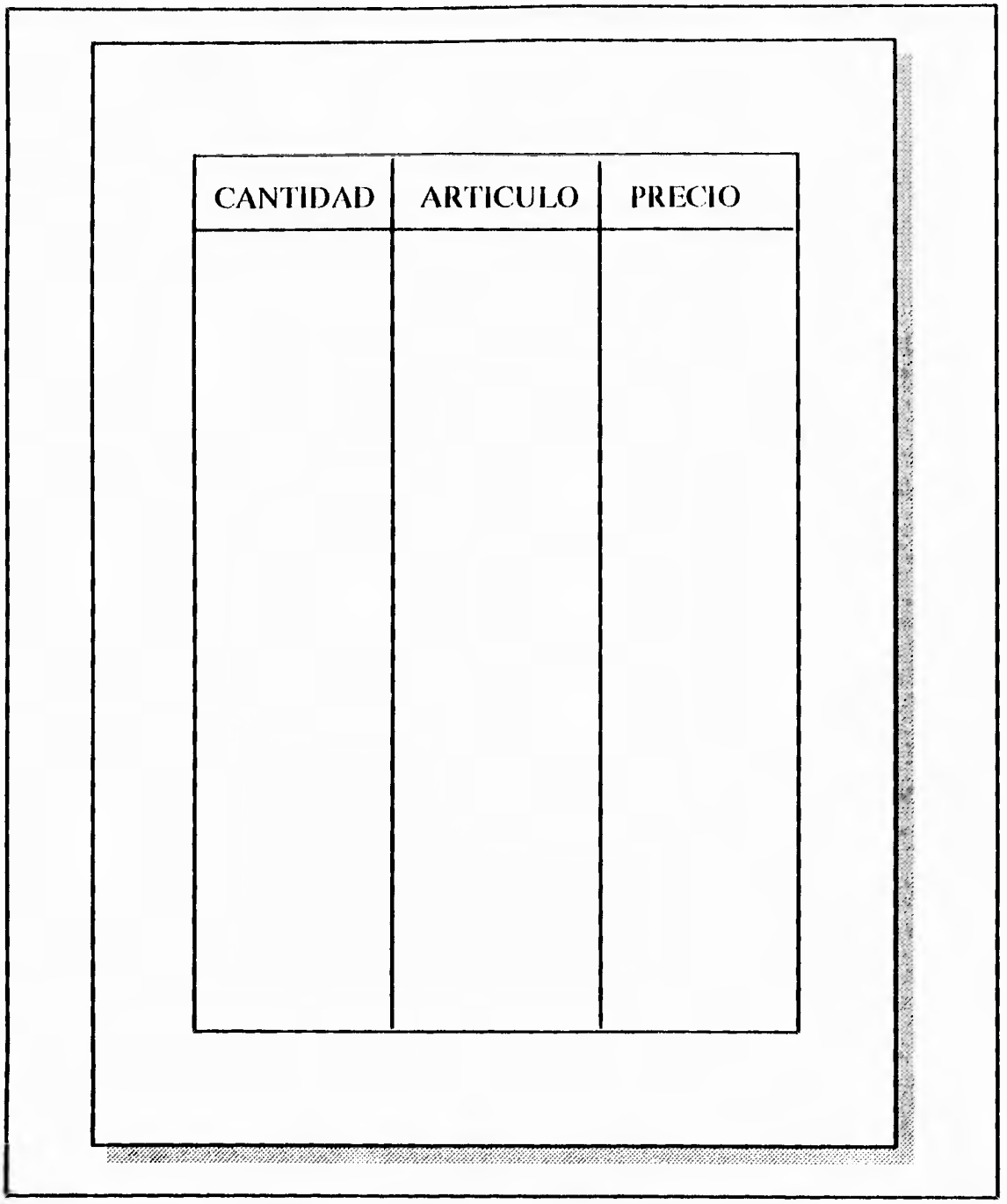
Una cadena puede estar formada de cualquier caracter del alfabeto, cualquier dígito o símbolo, o por una combinación de ellos, con una longitud máxima de 256 caracteres.

De esta manera se cubre el diseño del lenguaje propuesto para el producto; sin embargo, es importante mostrar un ejemplo de como el lenguaje puede realizar una forma y el resultado esperado por el compilador, lo cual se ilustra a continuación.

3.1.8 Ejemplo de descripción de forma

El siguiente ejemplo muestra la manera de diseñar una sencilla forma utilizando el lenguaje descriptor de formas.

```
/* forma ejemplo */
FORMA EJEMPLO;
/* Esta forma nos permite hacer una tabla con los siguientes encabezados *
 *        CANTIDAD            ARTICULO            PRECIO              *
 *                                                                     */
/* La impresora  en donde se imprimirá la forma es XEROX */
IMPRESORAS      XES;
ATRIBUTOS
   INICIO
         ORIENTACION   VERTICAL;
         TAMAÑO        CARTA;
         UNIDAD        CM;
         MARGENES
         INICIO
               SUPERIOR      0.5;
               IZQUIERDO     0.5;
               DERECHO       0.5;
               INFERIOR      0.5;
         FIN;
         PESO  2;
   FIN;
FUENTES
   XES:
         INICIO
               0      RES    Titan10iso-P
               1             Helvel2BI-P
         FIN;
/*  Descripción del cuerpo de la forma  */
DESCRIPCION
   INICIO
         RENGLON 1; COLUMNA  0;
         CAJA ANCHO  19  ALTO  24  PESO  3;
         RENGLON  1;  COLUMNA  10;
         LÍNEAS_Y  24 REPITE 1 CADA 10;
         RENGLON  3;  COLUMNA  0;
         LÍNEAS_X  18;
         RENGLON  2; COLUMNA  1;
         FT  1  "CANTIDAD";
         RENGLON  2;  COLUMNA  11;
         FT  1  "ARTICULO";
         RENGLON 2;  COLUMNA  21;
         FT  1 "PRECIO";
   FIN;
```



Ejemplo de la forma generada con el lenguaje

3.2 Biblioteca de rutinas comunes

Todo producto de software tiene ciertas rutinas o funciones que son útiles a más de un proceso y por ello es necesario contar con una biblioteca que permita el acceso a estas rutinas, minimizando el esfuerzo de diseño y programación, de esta manera las rutinas que se encuentran como comunes en este producto son las siguientes:

- Validación de vigencia de licencia
- Validación de existencia de impresora
- Validación de existencia de forma
- Validación de existencia de font
- Validación de comando
- Validación de archivo
- Despliegue de errores

Los diagramas de estructura de cada uno de las rutinas se muestra a continuación, así como el diseño detallado con pseudocódigo y la lista de variables que intervienen en cada una de las mismas.

3.2.1 Vigencia de licencia

La rutina de validación de licencia es utilizada por todos los módulos del producto; ya que para poder tener acceso se debe validar la vigencia de la licencia del producto, de tal manera que si no existe vigencia del mismo, ningún módulo podrá funcionar, asimismo debe identificar que la licencia se vencerá cuando falten 30 días o menos, a fin de que el usuario pueda adquirir la renovación.

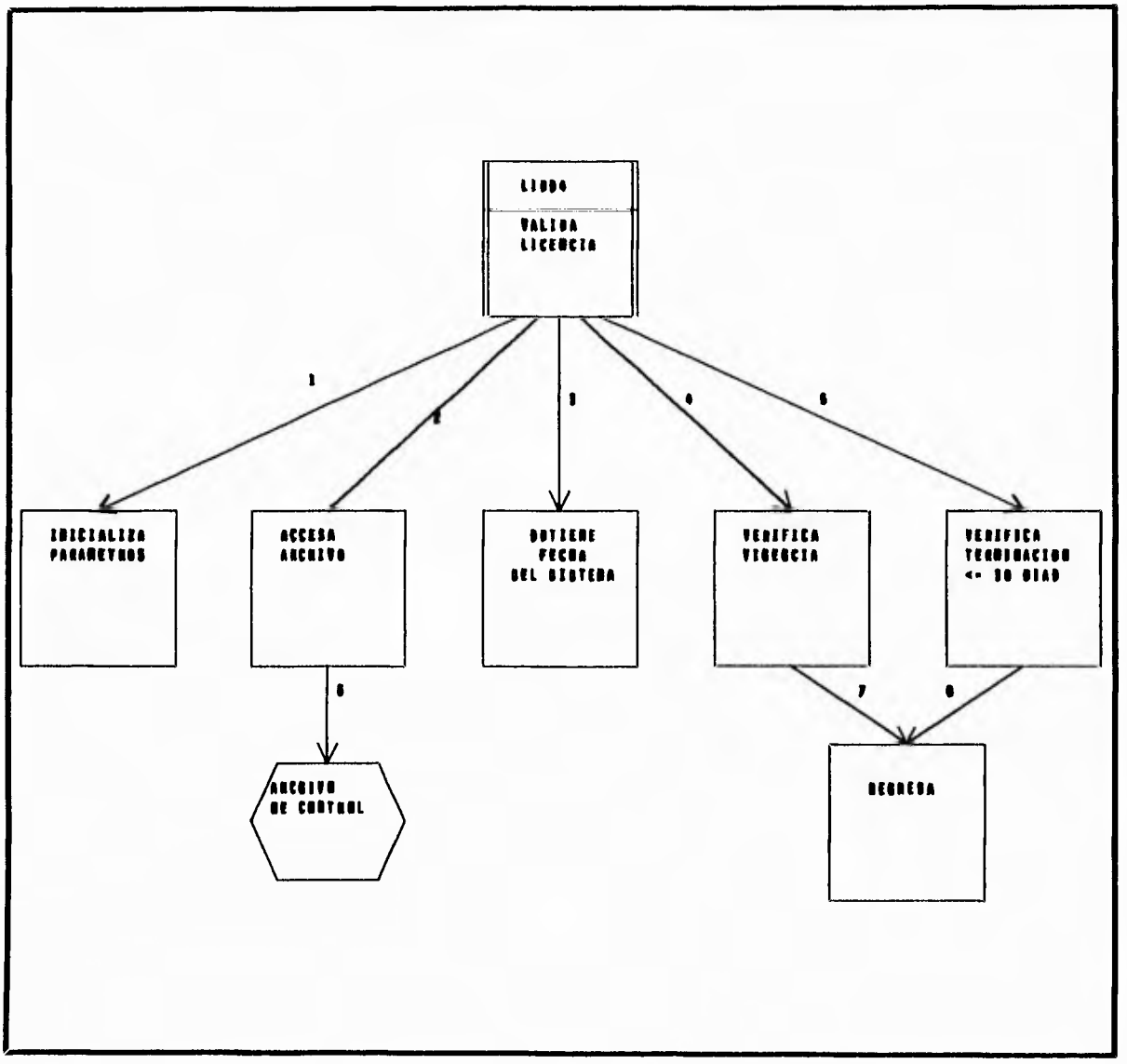


Fig. 3.1 Diagrama estructura de valida licencia

Diseño detallado en pseudocódigo

```
FUNCION VALIDA_LICENCIA
Parámetros L_LICVAL, I_NUMMEN
L_LICVAL = falso
I_NUMMEN = 0
Abrir ARC_CONTROL
Leer ARC_CONTROL
D_FECSIS=Fecha de hoy
Si D_FECSIS menor a D_FECLIC en R_CONTROL
    L_LICVAL = verdadero
sino
    I_NUMMEN = numero de mensaje licencia no valida
Fin_Si
I_DIFFEC=D_FECLIC - D_FECSIS
Si I_DIFFEC menor o igual a 30 y I_DIFFEC mayor a 0
    I_NUMMEN = numero de mensaje licencia por terminar
Fin_Si
Cerrar ARC_CONTROL
Regresa
```

3.2.2 Existencia de impresora

La rutina de validación de impresora es utilizada por todos los módulos del producto excepto por el compilador que sólo realiza la traducción de código fuente. Esta rutina es importante ya que si no es transacción de alta en el módulo de impresoras, no se podrán realizar los movimientos o acciones solicitadas.

Diseño detallado en pseudocódigo

```
FUNCION VALIDA_IMPRESORA
Parámetros CD_NOMIMP, L_VALIMP, CD_EMUIMP, I_NUMMEN
L_VALIMP = falso
I_NUMMEN = 0
Leer ARC_IMPRESORA llave CD_NOMIMP
Si Fin_Archivo es falso
    Inicio
    CD_EMUIMP = CD_EMULA en R_IMPRESORA
    L_VALIMP = verdadero
        Fin
sino
    I_NUMMEN = numero mensaje impresora no existe
Fin_Si
Regresa
```

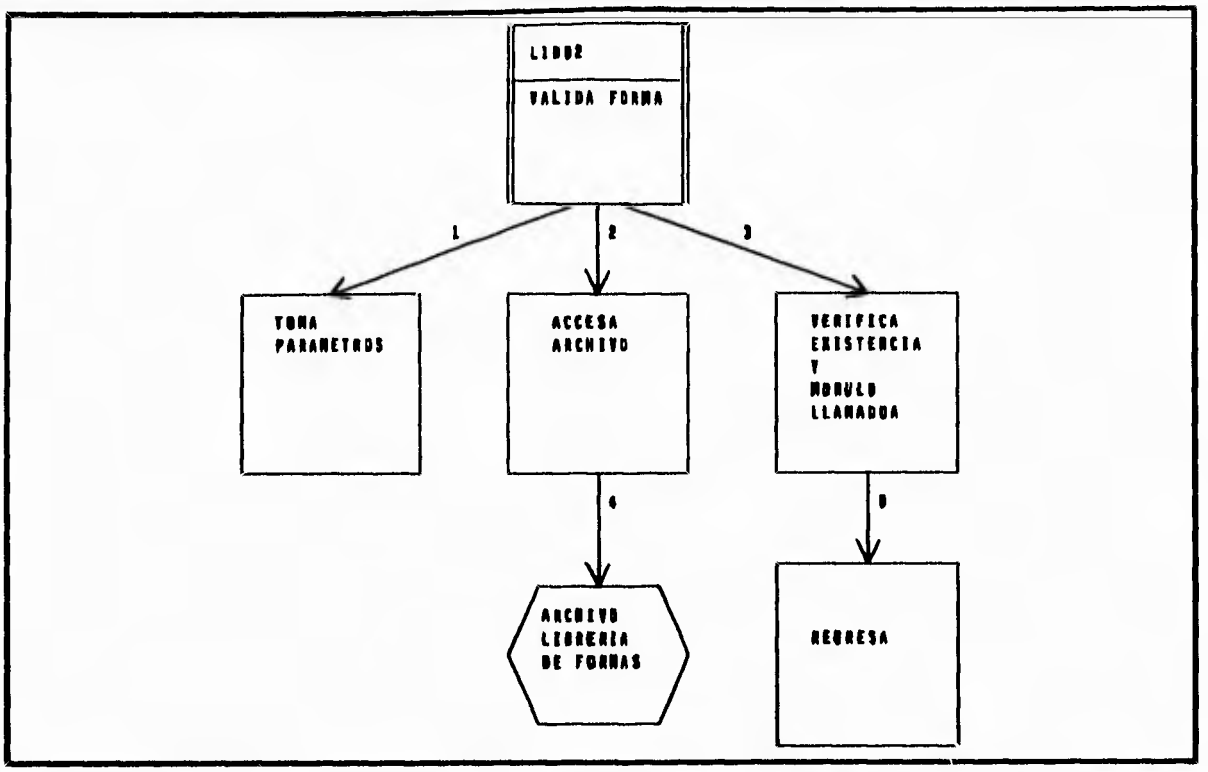


Fig. 3.2 Diagrama estructura de valida impresora

3.2.3 Existencia de forma

La rutina de validación de forma es utilizada por el módulo de impresión y por el compilador, en el primer caso, si la forma no existe o si existe y no coincide con la emulación de impresora, debe mandar un mensaje de que la misma no puede ser enviada y por lo tanto no se imprimirá el trabajo solicitado. En el caso del compilador, si ya existe la forma, se debe enviar el mensaje de sustitución por la nueva compilación y en caso contrario el mensaje será de añadir la forma a la librería.

Diseño detallado usando pseudocódigo

```
FUNCION VALIDA_FORMA
Parámetros I_NUMMOD, CD_NOMFOR, CD_EMUIMP, L_VALFOR, I_NUMMEN
L_VALFORM = falso
I_NUMMEN = 0
Abrir ARC_LIBFORMAS
Leer ARC_LIBFORMAS llave CD_NOMFOR + CD_EMUIMP
Si Fin Archivo es falso
    Inicio
    L_VALFOR = verdadero
    Si I_NUMMOD = 0                    (0=compilador)
```

```
            I_NUMMEN = numero mensaje de sustitución de forma
        sino
            I_NUMMEN = 0
        Fin_Si
        Fin
    sino
        Si I_NUMMOD = I                    (I= impresión)
            I_NUMMEN = numero mensaje de no existencia
        sino
            I_NUMMEN = numero de mensaje de alta en libreria
        Fin_Si
    Fin_Si
    Regresa
```

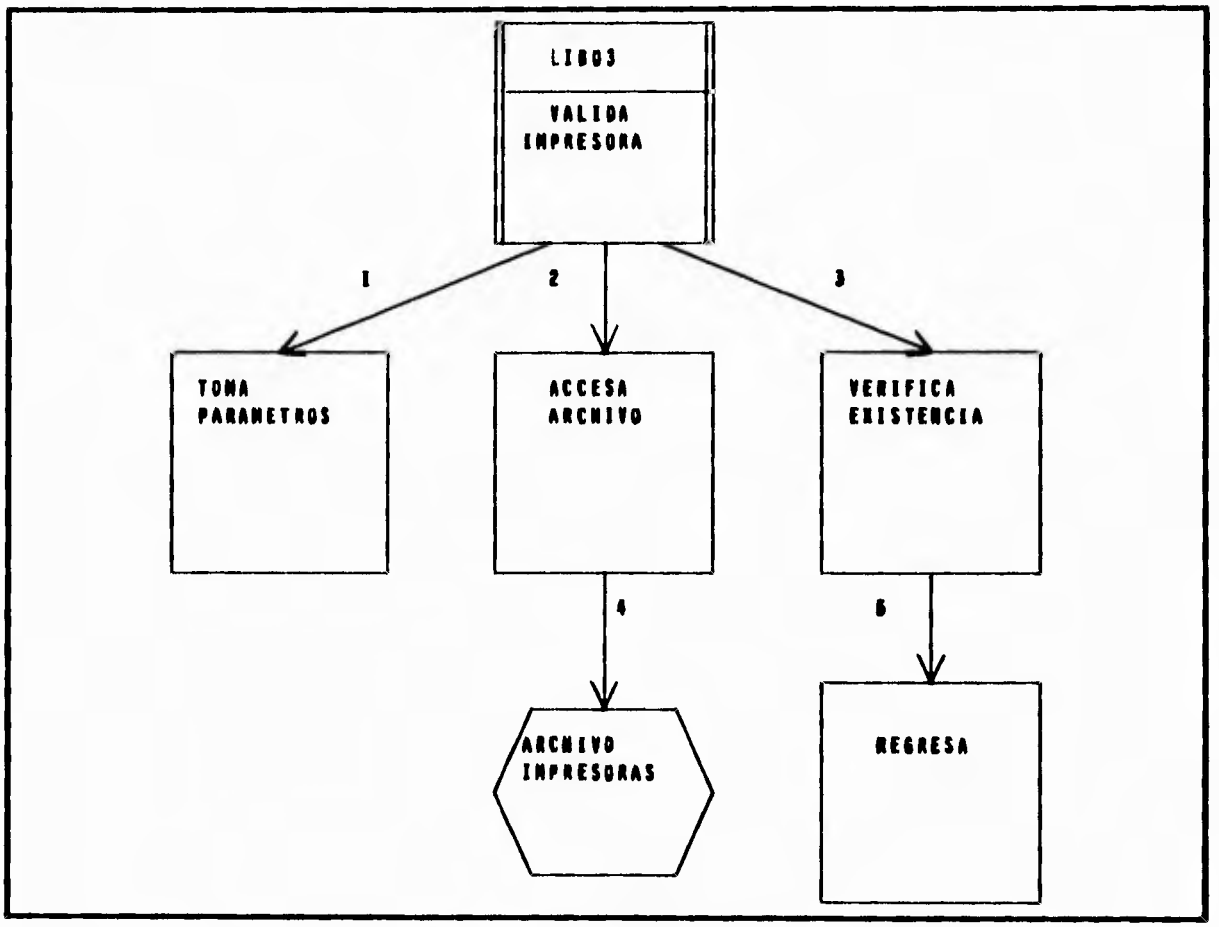


Fig. 3.3 Diagrama estructura de valida forma

3.2.4 Existencia de font

La rutina de validación de font es utilizada por el compilador y por el módulo de utilería, en el primer caso, si el font no existe debe enviar un mensaje de la no existencia para que la forma no sea generada y en el segundo caso, si el font ya existe en la librería de fonts, debe enviar un mensaje indicando de que el mismo no se agregará por ya existir.

Diseño detallado usando pseudocódigo

```
FUNCION VALIDA_FONT
Parámetros I_NUMMOD, CD_NOMFON, L_VALFON, I_NUMMEN
L_VALFON = falso
Leer ARC_LIBFONTS llave CD_NOMFON
Si Fin Archivo es falso
  Inicio
    L_VALFON = verdadero
    Si I_NUMMOD = 0                    (0 = utileria)
        I_NUMMEN = numero mensaje ya existe font en libreria
    Fin_Si
    Fin
sino
    Si I_NUMMOD = 1                    (1= compilador)
        I_NUMMEN = numero mensaje no existe font
        Fin_Si
Fin_Si
Regresa
```

3.2.5 Validación de comando

La rutina de validación de comando es utilizada por el compilador, la utilería del producto y el módulo de impresión, la principal función de ésta es realizar el rastreo y verificación sintáctica del comando.

Los comandos que se definen para este producto son los siguientes:

```
<comandos> := ----/ 1 / --------- <compilación> ------------#
                                |---- <utileria> -----|
                                |--- <impresión> -----|
```

Compilación

El comando de compilación le indicará al compilador las opciones y el nombre del archivo de forma fuente, de tal manera que las opciones validas son; -l para indicar que se desea el listado de la forma y -s para indicar que sólo se realice la revisión sintáctica de la forma fuente. De esta manera el comando se define como sigue:

```
<compilación> := <archivo> ------------------------ CR -----#
                                |-- -l --|    |-- -s --|
```

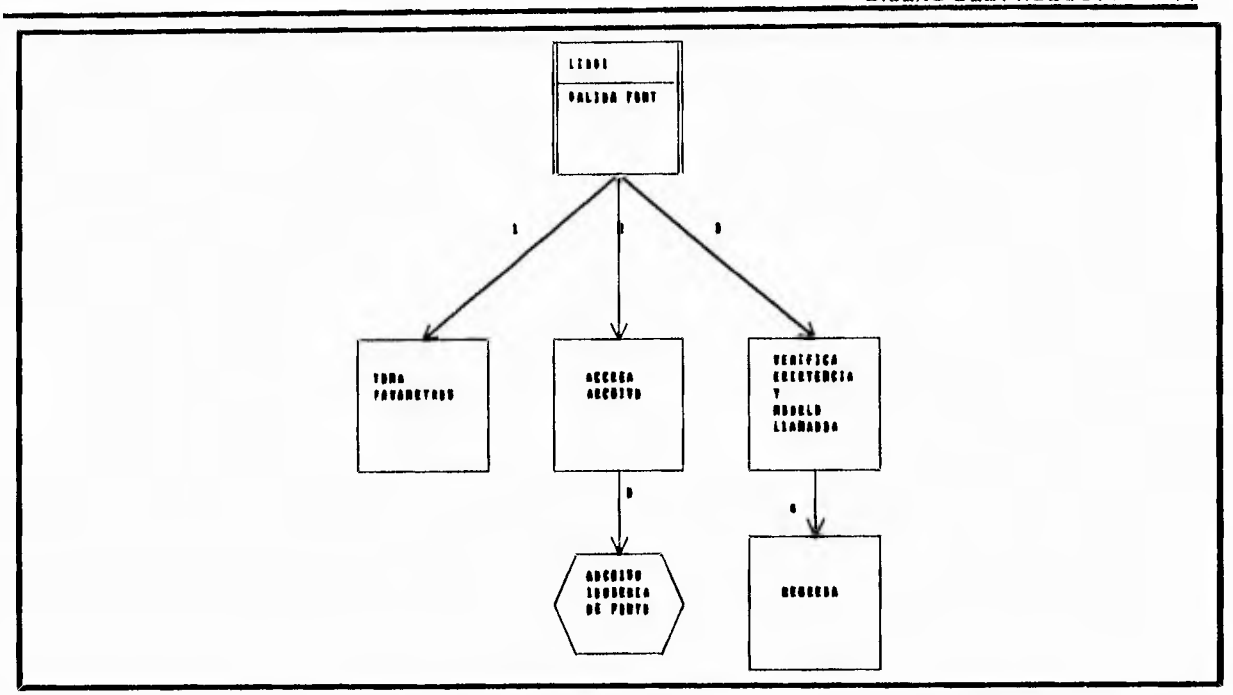


Fig. 3.4 Diagrama estructura de valida font

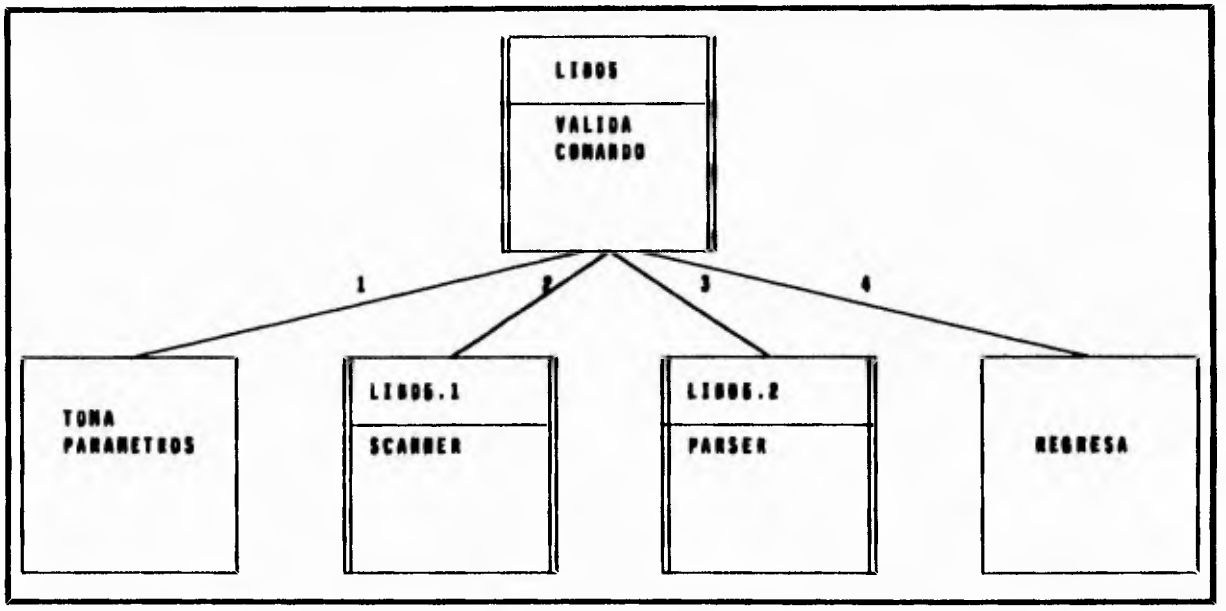


Fig. 3.5 Diagrama estructura de valida comando

Utilería

El comando de utilería permitirá saber si el usuario desea agregar fonts a la librería de fonts y copiarlos al host por medio de la opción -add, o bien obtener el listado de los fonts y formas que se tienen en el host con las opciones -lfo y -lfr respectivamente, en este caso se debe indicar a que impresora se dirigirá la salida de los listados. Por lo que el comando se define de la siguiente manera:

```
<utileria>:= ---/ 1 / ------------------------------- CR ----#
                      |---- -add -----------------|
                      |--- -lfo --- <impresora>---|
                      |--- -lfr --|
```

Impresión

El comando de impresión permite indicar al módulo lo que debe realizar, las opciones del comando -fr <forma> indicará la forma que se utilizará en la impresión; -pr <impresora> indicará en que impresora se desea la salida; -cp <número> indicará el número de copias deseadas y -sv indicará que se desea mantener el archivo de impresión. No obstante las dos últimas opciones sólo serán aplicables cuando se utilice la opción -fl <archivo> que es donde se especificará el archivo que se desea imprimir, la ausencia de esta opción se tomará como que el usuario sólo desea una muestra de la forma señalada. De esta manera el comando se define como sigue:

```
                  |------ < ----|
<impresión>:= ---- <parámetro> --- CR ---#

<parámetro>:= ------ -fr <forma> -------------------------------#
                  |-- -pr <impresora> --------------------|
                  |                      |---- < -----|      |
                  |-- -fl <archivo> ---- <opción> --------|

<opción>:= -------------------------------#
                  |-- -cp <numeró> --|
                  |-- -sv -----------|
```

Una vez definidos los comandos que serán aceptados para cada uno de los módulos que integran el producto, se procede a realizar el diseño del analizador de comandos, y como se puede apreciar se requiere de un pseudo-compilador para que se realice efectivamente la función deseada, según puede apreciarse en la figura siguiente, donde se tiene la estructura de la función de validación de comando.

3.2.6 Diseño del analizador del comando

Una vez que se han definido los comandos que serán utilizados, es necesario obtener la lista de los tokens que conforman las opciones de los comandos, por lo que a continuación se tiene la lista que muestra los tokens y su representación numérica para los comandos.

Número Token	token
10	-l
20	-s
30	-add
40	-lfo
50	-lfr
60	-fl
70	-fr
80	-pr
90	-cp
100	-sv
110	<identificador>
120	<constante numérica>
130	CR (Carriage Return)

Se puede observar que en total son 13 tokens, ya que los incrementos son de 10 en 10 y con ello se procede a elaborar los comandos, para determinar como sería la sintaxis que el verificador de comandos debe reconocer. Por ejemplo, el siguiente comando será reconocido por el scanner de validación en la manera que se muestra a continuación.

-fl notas.prt -fr default -pr laser1 -cp 2 -sv <CR>

El scanner de validación generará la siguiente secuencia de tokens.

60 110 70 110 80 110 90 120 100 130

El parser de validación recibirá la secuencia de tokens que generó el scanner de validación, para determinar la validez del comando, es por ello, que la estructura sintáctica del parámetro de contando se definió en la representación numérica de los tokens.

```
<compilación> := 110  ------------------------------- 130 ------#
                       |--- 10 ---|  |--- 20 --|



<utileria>:= ---/ 1 / ------------------------------- 130 -----#
                       |--- 30 -----------------|
                       |--- 40 ----- 110 --------|
                       |--- 50 --|

                    |----- < -----|
<impresión>:= ---- <parámetro> --- 130 ----#

<parámetro>:= ------ 70 -- 110 ----------------------#
                    |-- 80 -- 110 --------------------|
                                          |--- < ---|   |
                    |-- 60 -- 110 ------- <opción> ---|
```

```
<opción>:= --------------------------#
                  |-- 90 -- 120 --|
                  |-- 100 --------|
```

La principal función del scanner de validación, será convertir los comandos en una secuencia de tokens representados por su valor numérico y puesto que debe leerlo como una cadena de caracteres, carácter por carácter, y reconocer o aceptar todos los tokens, el diseño se hará en base al modelo matemático denominado aceptor de estado finito o autómata finito. La figura 3.6 muestra el diagrama de estado finito para los comandos, construido a partir de las estructuras sintácticas y de la lista de tokens de los comandos. Los nodos en el diagrama representan los estados del autómata, los arcos representan el estado de transición, el o los caracteres asociados a estos arcos indican la entrada que causa el estado de transición. En este diagrama el estado cero (0) representa el estado inicial y los estados que consisten de un par de círculos concéntricos representan los estados finales.

Los espacios se tratarán como separador de token, de esta forma si alguno es detectado, se permanecerá en el estado actual o se regresará al estado inicial. Los parámetros serán reconocidos por los estados 2 y 3, los cuales permitirán identificar los primeros 10 tokens previamente especificados. El estado 4 servirá para reconocer un identificador, el cual puede ser un nombre de impresora, de forma, o de archivo. El estado 5 reconocerá un valor numérico entero y el estado 6 permite reconocer el carriage return (CR), que sirve como terminador de comando. El arco que se etiqueta "error" indica que a detectado un carácter invalido en el comando.

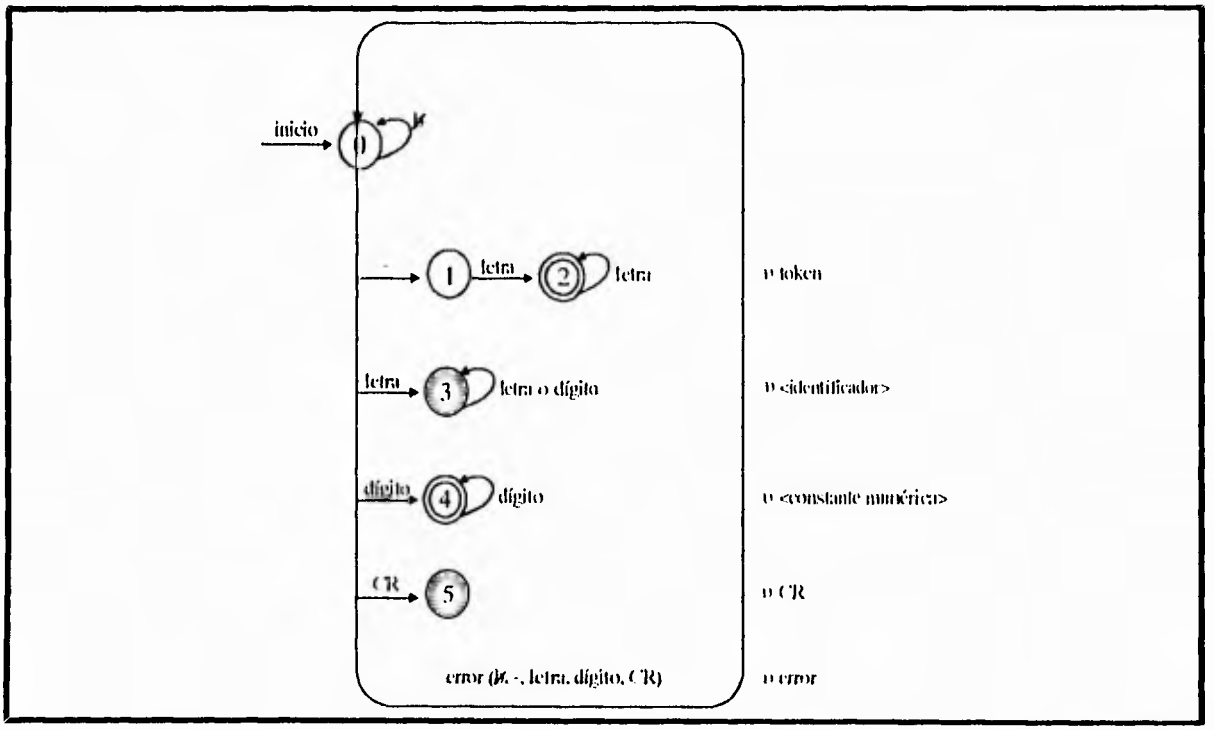


Fig. 3.6 Diagrama automata para validar comando

El conjunto de funciones que el scanner debe realizar son las siguientes:

GET_CHAR para obtener el siguiente carácter de la entrada del comando; KEYWORD determinará si el token es un atributo, en caso afirmativo regresará su valor numérico, de lo contrario regresará un cero indicando la obtención de un identificador; INSERT insertará en la tabla cada identificador o constante reconocido en el comando devolviendo la posición donde ésta se inserto; PUT_TOKEN escribirá la secuencia de salida en representación numérica de los tokens y cuando aplica también escribirá la posición en la tabla donde el token fue registrado.

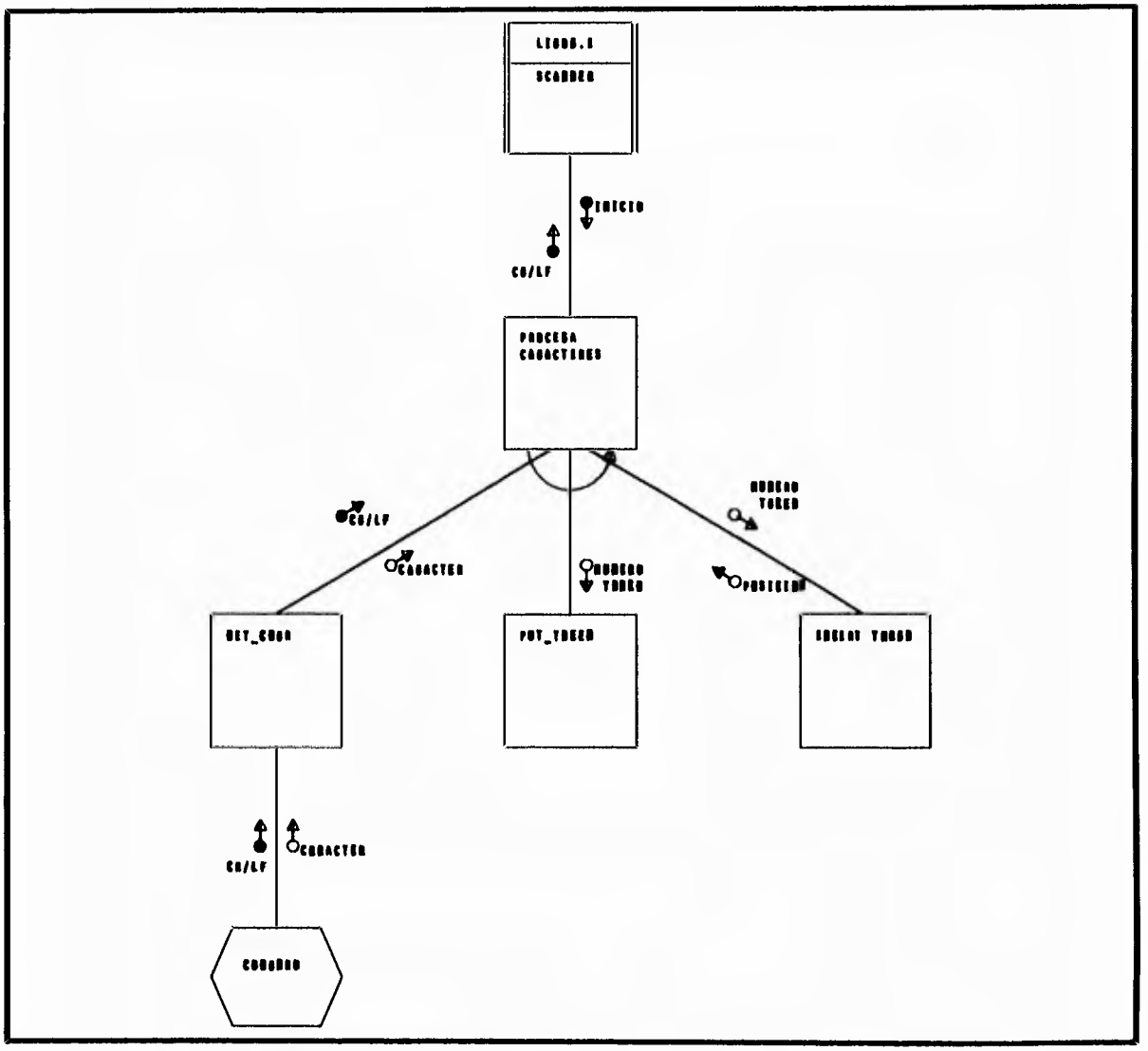


Fig. 3.7 Diagrama estructura scanner valida comando

Basados en el diagrama de estado finito o autónata y en el diagrama de estructura anterior se procede a diseñar detalladamente con pseudocódigo.

```
FUNCION SCANNER
PARAMETROS CD_COMANDO, L_COMVAL, I_NUMERR, IA_TOKENS, CDA_IDS
I_NUMEDO = 0
I_TOKEN = 0
I_NUMPOS = 0
C_CARACT = BLANCO
CD_TOKEN = NULO
L_COMVAL = F
L_FIN = F
REPITE
    Si I_NUMEDO = 0 entonces
        C_CARACT = GET_CHAR(comando)
        Si C_CARACT = BLANCO entonces
            I_NUMEDO = 0
        Si C_CARACT = "-" entonces
            I_NUMEDO = 1
            CD_TOKEN = CD_TOKEN + C_CARACT
        Si C_CARACT = LETRA entonces
            I_NUMEDO = 3
            CD_TOKEN = CD_TOKEN + C_CARACT
        Si C_CARACT = DIGITO entonces
            I_NUMEDO = 4
            CD_TOKEN = CD_TOKEN + C_CARACT
        Si C_CARACT = CR entonces
            I_NUMEDO = 5
        Si C_CARACT NO= "-"o LETRA o DIGITO o CR entonces
            Genera mensaje de error
    Si I_NUMEDO = 1 entonces
        C_CARACT = GET_CHAR(comando)
        Si C_CARACT = LETRA entonces
            I_NUMEDO = 2
            CD_TOKEN = CD_TOKEN + C_CARACT
        Si C_CARACT NO= LETRA entonces
            I_NUMEDO= 0
            Genera mensaje de error
    Si I_NUMEDO = 2 entonces
        C_CARACT = GET_CHAR(comando)
        Si C_CARACT = LETRA entonces
            I_NUMEDO = 2
            CD_TOKEN = CD_TOKEN + C_CARACT
        Si C_CARACT = BLANCO entonces
            I_NUMEDO = 0
            I_TOKEN = KEYWORD(CD_TOKEN)
            Si I_TOKEN NO= 0 entonces
                PUT_TOKEN(I_TOKEN)
            Si I_TOKEN = 0 entonces
                I_TOKEN = 110
```

```
                    I_NUMPOS = INSERT(CD_TOKEN,I_TOKEN)
                    PUT_TOKEN(I_TOKEN,I_NUMPOS)
                    CD_TOKEN = NULO
            Si C_CARACT NO= LETRA o BLANCO o CR entonces
                I_NUMEDO = 0
                CD_TOKEN = NULO
                Genera mensaje de error
        Si I_NUMEDO = 3 entonces
            C_CARACT = GET_CHAR(comando)
            Si C_CARACT = LETRA o DIGITO entonces
                I_NUMEDO = 3
                CD_TOKEN = CD_TOKEN + C_CARACT
            Si C_CARACT = BLANCO entonces
                I_NUMEDO = 0
                I_TOKEN = 110
                I_NUMPOS = INSERT(CD_TOKEN,I_TOKEN)
                PUT_TOKEN(I_TOKEN,I_NUMPOS)
                CD_TOKEN = NULO
            Si C_CARACT NO= LETRA o DIGITO o BLANCO o CR entonces
                I_NUMEDO = 0
                CD_TOKEN = NULO
                Genera mensaje de error
        Si I_NUMEDO = 4 entonces
            C_CARACT = GET_CHAR(comando)
            Si C_CARACT = DIGITO entonces
                I_NUMEDO = 4
                CD_TOKEN = CD_TOKEN + C_CARACT
            Si C_CARACT = BLANCO entonces
                I_NUMEDO = 0
                I_TOKEN = 120
                I_NUMPOS = INSERT(CD_TOKEN,I_TOKEN)
                PUT_TOKEN(I_TOKEN,I_NUMPOS)
                CD_TOKEN = NULO
            Si C_CARACT NO= DIGITO o BLANCO o CR entonces
                I_NUMEDO = 0
                CD_TOKEN = NULO
                Genera mensaje de error
        Si I_NUMEDO = 5 entonces
                I_TOKEN = 130
                I_NUMPOS = INSERT(CD_TOKEN,I_TOKEN)
                PUT_TOKEN(I_TOKEN,I_NUMPOS)
                CD_TOKEN = NULO
                    I_FIN = T
                    I_COMVAL = T
    Mientras NO I_FIN
    FIN SCANNER
```

Una vez que se ha diseñado el scanner de validación, el siguiente paso es diseñar el parser o analizador sintáctico del comando. Para ello y valiéndose de un diagrama de estado finito se obtiene lo que esta ilustrado en la figura 3.8.

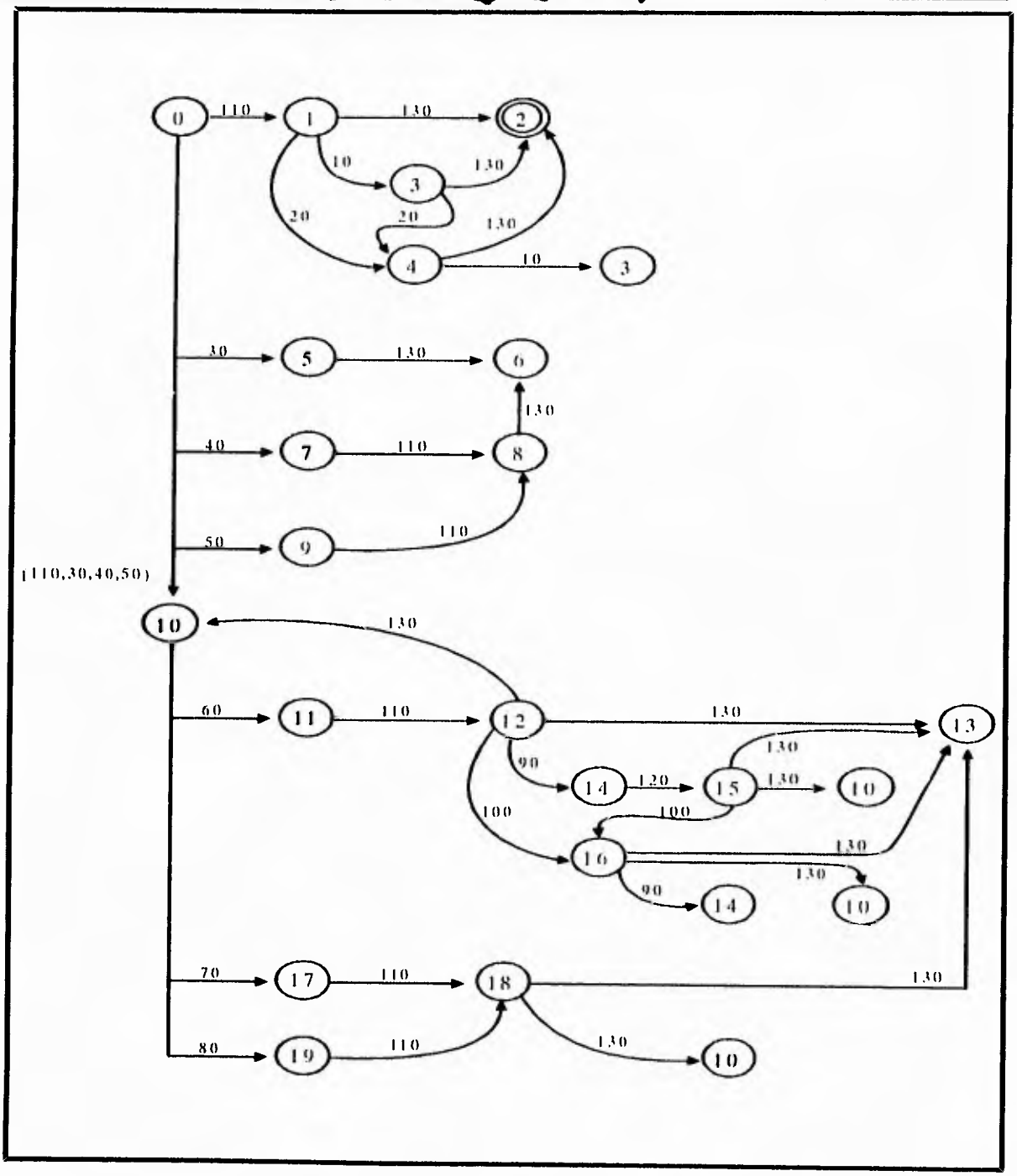


Fig. 3.8 Automata de parser para validar comando

Se observa que los parámetros del comando de compilación se reconocen en los estados 1, 2, 3 y 4, siendo el estado 1 la identificación del archivo de forma fuente y como opciones de compilación se reconocerán los estados 3 y 4.

Del comando de utilería los parámetros serán reconocidos por los estados 5 para agregar fonts, 7 y 9 para las opciones de listado de fonts y formas respectivamente, junto con el estado 8 para reconocer el nombre de la impresora donde se obtendrá.

En el caso de comando de impresión, los estados del 10 al 19 identificarán las opciones e identificadores necesarios. Los errores se generarán de acuerdo al seguimiento del diagrama. El diagrama de estructura para esta función de parser de validación, se muestra en la siguiente figura 3.9

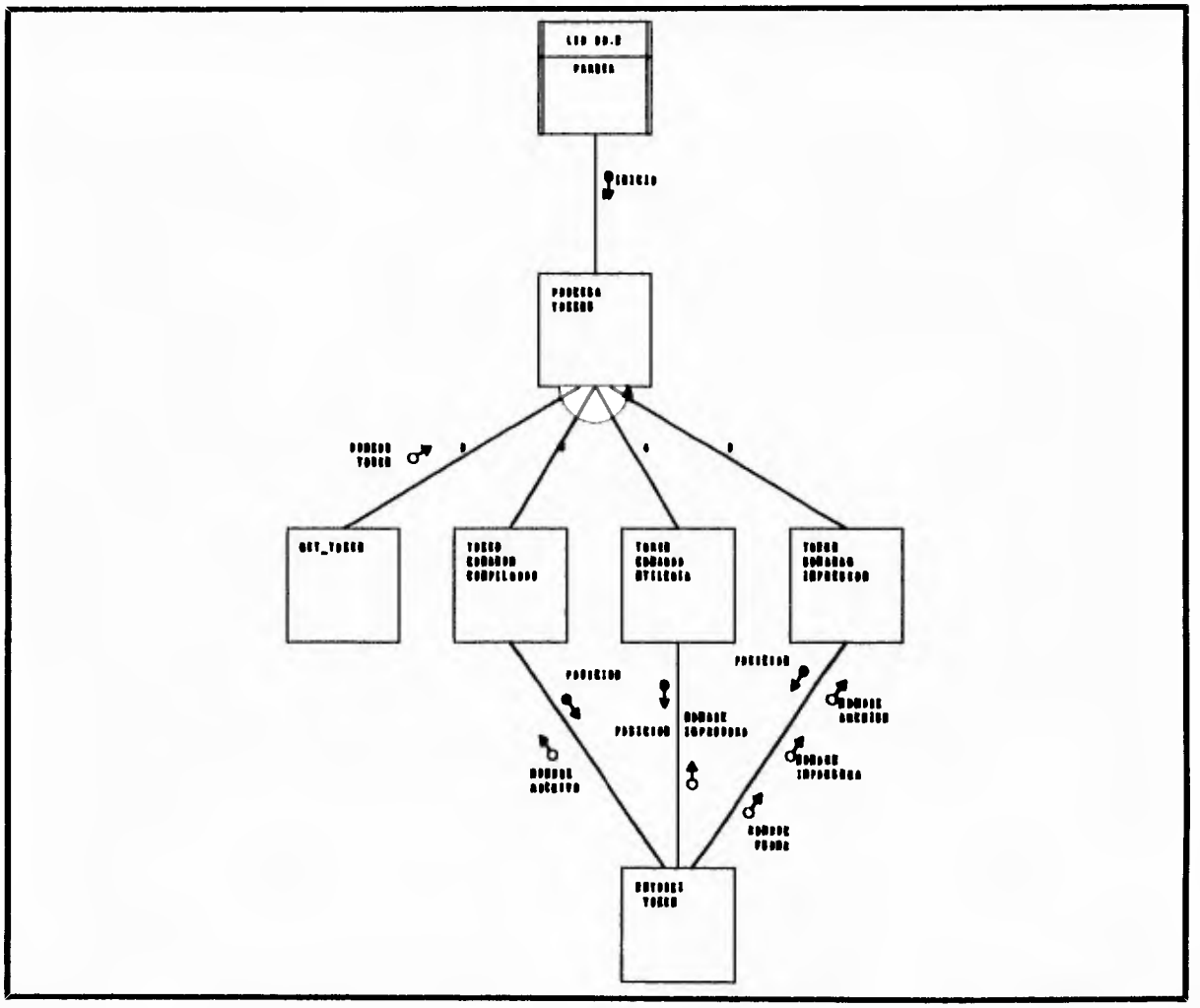


Fig. 3.9 Diagrama estructura parser de valida comando

El diseño detallado del parser se establece a continuación:

```
FUNCION PARSER
PARAMETROS I_NUMMOD, IA_TOKENS, CDA_IDS, L_COMVAL, I_NUMERR, CD_FILE, CD_FORMA, CD_PRINTER,
I_NUMCOP, L_SAVE, CD_OPC1, CD_OPC2
I_TOKEN = 0
I_NUMEDO = 0
I_FORMA = 0
I_ARCHIVO = 0
I_PRINT = 0
I_SAVE = 0
I_COPIA = 0
I_SINTAX = 0
I_LISTA = 0
L_TERMINA = F
L_COMVAL = F
L_SAVE = F
Repite
    Si I_NUMEDO = 0 entonces
        I_TOKEN = GET_TOKEN(Lista tokens)
        Si I_TOKEN = 110 entonces
            CD_FILE = EXTRACT(I_TOKEN)
            I_NUMEDO = 1
        Si I_TOKEN = 30 entonces
            I_NUMEDO = 2
        Si I_TOKEN = 40 entonces
            I_NUMEDO = 7
        Si I_TOKEN = 50 entonces
            I_NUMEDO = 9
        Si I_NUMEDO NO= 110 o 30 o 40 o 50 entonces
            I_NUMEDO = 10
    Si I_NUMEDO = 1 entonces
        I_TOKEN = GET_TOKEN(Lista Tokens)
        Si I_TOKEN = 10 entonces
            I_LISTA = I_LISTA + 1
            CD_OPC1 = EXTRACT(I_TOKEN)
            I_NUMEDO = 3
        Si I_TOKEN = 20 y I_SINTAX = 0 entonces
            I_SINTAX = I_SINTAX + 1
            CD_OPC2 = EXTRACT(I_TOKEN)
            I_NUMEDO = 4
        Si I_TOKEN = 130 entonces
            I_NUMEDO = 2
        Si I_LISTA  1 o I_SINTAX  1 o
            I_TOKEN NO= 10 o 20 o 130 entonces
            Genera mensaje de error
            L_TERMINA = T
    Si I_NUMEDO = 2 entonces
        Si I_NUMMOD = 1
            L_TERMINA = T
```

```
                L_COMVAL= T
Si I_NUMEDO = 3 entonces
    I_TOKEN = GET_TOKEN (Lista Tokens)
    Si I_TOKEN = 20 entonces
        I_SINTAX = I_SINTAX + 1
        CD_OPC2 = EXTRACT(I_TOKEN)
        I_NUMEDO = 4
    Si I_TOKEN = 130 entonces
        I_NUMEDO = 2
    Si I_SINTAX  1 o
        I_TOKEN NO= 20 o 130 entonces
        Genera mensaje de error
        L_TERMINA = T
Si I_NUMEDO = 4 entonces
    I_TOKEN = GET_TOKEN (Lista Tokens)
    Si I_TOKEN = 10 entonces
        CD_OPC1 = EXTRACT(I_TOKEN)
        I_LISTA = LISTA + 1
        I_NUMEDO = 3
    Si I_TOKEN = 130 entonces
        I_NUMEDO = 2
    Si I_LISTA  1 o
        I_TOKEN NO= 10 o 130 entonces
        Genera mensaje de error
        L_TERMINA = T
Si I_NUMEDO = 5 entonces
    I_TOKEN = GET_TOKEN (Lista Tokens)
    Si I_TOKEN = 130 entonces
        I_NUMEDO = 6
    Si I_TOKEN NO= 130 entonces
        Genera mensaje de error
        L_TERMINA = T
Si I_NUMEDO = 6 entonces
    Si I_NUMMOD = 2
        L_TERMINA = T
            L_COMVAL= T
Si I_NUMEDO = 7 entonces
    I_TOKEN = GET_TOKEN (Lista Tokens)
    Si I_TOKEN = 110 entonces
        CD_PRINTER = EXTRACT(I_TOKEN)
        I_NUMEDO = 8
    Si I_TOKEN NO= 110 entonces
        Genera mensaje de error
        L_TERMINA = T
Si I_NUMEDO = 8 entonces
    I_TOKEN = GET_TOKEN (Lista Tokens)
    Si I_TOKEN = 130 entonces
        I_NUMEDO = 6
    Si I_TOKEN NO= 130 entonces
        Genera mensaje de error
```

```
            L_TERMINA = T
    Si I_NUMEDO = 9 entonces
        I_TOKEN = GET_TOKEN (Lista Tokens)
        Si I_TOKEN = 110 entonces
            I_NUMEDO = 8
        Si I_TOKEN NO= 110 entonces
            Genera mensaje de error
            L_TERMINA = T
    Si I_NUMEDO = 10 entonces
        Si I_TOKEN = 60 entonces
            I_ARCHIVO = I_ARCHIVO + 1
            I_NUMEDO = 11
        Si I_TOKEN = 70 entonces
            I_FORMA = I_FORMA + 1
            I_NUMEDO = 17
        Si I_TOKEN = 80 entonces
            I_PRINTER = I_PRINTER + 1
            I_NUMEDO = 19
        Si I_ARCHIVO  1 o I_FORMA  1  o I_PRINTER  1 o
            I_TOKEN NO= 60 o 70 o 80 entonces
            Genera mensaje de error
            L_TERMINA = T
    Si I_NUMEDO = 11 entonces
        I_TOKEN = GET_TOKEN (Lista Tokens)
        Si I_TOKEN = 110 entonces
            CD_FILE = EXTRACT(I_TOKEN)
            I_NUMEDO = 12
        Si I_TOKEN NO= 110 entonces
            Genera mensaje de error
            L_TERMINA = T
    Si I_NUMEDO = 12 entonces
        I_TOKEN = GET_TOKEN (Lista Tokens)
        Si I_TOKEN = 90 entonces
            I_COPIA = I_COPIA + 1
            I_NUMEDO = 14
        Si I_TOKEN NO= 100 entonces
            I_SAVE = I_SAVE + 1
            I_NUMEDO = 16
        Si I_TOKEN = 130 entonces
            I_NUMEDO = 13
        Si I_COPIA  1 o I_SAVE  1 entonces
            Genera mensaje de error
            L_TERMINA = T
        Si I_TOKEN NO= 90 o 100 o 130 entonces
            I_NUMEDO = 10
    Si I_NUMEDO = 13 entonces
        Si I_NUMMOD = 3
            L_TERMINA = T
                L_COMVAL= T
    Si I_NUMEDO = 14 entonces
```

```
        I_TOKEN = GET_TOKEN (Lista Tokens)
        Si I_TOKEN = 120 entonces
            I_NUMCOP = EXTRACT(I_TOKEN)
            I_NUMEDO = 15
        Si I_TOKEN NO= 120 entonces
            Genera mensaje de error
            I_TERMINA = T
    Si I_NUMEDO = 15 entonces
        I_TOKEN = GET_TOKEN (Lista Tokens)
        Si I_TOKEN = 100 entonces
                I_SAVE = T
            I_SAVE = I_SAVE + 1
            I_NUMEDO = 16
        Si I_TOKEN = 130 entonces
            I_NUMEDO = 13
        Si I_TOKEN NO= 100 o 130 entonces
            I_NUMEDO = 10
        Si I_SAVE  1 entonces
            Genera mensaje de error
            I_TERMINA = T
    Si I_NUMEDO = 16 entonces
        I_TOKEN = GET_TOKEN (Lista Tokens)
        Si I_TOKEN = 90 entonces
            I_COPIA = I_COPIA + 1
            I_NUMEDO = 14
        Si I_TOKEN = 130 entonces
            I_NUMEDO = 13
        Si I_TOKEN NO= 90 o 130 entonces
            I_NUMEDO = 10
        Si I_COPIA  1 entonces
            Genera mensaje de error
            I_TERMINA = T
    Si I_NUMEDO = 17 entonces
        I_TOKEN = GET_TOKEN (Lista Tokens)
        Si I_TOKEN = 110 entonces
            CD_NOMFOR = EXTRACT(I_TOKEN)
            I_NUMEDO = 18
        Si I_TOKEN NO= 110 entonces
            Genera mensaje de error
            I_TERMINA = T
    Si I_NUMEDO = 18 entonces
        I_TOKEN = GET_TOKEN (Lista Tokens)
        Si I_TOKEN = 130 entonces
            I_NUMEDO = 13
        Si I_TOKEN NO= 130 entonces
            I_NUMEDO = 10
    Si I_NUMEDO = 19 entonces
        I_TOKEN = GET_TOKEN (Lista Tokens)
        Si I_TOKEN = 110 entonces
            CD_PRINTER = EXTRACT(I_TOKEN)
```

```
                I_NUMEDO = 18
            Si I_TOKEN NO= 110 entonces
                Genera mensaje de error
                I_TERMINA = T
Mientras NO I_TERMINA
FIN PARSER
```

De esta manera se procede a realizar la rutina principal de verificacion de comando, de la siguiente manera:

```
FUNCION COMANDO
PARAMETROS I_NUMMOD,CD_COMANDO, I_COMVAL, I_NUMERR, CD_FILE, CD_FORMA, CD_PRINTER, I_NUM-
COP, I_SAVE, CD_OPC1, CD_OPC2
SCANNER(CD_COMANDO,I_COMVAL,I_NUMERR,IA_TOKENS,CDA_IDS)
Si I_COMVAL entonces
    PARSER(I_NUMMOD, IA_TOKENS, CDA_IDS, I_COMVAL, I_NUMERR, CD_FILE, CD_FORMA, CD_PRINTER, I_NUM-
COP, I_SAVE, CD_OPC1, CD_OPC2)
FinSi
Regresa
```

3.2.7 Validación de archivo

La rutina de validación de archivo es utilizada por el compilador y por el módulo de impresión, es muy importante verificar la existencia del archivo; ya que el compilador no deberá realizar la compilación si no existe el archivo y el módulo de impresión no deberá iniciar la impresión si el archivo solicitado no existe.

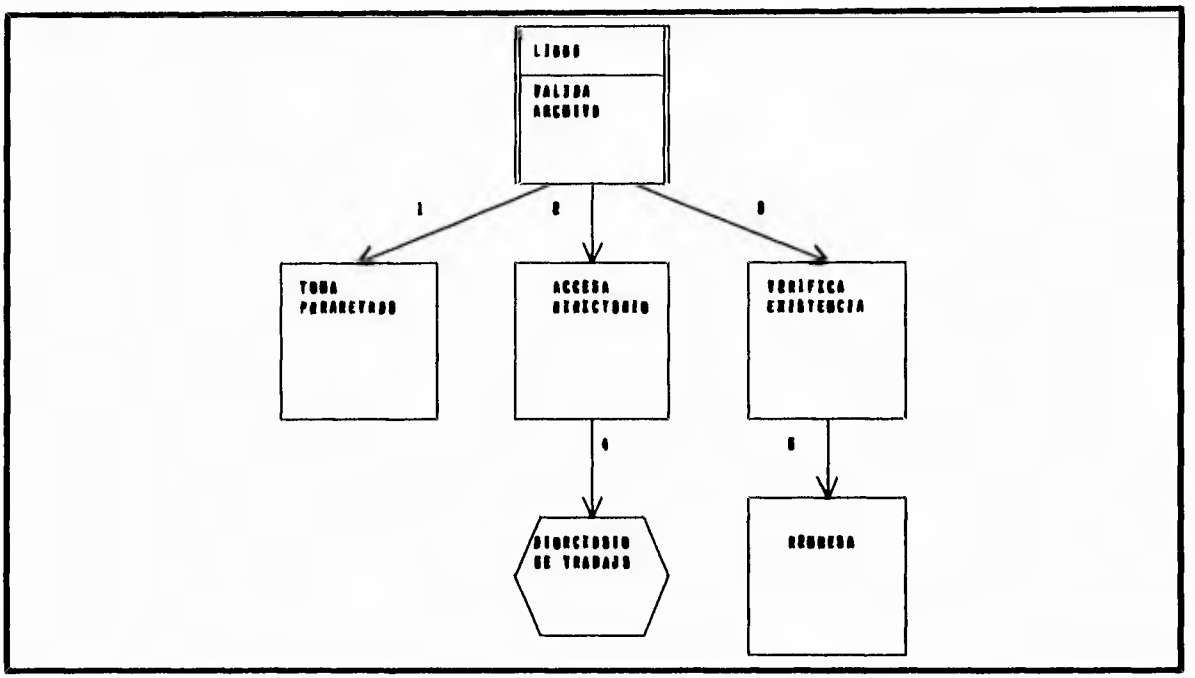


Fig. 3.10 Diagrama de estructura de valida archivo

El pseudocódigo para esta función es el siguiente:

```
FUNCION VALIDA_ARCHIVO
Parámetros CD_NOMARC, L_VALARC, I_NUMMEN
L_VALARC = falso
I_NUMMEN = 0
Leer Directorio (CD_NOMARC)
Si Fin_Archivo es falso
    Inicio
        L_VALARC = verdadero
    Fin
sino
    I_NUMMEN = numero mensaje iarchivo no existe
Fin_Si
Regresa
```

3.2.8 Manejo de mensajes de error

Se tendrá una rutina para el manejo de los mensajes de error, los cuales se desplegarán al usuario de acuerdo con el número que lo identifica, de esta manera la rutina solamente recibirá el número de mensaje y lo desplegará al usuario.

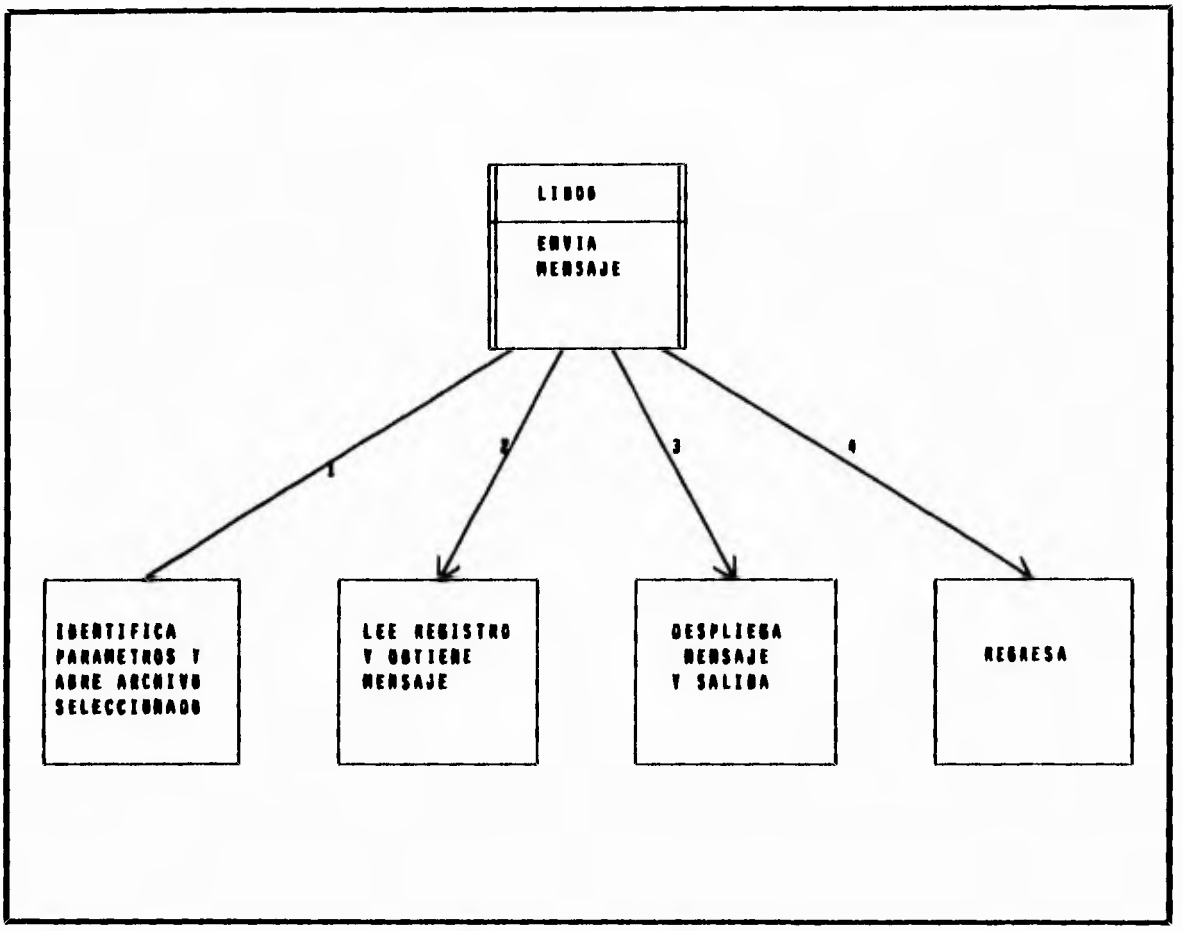
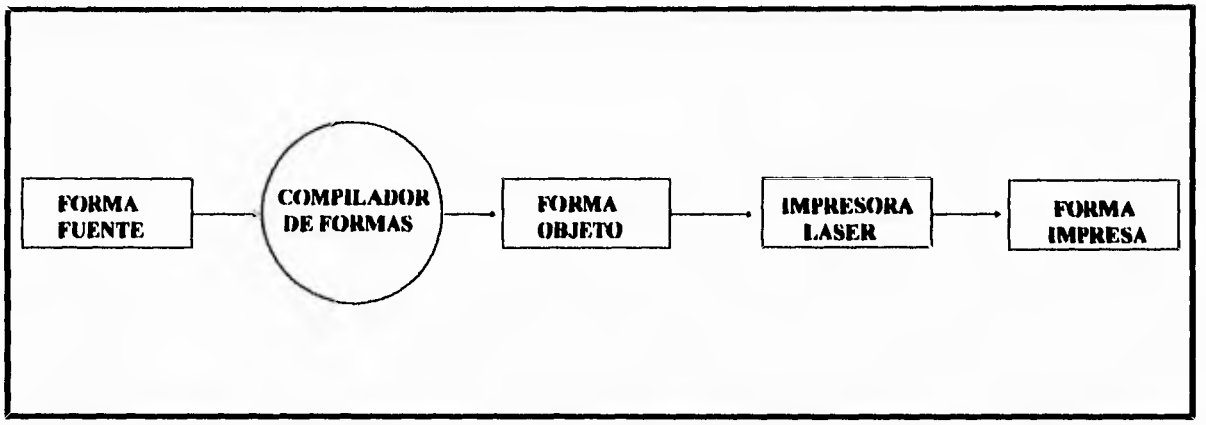


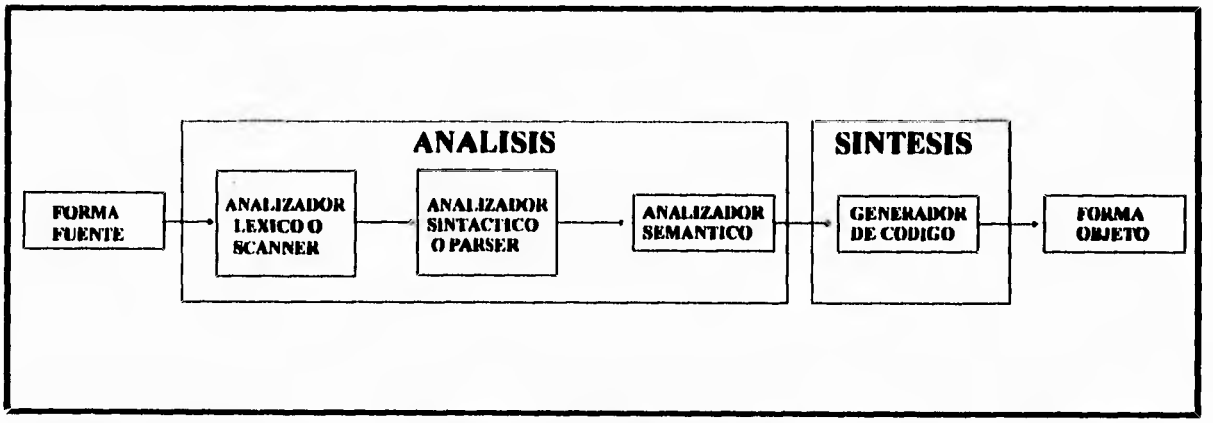
Figura 3.11 Diagrama de estructura para enviar mensaje

3.3 Diseño del compilador

Como se indicó en el capítulo anterior, el compilador de formas convertirá una forma fuente, escrita por el usuario, en una forma objeto. La forma fuente estará escrita en el lenguaje de descripción de formas y la forma objeto se escribirá en secuencias de escape, entendibles para una impresora láser Xerox (XES) o Hewlett Packard (PCL4).



El compilador de formas realizará principalmente dos tareas: el análisis de la forma fuente y la síntesis de su correspondiente forma objeto. La tarea de análisis estará dividida en tres partes: análisis léxico, análisis sintáctico y análisis semántico; mientras que la tarea de síntesis sólo consistirá en la generación de las secuencias de escape.



La forma fuente será leída por el analizador léxico o scanner como una cadena de caracteres para identificar y separar las partes básicas, llamadas tokens, que la componen. Un token puede ser una palabra reservada (p.e. FORMA, PESO, XES o END), un identificador, una constante numérica o algún caracter especial (p.e. ;).
Por razones de eficiencia, cada token será representado por un valor numérico único. En resumen, el scanner convertirá una forma fuente en una secuencia de tokens representados por su valor numérico.

A continuación se muestra la lista de tokens y su representación numérica que componen el lenguaje de descripción de formas.

Número Token	token
10	FORMA/FORM
20	IMPRESORAS/PRINTERS/IMP/PRI
30	ATRIBUTOS/ATTRIBUTES/ATR/ATT
40	INICIO/BEGIN/INI/BEG
50	FIN/END
60	FUENTES/FONTS
70	DESCRIPCION/DESCRIPTION/DESC
80	ORIENTACION/ORIENTATION/ORIE
90	TAMAÑO/SIZE/TAMA
100	MARGENES/MARGINS/MARG
110	CXLÍNEAS/CXLINE/CXL
120	LXPAGINA/LXPAGE/LXP
130	UNIDAD/UNIT/UNI
140	PESO/WEIGHT/WT
150	XES
160	PCL4
170	RES
180	RETRATO/PORTRAIT/VERTICAL/RET/POR/VER
190	PAISAJE/LANDSCAPE/HORIZONTAL/PAI/LAN/HOR
200	CARTA/LETTER
210	SUPERIOR/TOP/SUP
220	INFERIOR/BOTTOM/INF
230	DERECHO/RIGHT/DER/RGT
240	IZQUIERDO/LEFT/IZQ/LFT
250	RENGLON/ROW/REN
260	COLUMNA/COLUMN/COL
270	+
280	-
290	LÍNEAS_X/LINE_X
300	LÍNEAS_Y/LINE_Y
310	REPITE/REPEAT/REP
320	CADA/EACH
330	CAJA/BOX
340	ANCHO/WIDTH/WD
350	ALTO/DEPTH
360	DEBAJO/DOWN
370	DERECHA/AWAY

Número Token	token
380	INCLUYE/INCLUDE
390	FT
400	CENTIMETROS/CENTIMETERS/CM
410	PULGADAS/INCHES/IN
420	PUNTOS/DOTS
430	JUSTIFICA/JUSTIFY/JUST
440	CENTRO/CENTER/CEN
450	ARRIBA/UPPER/AR/UP
460	ABAJO/LOWER/AB/LO
470	MEDIO/MIDDLE/MD
480	;
490	"<cadena>"
500	identificador
510	constante_entera
520	constante_real

Con la lista anterior se puede observar que los tokens son en total 52, ya que los incrementos son de 10 en 10, con ello se procede a elaborar las sentencias y secciones del lenguaje, para determinar como sería la sintaxis que el compilador del lenguaje debe reconocer.

Los comentarios dentro de una forma fuente serán identificados e ignorados por el scanner; porque no se requieren en las siguientes fases de compilación, solo son útiles para el usuario. Por esta razón, los caracteres que se usan para indicar el inicio y terminación de comentarios (/*, */ y %) no forman parte de la lista de tokens.

Por ejemplo, la sentencia con el comentario

```
LÍNEAS_X 3 COL PESO 1;  % DIBUJA LÍNEAS EN HORIZONTAL
```

será convertida por el scanner en la siguiente secuencia de tokens, representados por su valor numérico:

290
510
260
140
510
480

El analizador sintáctico o parser recibirá la secuencia de tokens generada por el scanner, para identificar y revisar que estén correctamente escritas todas las proposiciones y sentencias que estén contenidas en la forma fuente. El análisis sintáctico estará basado en las estructuras sintácticas que definen el lenguaje de descripción de formas.

También por razones de eficiencia, las estructuras sintácticas del lenguaje de descripción de formas ahora se definen usando la representación numérica de todos sus tokens.

Sección FORMA

<nombre forma>:=

```
--- 10 -- 500 --- 480 ---#
```

Sección IMPRESORAS

<sección impresoras>:=

```
--- 20 ------ 150 ----- 480 ---#
          |-- 160 --|
```

<imp>:=

```
-------- 150 -------#
   |--- 160 ---|
```

Sección ATRIBUTOS

<sección atributos>:=

```
                  |----- < -----|
--- 30 --- 40 ---- <atributo> --- 50 --- 480 ---#
```

<atributo>:=

```
------- 80 ---- <orientación> ------- 480 ---#
   |--- 90 ---- <tam papel> -----|
   |--- 130 --- <unidad> --------|
   |--- 100 --- <márgenes> ------|
   |--- 110 --- 510 -------------|
   |--- 120 --- 510 -------------|
   |--- 140 --- 510 -------------|
```

<orientación>:=

```
------ 180 -------#
   |--- 190 ---|
```

<tam papel>:=

```
--- 200 ---#
```

<unidad>:=

```
------- 400 -------#
   |--- 410 ---|
   |--- 420 ---|
```

<márgenes>:=

```
          |------- < ---------|
--- 40 ----- <lista márgenes> --- 50 --- 480 ---#
```

<lista márgenes>:=

```
--------- 210 ------- 520 --- 480 ---#
                       |--- 220 ---|
                       |--- 230 ---|
                       |--- 240 ---|
```

<peso>:=

```
--- 140 --- 510 ---#
```

Sección FUENTES

<sección fuentes>:=

```
          |--------- < ----------------------------|
          |                      |----- < ----|      |
-- 60 --- <imp> --- 40 ---- <fuente> ----- 50 ------ 480 ---#
```

<imp>:=

```
-------- 150 -------#
   |--- 160 ---|
```

<fuente>:=

```
 --- 510 ------------------- 500 --- 480 ---#
          |--- 170 ---|
```

Sección DESCRIPCION

<sección descripción>:=

```
                  |------ < ------|
 --- 70 --- 40 --- <sentencia> ----- 50 --- 480 ---#
```

<sentencia>:=

```
-------------------------------#
   |--- <posiciona  línea> -----|
   |--- <posiciona  columna> ---|
   |--- <dibuja linea> ----------|
   |--- <dibuja cajas ------|
   |--- <texto> ---------|
```

<posiciona línea>:=

```
 --- 250 ------------------- 520 --- 480 ---#
                          |--- 270 ---|
                          |--- 280 ---|
```

<posiciona columna>:=

```
--- 260 ------------------- 520 --- 480 ---#
                          |--- 270 ---|
                          |--- 280 ---|
```

<dibuja línea>:=

```
------- <línea horizontal> ---- 480 ---#
   |-- <línea vertical> ---|
```

<línea horizontal>:=

```
--- 290 --- 520 ------------------------------>
                          |----- 260 ------|
                          |--- <unidad> ---|
   >=============================================================----#
          |--- <peso> ---|           |--- <repite línea> ---|
```

<línea vertical>:=

```
--- 300 --- 520 ------------------------------>
                          |---- 260 -------|
                          |--- <unidad> ---|
   >=============================================================-------#
          |--- <peso> ---|           |--- <repite línea> ---|
```

<repite línea>:=

```
--- 310 --- 510 --- 320 --- 510 ---------------------------#
                                |-- 250 -------|
                                |-- 260 -------|
                                |-- <unidad> --|
```

<dibuja caja>:=

```
--- 330 --- 340 --- 520 -------------------- 350 --- 520 ---------------------->
                    |---- 260 -----|                      |--- 250 ----|
                    |-- <unidad> --|                      |- <unidad> -|
   >-------------------------------------------------------------------- 480 ---#
     |-- <peso> --|    |-- <repite caja> --|      |-- <texto en caja> -|
```

<repite caja>:=

```
--- 310 --- 510 -------- 370 -------#
               |--- 360 ---|
```

<texto en caja>:=

```
--- 380 --- 390 --- 510 --- <justificación >--- 490 ---#
```

<justificación>:=

```
--- 430 ----- 230 ---------------- 450 --------#
          |-- 240 --|          |-- 460 --|
          |-- 440 --|          |-- 470 --|
```

<texto>:=

```
--- 390 --- 510 --- 490 --- 480 ---#
```

El analizador semántico también recibirá la secuencia de tokens generada por el scanner, pero validada por el parser, esto significa que la forma fuente no tiene errores de sintaxis. En esta fase de compilación, el análisis consiste en identificar y determinar el significado de cada proposición y sentencia para que sea convertida a secuencias de escape por el generador de código. Con estas secuencias de escape obtenidas se construirá finalmente la forma objeto en XES y/o PCL4, según se indica en la forma fuente.

El diagrama de estructura del compilador de formas se muestra en la figura 3.12. Los módulos valida licencia, valida comando y valida archivo pertenecen al conjunto de rutinas comunes del sistema, el diseño de estos módulos se trató en la sección anterior.

El diseño de los módulos restantes, scanner, parser, analizador semántico y generador de código, serán tratados por separado en esta misma sección.

Como primer paso, el módulo valida licencia será invocado por el compilador para checar la vigencia de la licencia de uso del producto de software, si la licencia aún es válida continuará con el proceso de compilación. El segundo paso será llamar el módulo valida comando, si el comando se escribió correctamente el proceso de compilación continuará. El módulo valida archivo será invocado como tercer paso para checar la existencia del archivo o forma fuente a compilar. Después de realizar las validaciones mencionadas, el compilador procederá a realizar el análisis léxico y sintáctico de la forma fuente invocando los módulos scanner y parser. Finalmente, si no se detectó ningún error durante las fases anteriores, el módulo analizador semántico y generador de código será llamado para crear la forma objeto. Este proceso de compilación se describe a continuación haciendo uso de pseudocódigo.

```
Valida licencia
Si licencia válida entonces
    Valida comando
    Si comando válido entonces
        Valida archivo
        Si archivo válido entonces
            Análisis léxico o scanning
            Si análisis válido entonces
                Análisis sintáctico o parsing
                Si análisis válido entonces
                    Análisis semántico y generación de código
Fin compilación
```

Este pseudocódigo representa la rutina principal del compilador de formas, ahora se procede a diseñar los módulos scanner, parser, analizador semántico y generador de código.

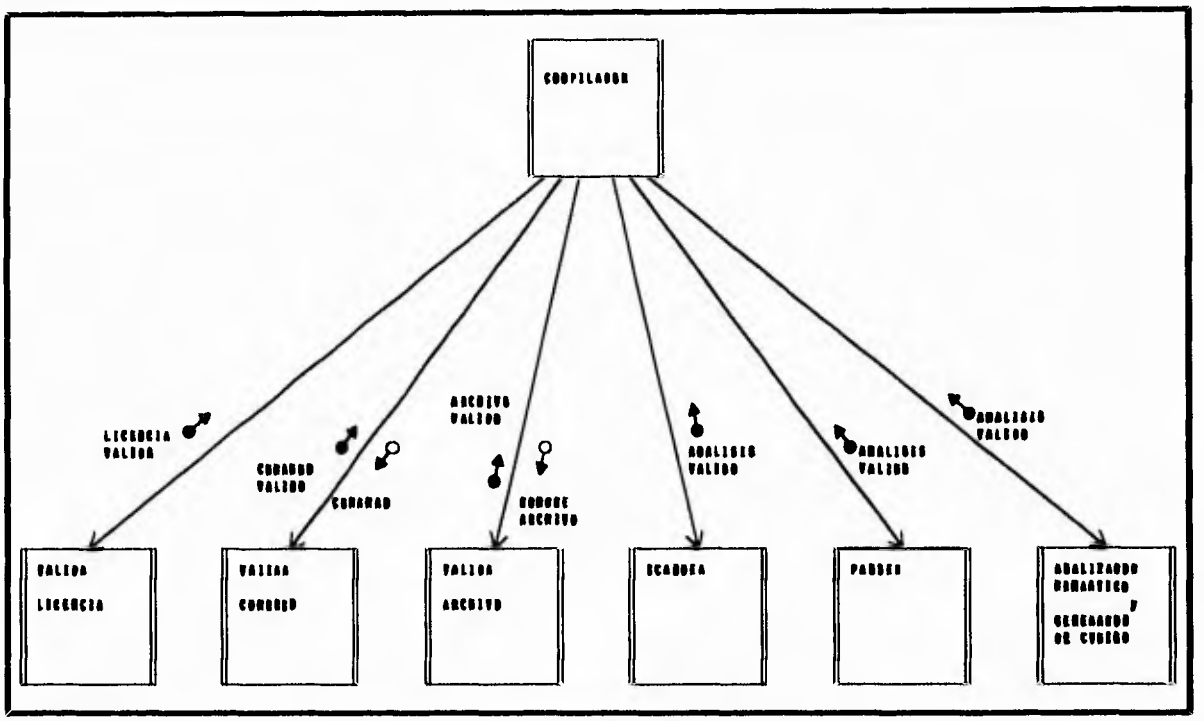


Fig. 3.12 Diagrama de estructura de compilador

3.3.1 Scanner

Como ya se mencionó, la principal función del scanner será convertir la forma fuente en una secuencia de tokens representados por su valor numérico. Debido a que el scanner debe leer la forma fuente como una cadena de caracteres, carácter por carácter, y reconocer o aceptar todos los tokens, su diseño se hará en base a un modelo

matemático llamado aceptor de estado finito o autómata finito. Los diagramas de estado finito o diagramas de transición se utilizan para representar gráficamente los aceptores de estado finito.

La figura 3.13 muestra el diagrama de estado finito para el lenguaje de descripción de formas, este diagrama se construye a partir de las estructuras sintácticas y de la lista de tokens del lenguaje.

Los nodos del diagrama representan los estados del autómata. Los arcos representan el estado de transición, el o los caracteres asociados a estos arcos indican la entrada que causa el estado de transición. En este diagrama el estado cero (0) representa el estado inicial y los estados que consisten de un par de círculos concéntricos representan los estados finales.

Los blancos o espacios serán tratados por el scanner únicamente como separadores de tokens, cada vez que se lea este carácter el scanner permanecerá o irá al estado inicial (0). Los estados 1, 2, 3 y 4 son usados para identificar comentarios. Si se lee la secuencia de caracteres /*, todos los caracteres siguientes serán considerados como parte del comentario hasta que se lea la secuencia */. De forma similar, si se lee el carácter %, todos los caracteres siguientes serán considerados como parte del comentario hasta que se detecte el final de la línea que se esta leyendo. Porque los comentarios no son necesarios en las siguientes fases de compilación, el scanner regresará al estado inicial inmediatamente después de terminar de leerlos.

El estado 5 es usado para reconocer palabras reservadas o identificadores, un identificador puede ser el nombre de la forma que se está compilando o el nombre de las fuentes. El primer carácter de un identificador siempre deberá ser una letra y los siguientes pueden ser letras, dígitos o los caracteres - y _. Para determinar si se trata de una palabra reservada o de un identificador, el scanner buscará el token reconocido en una tabla que contiene todas las palabras reservadas del lenguaje de descripción de formas, si el token no se encuentra en esta tabla entonces será considerado como un identificador.

Todos los identificadores reconocidos serán almacenados en una tabla que será usada en las siguientes fases de compilación.

Las constantes numéricas son reconocidas en los estados 6, 7 y 8, las constantes enteras en el estado 6 y las constantes reales en los tres estados. Debido a que una forma fuente puede contener un número indeterminado de constantes, el scanner registrará en una tabla cada constante que identifique. Esta tabla de constantes también será usada en las siguientes fases de compilación.

Los estados 9, 10 y 11 identifican los tokens +, - y ; respectivamente. Los estados 12 y 13 reconocen una cadena de caracteres que inicia y termina con el carácter ", estas cadenas son usadas en la sentencia FT (texto libre) del lenguaje de descripción de formas. De igual forma que los identificadores y las constantes, todas las cadenas de caracteres identificadas en los estados 12 y 13 serán insertadas en una tabla que será usada en las siguientes fases de compilación.

El arco etiquetado con 'error' indica que un carácter inválido ha sido encontrado en un lugar determinado de la forma fuente, algunos caracteres especiales (p.e. = ? . #) sólo pueden ser usados en comentarios o en cadenas de caracteres usadas en la sentencia FT.

Existe un conjunto de funciones comunes que el scanner debe realizar frecuentemente durante el análisis léxico de una forma fuente. La función GET_CHAR obtiene el siguiente carácter de entrada de la forma fuente. La función KEYWORD determina si el argumento (token) es palabra reservada, si lo es, regresa la representación

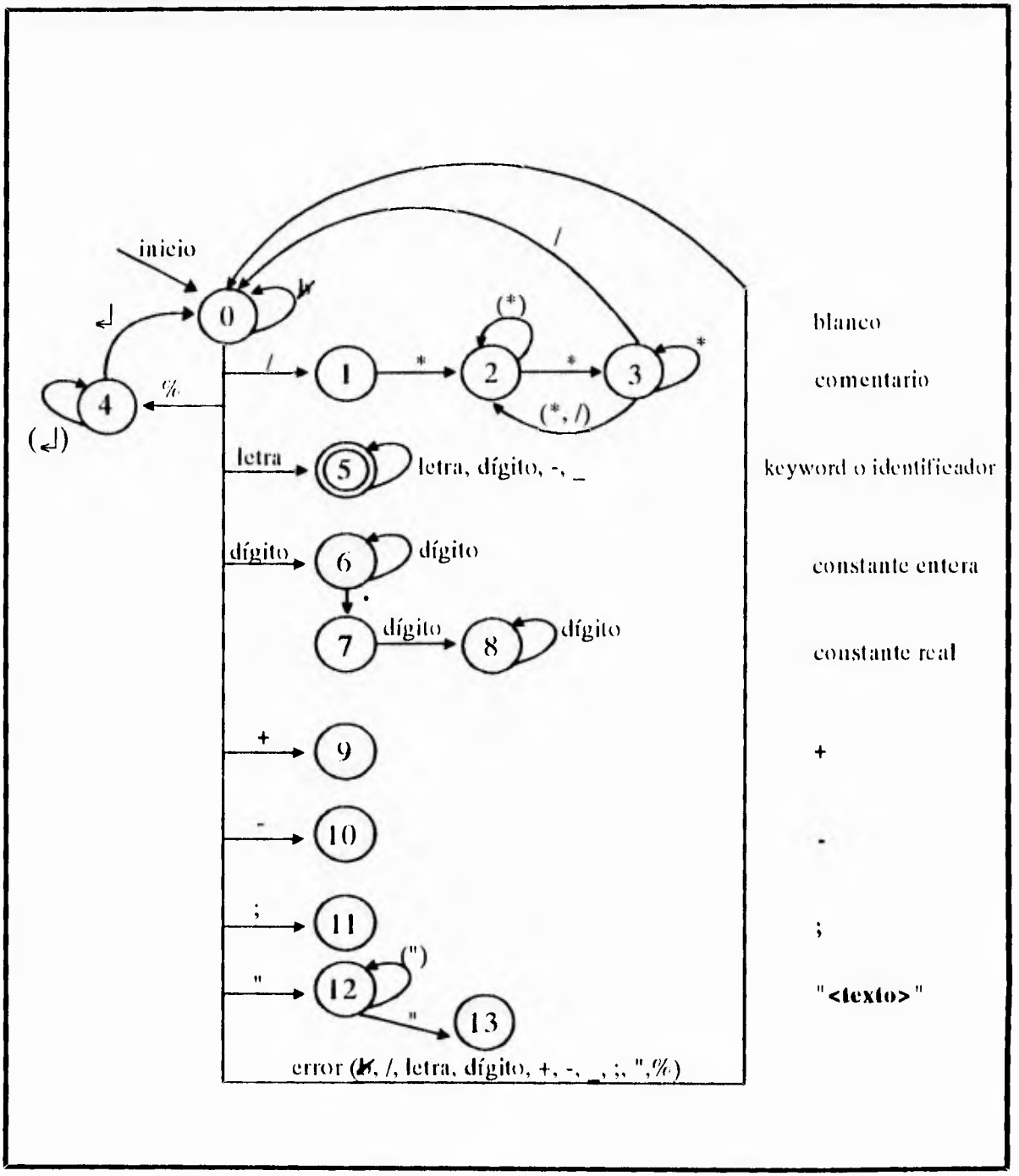


Fig. 3.13 Autómata del compilador

numérica del token, de lo contrario regresa un cero para indicar que se trata de un identificador. La función INS_TOKEN inserta en la tabla adecuada cada identificador, constante o texto libre identificado en la forma fuente y regresa la posición donde fue insertado el token. Finalmente, la función PUT_TOKEN escribe en la secuencia de salida la representación numérica de los tokens reconocidos y cuando aplica escribe también la posición del token dentro de la tabla donde se registro.

La principal función del scanner es identificar tokens dentro de una forma fuente, esta es una tarea repetitiva durante el proceso de análisis léxico.

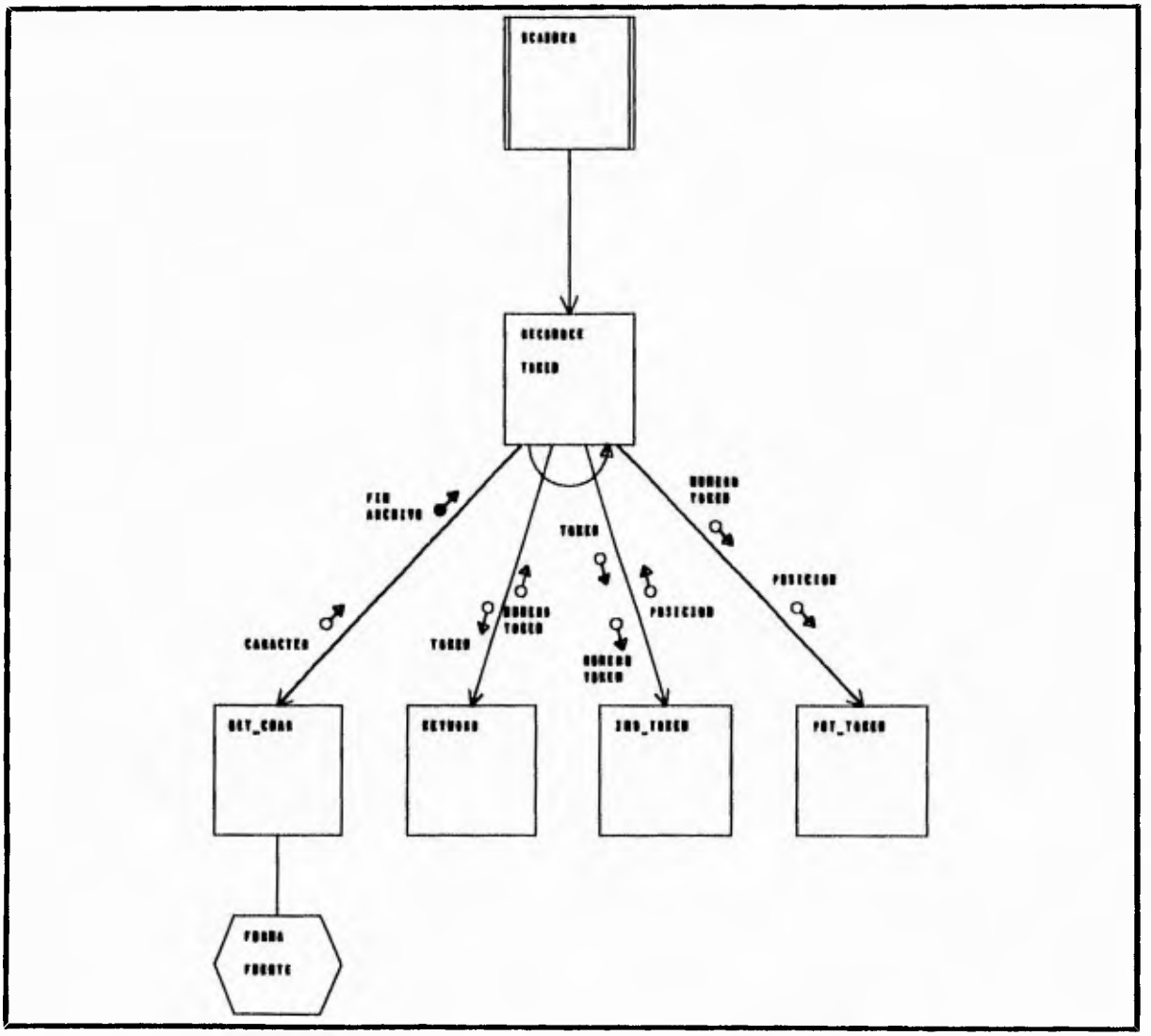


Fig. 3.14 Diagrama de estructura del scanner

pseudocódigo

La codificación del scanner se basa principalmente en el diagrama de estado finito para el lenguaje de descripción de formas (fig. 3.13) y en las funciones descritas en el diagrama de estructura (fig. 3.14). A continuación se muestra la implementación del scanner haciendo uso de pseudocódigo.

```
Inicializa variables
i_estado <- 0
i_token <- 0
i_posicion <- 0
c_caracter <- blanco
cd_token <- nulo
cd_texto <- nulo
Repite mientras no sea fin de archivo (forma fuente)
    Si i_estado = 0 entonces
        c_caracter <- GET_CHAR(forma fuente)
        Si c_caracter = blanco entonces
        i_estado <- 0
    Si c_caracter = / entonces
        i_estado <- 1
    Si c_caracter = % entonces
        i_estado <- 4
    Si c_caracter = letra entonces
        i_estado <- 5
        cd_token <- cd_token + c_caracter
    Si c_caracter = digito entonces
        i_estado <- 6
        cd_token <- cd_token + c_caracter
    Si c_caracter = + entonces
        i_estado <- 9
    Si c_caracter = - entonces
        i_estado <- 10
    Si c_caracter = ; entonces
        i_estado <- 11
    Si c_caracter = " entonces
        i_estado <- 12
    Si c_caracter no= blanco / % letra digito + - ; " entonces
        i_estado <- 0
        genera mensaje de error
    Si i_estado = 1 entonces
        c_caracter <- GET_CHAR(forma fuente)
        Si c_caracter = * entonces
            i_estado <- 2
        Si c_caracter no= * entonces
            i_estado <- 0
            genera mensaje de error
    Si i_estado = 2 entonces
        c_caracter <- GET_CHAR(forma fuente)
        Si c_caracter = * entonces
```

```
        i_estado <- 3
        Si c_caracter no= * entonces
            i_estado <- 2
    Si i_estado = 3 entonces
        c_caracter <- GET_CHAR(forma fuente)
        Si c_caracter = / entonces
            i_estado <- 0
        Si c_caracter = * entonces
            i_estado <- 3
        Si c_caracter no= / * entonces
            i_estado <- 2
    Si i_estado = 4 entonces
        c_caracter <- GET_CHAR(forma fuente)
        Si c_caracter = enter entonces
            i_estado <- 0
        Si c_caracter no= enter entonces
            i_estado <- 4
    Si i_estado = 5 entonces
        c_caracter <- GET_CHAR(forma fuente)
        Si c_caracter = letra digito - _ entonces
            i_estado <- 5
            cd_token <- cd_token + c_caracter
        Si c_caracter = blanco ; entonces
            i_estado <- 0
            i_token <- KEYWORD(cd_token)
            Si i_token no= 0 entonces
                PUT_TOKEN(i_token)
            Si i_token = 0 entonces
                i_token = 500
                i_posicion <- INSERT(cd_token,i_token)
                PUT_TOKEN(i_token,i_posicion)
                cd_token <- nulo
        Si c_caracter = ; entonces
            i_token <- 480
            PUT_TOKEN(i_token)
        Si c_caracter no= letra digito - _ blanco ; entonces
            i_estado <- 0
            cd_token <- nulo
            genera mensaje de error
    Si i_estado = 6 entonces
        c_caracter <- GET_CHAR(forma fuente)
        Si c_caracter = digito entonces
            i_estado <- 6
            cd_token <- cd_token + c_caracter
        Si c_caracter = blanco ; entonces
            i_estado <- 0
            i_token <- 510
            i_posicion <- INSERT(cd_token,i_token)
            PUT_TOKEN(i_token,i_posicion)
            cd_token <- nulo
```

```
        Si e_caracter = ; entonces
            i_token <- 480
            PUT_TOKEN(i_token)
        Si e_caracter = . entonces
            i_estado <- 7
            cd_token <- cd_token + e_caracter
        Si e_caracter no= digito blanco ; . entonces
            i_estado <- 0
            cd_token <- nulo
            genera mensaje de error
Si i_estado = 7 entonces
        e_caracter <- GET_CHAR(forma fuente)
        Si e_caracter = digito entonces
            i_estado <- 8
            cd_token <- cd_token + e_caracter
        Si e_caracter no= digito entonces
            i_estado <- 0
            cd_token <- nulo
            genera mensaje de error
Si i_estado = 8 entonces
        e_caracter <- GET_CHAR(forma fuente)
        Si e_caracter = digito entonces
            i_estado <- 8
            cd_token <- cd_token + e_caracter
        Si e_caracter = blanco ; entonces
            i_estado <- 0
            i_token <- 520
            i_posicion <- INSERT(cd_token,i_token)
            PUT_TOKEN(i_token,i_posicion)
            cd_token <- nulo
        Si e_caracter = ; entonces
            i_token <- 480
            PUT_TOKEN(i_token)
        Si e_caracter no= digito blanco ; entonces
            i_estado <- 0
            cd_token <- nulo
            genera mensaje de error
Si i_estado = 9 entonces
        i_estado <- 0
        i_token <- 270
        PUT_TOKEN(i_token)
Si i_estado = 10 entonces
        i_estado <- 0
        i_token <- 280
        PUT_TOKEN(i_token)
Si i_estado = 11 entonces
        i_estado <- 0
        i_token <- 480
        PUT_TOKEN(i_token)
Si i_estado = 12 entonces
```

```
            c_caracter <- GET_CHAR(forma fuente)
            Si c_caracter = '' entonces
                i_estado <- 13
            Si c_caracter no= '' entonces
                i_estado <- 12
                cd_texto <- cd_texto + c_caracter
        Si i_estado = 13 entonces
            i_estado <- 0
            i_token <- 490
            i_posicion <- INSERT(cd_texto,i_token)
            PUT_TOKEN(i_token,i_posicion)
            cd_texto <- nulo
    Fin repite
    Fin analisis lexico
```

Descripcion de variables:

i_estado-representa el estado en que se encuentra el autómata
c_caracter-contiene el último carácter leído de la forma fuente
cd_token-contiene la cadena de caracteres que forman un token
i_token-contiene la representación numérica del token
i_posición-contiene la posición del token en la tabla de identificadores, de constantes, o de textos libres
cd_texto- contiene la cadena de caracteres que forman un texto libre usado en la sentencia FT

3.3.2 Parser

La secuencia de tokens y la tabla de identificadores generadas por el scanner, serán leídas por el parser para realizar el análisis sintáctico de la forma fuente. Este análisis consiste en revisar los siguientes puntos:

* Que las cinco secciones de una forma fuente estén presentes y en el orden correcto.

- Que cada sección contenga las sentencias adecuadas.
- Que todas las sentencias estén escritas correctamente.
- Que los fonts a usar sean válidos.

La figura 3.15 muestra el diagrama de estructura para el parser

Los módulos Sección Forma, Sección Impresoras, Sección Atributos, Sección Fuentes y Sección Descripción serán invocados por el módulo principal (Parser), en el orden indicado, para validar que las secciones de la forma fuente estén completas y correctamente escritas.

El módulo repetitivo Reconoce Impresora, será invocado por el módulo Sección Impresoras para identificar y validar la(s) sentencia(s) que declara(n) la(s) impresora(s) a utilizar para imprimir la forma diseñada. El módulo Sección Atributos, invocará al módulo Reconoce Atributos para identificar y validar las sentencias que declaran los atributos de la forma diseñada. Al módulo Reconoce Fuente, será llamado por el módulo Sección Fuentes para identificar y validar las sentencias que declaran los fonts a utilizar en la impresión de la

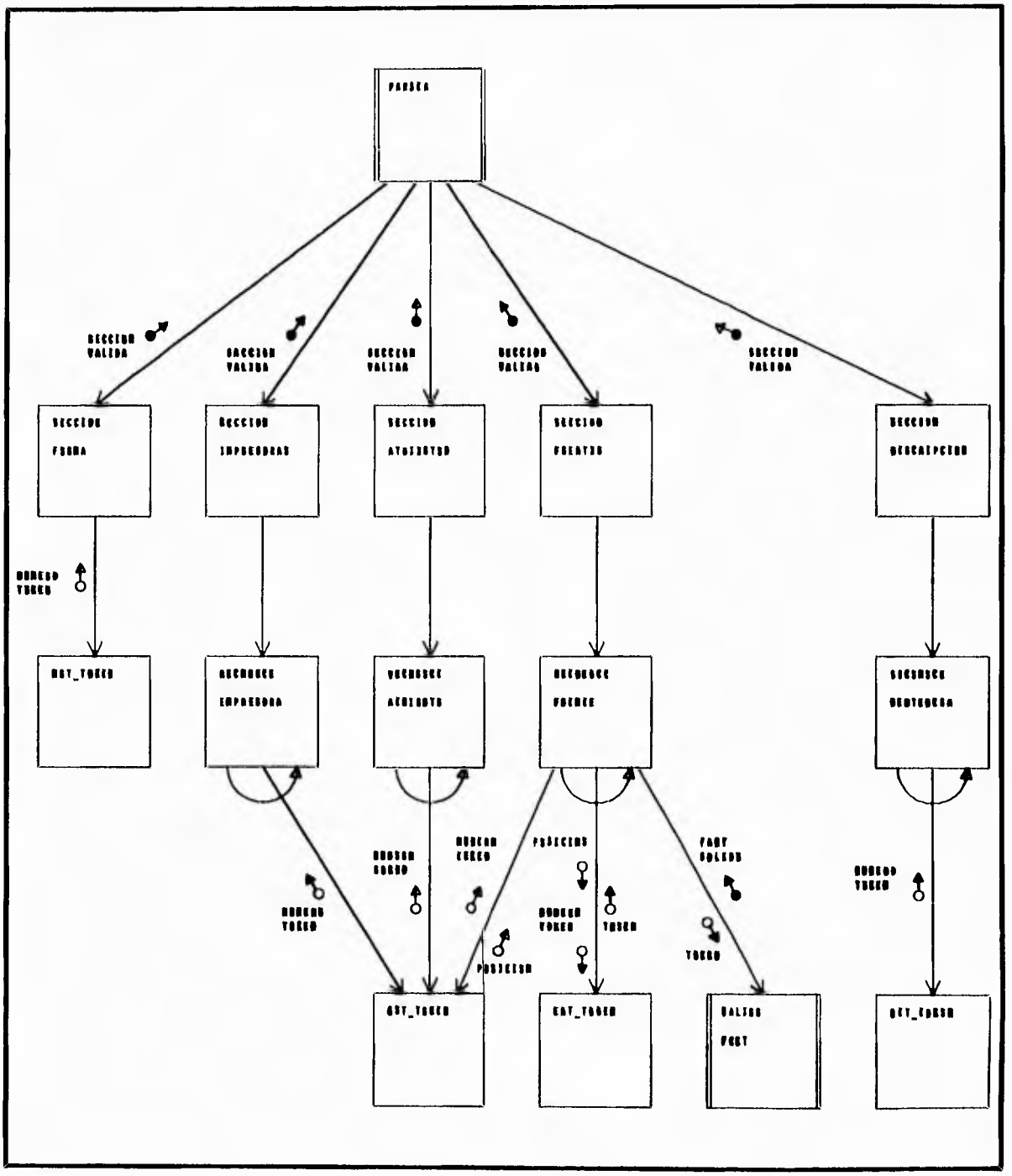


Fig. 3.15 Diagrama de estructura del parser

forma diseñada. Para identificar y validar las sentencias que pertenecen a la sección descripción, el módulo Reconoce Sentencia será invocado por el módulo Sección Descripción tantas veces como sentencias existan.

Con el fin de checar que los fonts declarados en la forma fuente sean válidos, que existan en la librería de fonts, el módulo Reconoce Fuentes invocará la función EXT_TOKEN para extraer de la tabla de identificadores, creada por el Scanner, el nombre del font a validar. Después de obtener el nombre del font, el módulo externo Valida Font será invocado para validarlo.

La función GET_TOKEN será invocada por los módulos Sección Forma, Reconoce Impresora, Reconoce Atributo, Reconoce Fuente y Reconoce Sentencia para obtener el siguiente token de la secuencia de tokens creada por el scanner.

Pseudocódigo:

```
Parser:
Analiza sección forma
Si sección válida entonces
  Analiza sección impresoras
  Si sección válida entonces
    Analiza sección atributos
    Si sección válida entonces
      Analiza sección fuentes
      Si sección válida entonces
        Analiza sección descripción
        Si sección válida entonces
          Continua con la compilación
        Si no interrumpe compilación
      Si no interrumpe compilación
    Si no interrumpe compilación
  Si no interrumpe compilación
Si no interrumpe compilación
Fin análisis sintáctico

Sección Forma:
i_token  GET_TOKEN(secuencia tokens)
Si i_token = 10 entonces
  i_token  GET_TOKEN(secuencia tokens)
  Si i_token = 500 entonces
    i_token  GET_TOKEN(secuencia tokens)
    Si i_token = 480 entonces
      l_seccion_valida  V
    Si no
      l_seccion_valida  F
  Si no
    l_seccion_valida  F
Si no
  l_seccion_valida  F
Fin análisis sección forma
```

```
Sección Impresoras:
i_token  GET_TOKEN(secuencia tokens)
Si i_token = 20 entonces
   i_token  GET_TOKEN(secuencia tokens)
   Repite hasta que i_token = 480
      Reconoce impresora
      Si la impresora es válida entonces
         l_seccion_valida  V
      Si no
         l_seccion_valida  F
      i_token  GET_TOKEN(secuencia tokens)
   Fin repite
Si no
   l_seccion_valida  F
Fin análisis sección impresoras

Sección Atributos:
i_token  GET_TOKEN(secuencia tokens)
Si i_token = 30 entonces
   i_token  GET_TOKEN(secuencia tokens)
   Si i_token = 40 entonces
      i_token  GET_TOKEN(secuencia tokens)
      Repite hasta que i_token = 50
         Reconoce atributo
         Si el atrbuto es válido entonces
            l_seccion_valida  V
         Si no
            l_seccion_valida  F
         i_token  GET_TOKEN(secuencia tokens)
      Fin repite
      i_token  GET_TOKEN(secuencia tokens)
      Si i_token no= 480 entonces
         l_seccion_valida  F
   Si no
      l_seccion_valida  F
Si no
   l_seccion_valida  F
Fin análisis sección atributos

Sección Fuentes:
i_token  GET_TOKEN(secuencia tokens)
Si i_token = 60 entonces
   i_token  GET_TOKEN(secuencia tokens]
   Repite hasta que i_token = 480
      Si i_token = 150 160 entonces
         i_token  GET_TOKEN(secuencia tokens)
         Si i_token = 40 entonces
            i_token  GET_TOKEN(secuencia tokens)
            Repite hasta que i_token = 50
               Reconoce fuente
```

```
                cd_token EXT_TOKEN(i_token,i_posicion)
                Valida_Font(cd_token)
                 Si el font es válido entonces
                   l_seccion_valida V
                 Si no
                   l_seccion_valida F
                 i_token GET_TOKEN(secuencia tokens)
              Fin repite
            Si no
              l_seccion_valida F
          Si no
            l_seccion_valida F
          i_token GET_TOKEN(secuencia tokens)
        Fin repite
      Si no
        l_seccion_valida F
      Fin análisis sección fuentes

      Sección Descripción:
      i_token GET_TOKEN(secuencia tokens)
      Si i_token = 70 entonces
        i_token GET_TOKEN(secuencia tokens)
        Si i_token = 40 entonces
          i_token GET_TOKEN(secuencia tokens)
          Repite hasta i_token = 50
            Reconoce sentencia
            Si la sentencia es válida entonces
              l_seccion_valida V
            Si no
              l_seccion_valida F
            i_token GET_TOKEN(secuencia tokens)
          Fin repite
          i_token GET_TOKEN(secuencia tokens)
          Si i_token no= 480 entonces
            l_seccion_valida F
        Si no
          l_seccion_valida F
      Si no
        l_seccion_valida F
      Fin análisis sección descripción
```

Descripción de variables:

i_token - contiene la representación numérica del token
cd_token - contiene la cadena de caracteres que forman un token
i_posicion - contiene la posición del token en la tabla de identificadores
l_seccion_valida - variable lógica que indica si la sección analizada es válida

3.3.3 Analizador semántico y generador de código

Estos procesos se ejecutarán de manera simultánea, ya que en el caso de nuestro producto de software, se generará código objeto para impresoras láser (XES o PCL4).

Se tomarán de la salida del parser la lista de tokens y las tablas de identificadores, constantes y cadenas. En primer lugar mencionaremos que el análisis semántico y la generación de código empezará cuando se detecte el inicio de la sección ATRIBUTOS, debido a que las secciones FORMA e IMPRESORAS sólo determinan el nombre del archivo de salida y el tipo de secuencias de escape que se generarán respectivamente.

Las secuencias de escape que se vayan generando durante el proceso estarán guardadas temporalmente en un arreglo en memoria, esto con el fin de acelerar la ejecución de la compilación.

Al iniciar los atributos, el analizador semántico guardará en una variable la orientación de la forma; ya que esto es determinante en el momento de pasar a la sección de fuentes. En esta parte, se debe generar una secuencia de escape única que contemple todos los atributos.

Como paso inmediato, se deben convertir las fuentes a secuencias de escape válidas, tomando de la tabla de identificadores el nombre de las mismas. Es importante aclarar que en el caso de que se traten de fuentes XES, habrá un proceso especial, para checar que la orientación de la fuente que se usará para los datos variables, sea la misma que se especificó en los atributos de la forma. En este punto se debe dejar indicado cuando deben cargarse las fuentes a la memoria de la impresora.

Una vez concluidos con éxito los pasos anteriores, se procederá a generar las secuencias de escape para las sentencias que integran el cuerpo de la forma. Como caso especial, cuando se encuentre el token que corresponda a la palabra reservada REPITE, el proceso deberá tener la capacidad de tomar los parámetros de esta palabra reservada y repetir las secuencias de escape de acuerdo a ellos.

Como paso final, se deberá crear un archivo en el dispositivo de almacenamiento, donde se vaciarán de la memoria principal todas las secuencias de escape generadas durante el transcurso del proceso.

Cuando se declare en la sección impresoras que se deben generar secuencias de escape tanto para XES como para PCL4, el proceso mencionado en párrafos anteriores se realizará dos veces.

El diagrama que muestra los pasos que se seguirán en el análisis semántico y la generación de código es el siguiente :

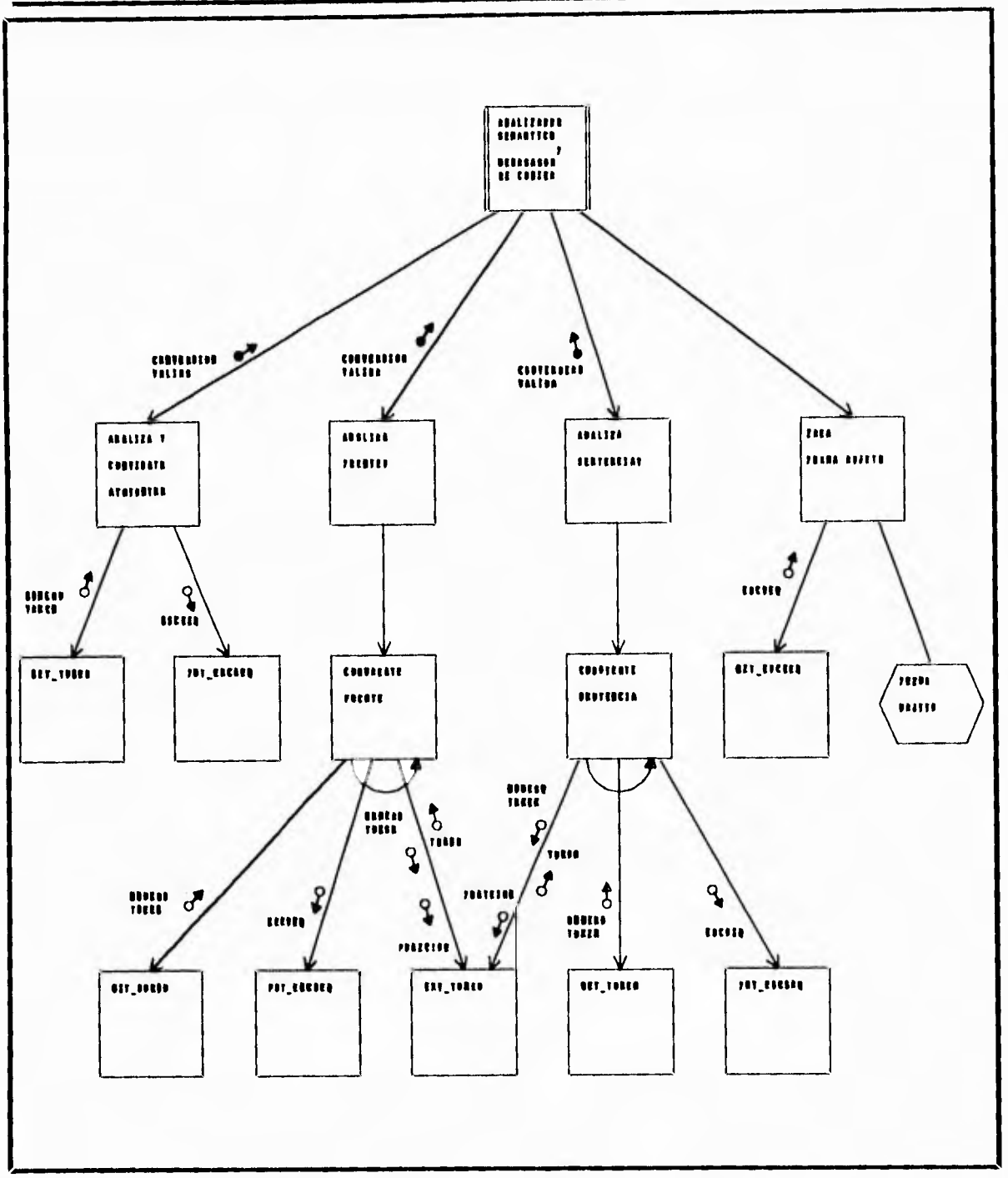


Fig. 3.16 Diagrama estructura de generador de código

PSEUDOCODIGO

```
i_numtoken = 0
i_posicion = 1
MIENTRAS i_numtoken  30
    GET TOKEN (Secuencia de tokens)

fin MIENTRAS

SEC_ATR(ed_sec,arr_sec)

SI !ERROR
    SEC_FNT(ed_sec,arr_sec)
SI no
    TERMINAR
fin SI

SI !ERROR
    SEC_SNT(ed_sec,arr_sec)
SI no
    TERMINAR
fin SI

SI !ERROR
    GEN_ARCH(arr_sec, ed_nomarch)
SI no
    TERMINAR
fin SI


TERMINAR

Funcion SEC_ATR (ed_sec,arr_sec)

l_bandera = falso


    MIENTRAS i_numtoken  50
        GET_TOKEN(secuencia de tokens)
        PARA
            i_numtoken =  80
                GET_TOKEN(secuencia de tokens)
                ed_orientacion = cadena_tabla ident (i_numtoken, i_postbl)

            i_numtoken =  90
                GET_TOKEN(secuencia de tokens)
                ed_tamaño = cadena_tabla ident (i_numtoken, i_postbl)
                i_puntosh = 8.5
                i_puntosv = 11
```

```
                    i_numtoken = 100
                    MIENTRAS i_numtoken  50
                        GET_TOKEN(secuencia de tokens)
                        PARA
                        i_numtoken  = 210
                    GET_TOKEN(secuencia de tokens
                    i_superior = valor_tabla_ctes (i_numtoken, i_postbl)

                        i_numtoken  = 220
                        GET_TOKEN(secuencia de tokens)
                        i_inferior = valor_tabla_ctes (i_numtoken, i_postbl)
                        i_numtoken  = 230
                    GET_TOKEN(secuencia de tokens)
                    i_derecho = valor_tabla_ctes (i_numtoken, i_postbl)
                        i_numtoken  = 240
                        GET_TOKEN(secuencia de tokens)
                        i_izquierdo = valor_tabla_ctes (i_numtoken, i_postbl)
                        fin PARA

                    fin MIENTRAS
                    i_numtoken = 0

                i_numtoken = 110
                    GET_TOKEN(secuencia de tokens)
                    i_exl = valor_tabla_ctes (i_numtoken, i_postbl)

                i_numtoken = 120
                    GET_TOKEN(secuencia de tokens)
                    i_lxp = valor_tabla_ctes (i_numtoken, i_postbl)
                 i_numtoken = 130
                    GET_TOKEN(secuencia de tokens)
                    cd_unidad = cadena_tabla ident (i_numtoken, i_postbl)

                i_numtoken = 140
                    GET_TOKEN(secuencia de tokens)
                    i_peso = valor_tabla_ctes (i_numtoken, i_postbl)
            fin PARA
            SI l_bandera = verdadero
                CREA_SEC (cd_sec)
            fin SI

        fin MIENTRAS
    PUT_ESCSEQ (arr_sec)

    Funcion SEC_FNT (cd_sec,arr_sec)
     MIENTRAS i_numtoken  50
        GET_TOKEN(secuencia de tokens)

            SI (i_numtoken = 150) ó (i_numtoken = 160)
                MIENTRAS i_numtoken  50
```

```
                GET_TOKEN(secuencia de tokens)
                PARA
                    i_numtoken = 510
                        i_numfont = valor_tabla_ctes (i_numtoken, i_postbl)
                        i_numtoken = 170
                        l_cargar= verdadero

                    i_numtoken = 500
                        cd_nomfont = valor_tablaident (i_numtoken, i_postbl)
                        cd_orie2 = obtener_ultima_letra (cd_nomfont)
                        SI cd_orie2  cd_orentacion
                                    mensaje_error (num_error)
                                    TERMINAR
                        fin SI

                fin PARA
                cd_sec = cd_nomfont + chr(i_numfont)
                CREA _SEC(cd_sec)
                PUT_ESCSEQ(arr_sec)
            fin MIENTRAS
        fin SI
    i_numtoken = 0
fin MIENTRAS

Funcion SEC_SNT

MIENTRAS i_numtoken  50
    GET_TOKEN(secuencia de tokens)
    PARA
        i_numtoken = 250
            GET_TOKEN(cadena de tokens)
            i_posx = valor_tabla_ctes (i_numtoken, i_postbl)
            PUT_SEC(Posiciona renglón)

        i_numtoken = 260
            i_posy = valor_tabla_ctes (i_numtoken, i_postbl)
            PUT_SEC(Posiciona columna)

        i_numtoken = 290
            Mientras i_numtoken  480
                GET TOKEN(cadena de tokens)
                SI i_numtoken = 310
                    GET_TOKEN(cadena de tokens)
                    i_veces =  valor_tabla_ctes (i_numtoken, i_postbl)
                    GET_TOKEN(cadena de tokens)
                    GET_TOKEN(cadena de tokens)
                    i_cada = valor_tabla_ctes (i_numtoken, i_postbl)
            FIN MIENTRAS
            PUT_SEC(dibuja_línea_x)
```

```
i_numtoken = 300
        Mientras i_numtoken  480
        GET TOKEN(cadena de tokens)
        SI i_numtoken = 310
            GET_TOKEN(cadena de tokens)
            i_veces =  valor_tabla_ctes (i_numtoken, i_postbl)
            GET_TOKEN(cadena de tokens)
            GET_TOKEN(cadena de tokens)
            i_cada = valor_tabla_ctes (i_numtoken, i_postbl)
    FIN MIENTRAS
    PUT_SEC(dibuja_linea_y)
i_numtoken = 330
        Mientras i_numtoken  480
        GET TOKEN(cadena de tokens)
        SI i_numtoken = 340
            GET_TOKEN(cadena de tokens)
            i_ancho =  valor_tabla_ctes (i_numtoken, i_postbl)
        FIN SI
        SI i_numtoken = 260
            GET_TOKEN(cadena de tokens)
            i_unidad =  valor_tabla_ctes (i_numtoken, i_postbl)
        FIN SI

        SI i_numtoken = 350
            GET_TOKEN(cadena de tokens)
            i_alto =  valor_tabla_ctes (i_numtoken, i_postbl)
        FIN SI

        SI i_numtoken = 250
            GET_TOKEN(cadena de tokens)
            i_reng =  valor_tabla_ctes (i_numtoken, i_postbl)
        FIN SI

        SI i_numtoken = 310
            GET_TOKEN(cadena de tokens)
            i_veces =  valor_tabla_ctes (i_numtoken, i_postbl)
            GET_TOKEN(cadena de tokens)
        FIN SI

        SI i_numtoken = 380
            GET_TOKEN(cadena de tokens)
            GET_TOKEN(cadena de tokens)
            i_numfte =  valor_tabla_ctes (i_numtoken, i_postbl)
            GET_TOKEN(cadena de tokens)
            GET_TOKEN(cadena de tokens)
            i_texto= valor_tablaident (i_numtoken, i_postbl)
        FIN SI

    FIN MIENTRAS
    PUT_SEC(dibuja_caja)
```

```
i_numtoken = 390
    GET_TOKEN(cadena de tokens)
    i_numfte = valor_tabla_ctes (i_numtoken, i_postbl)
    GET_TOKEN (cadena de tokens)
    cd_cadena = valor_tablaident (i_numtoken,i_postbl)
    PUT_SEC(texto_libre)
FIN PARA
```

3.4 Módulo de mantenimiento de impresoras

El diseño del módulo de mantenimiento de impresoras se establece mediante la siguiente carta de estructura, la cual muestra los módulos o funciones que la conforman, cada uno de estos describe la función que realizará. Los módulos o rutinas que se llaman de forma externa, serán descritos como cartas individuales, las cuales requerirán de un paso de parámetros adecuados para realizar su función correspondiente.

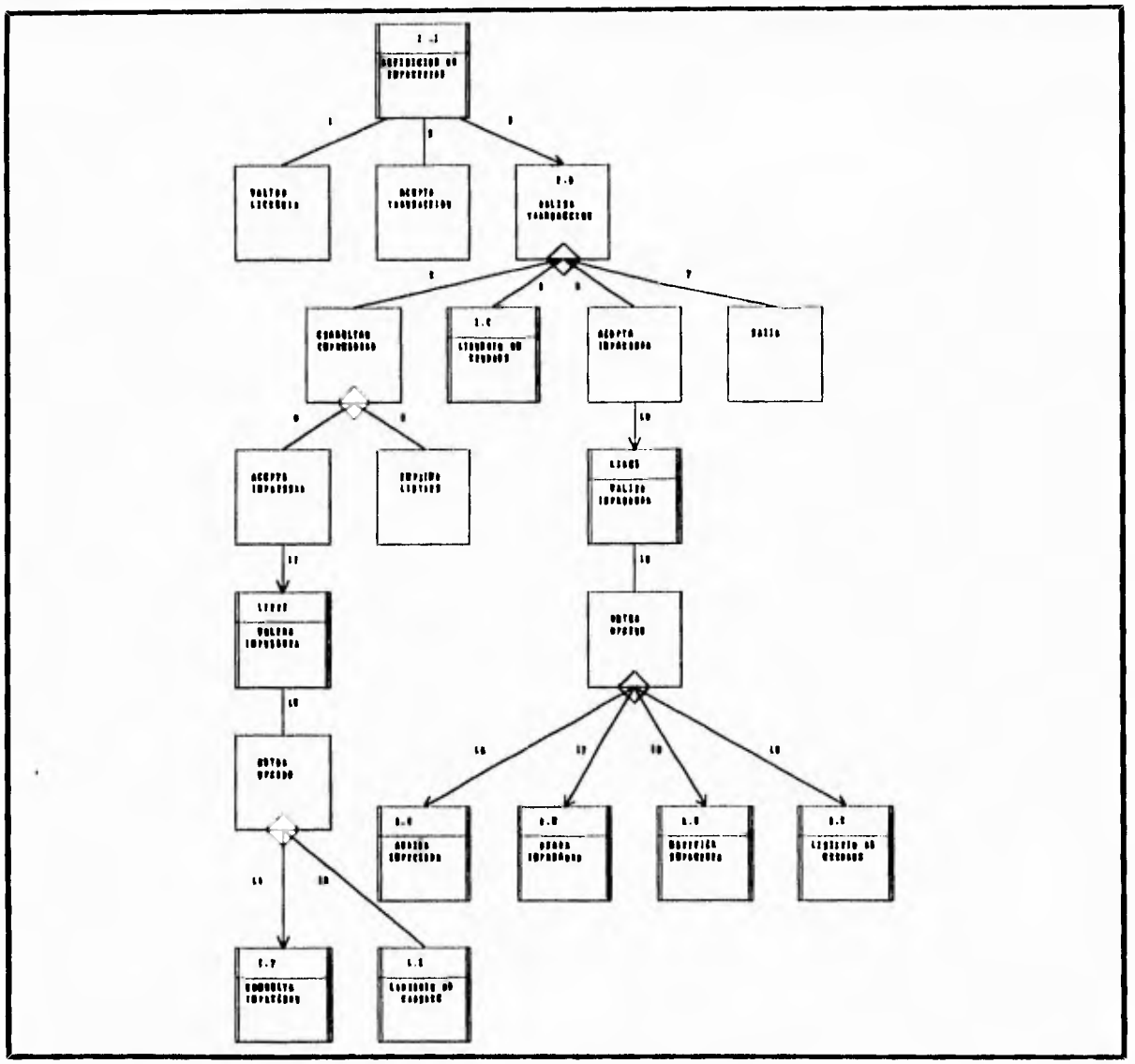


Fig. 3.17 Diagrama estructura definición impresoras

En terminos generales, el módulo de mantenimiento de impresoras deberá validar la vigencia del producto, a continuación tendrá que validar la transacción a realizar, esto se realizará llamando la rutina Valida_transacción, cuya función es la de desplegar la lista de opciones, validar el código de la opción que se ha seleccionado y regresar un valor en la variable llamada I_CODOPCION, en caso de que el código de la opción sea incorrecto, será llamada la librería de errores para que despliegue el mensaje correspondiente.

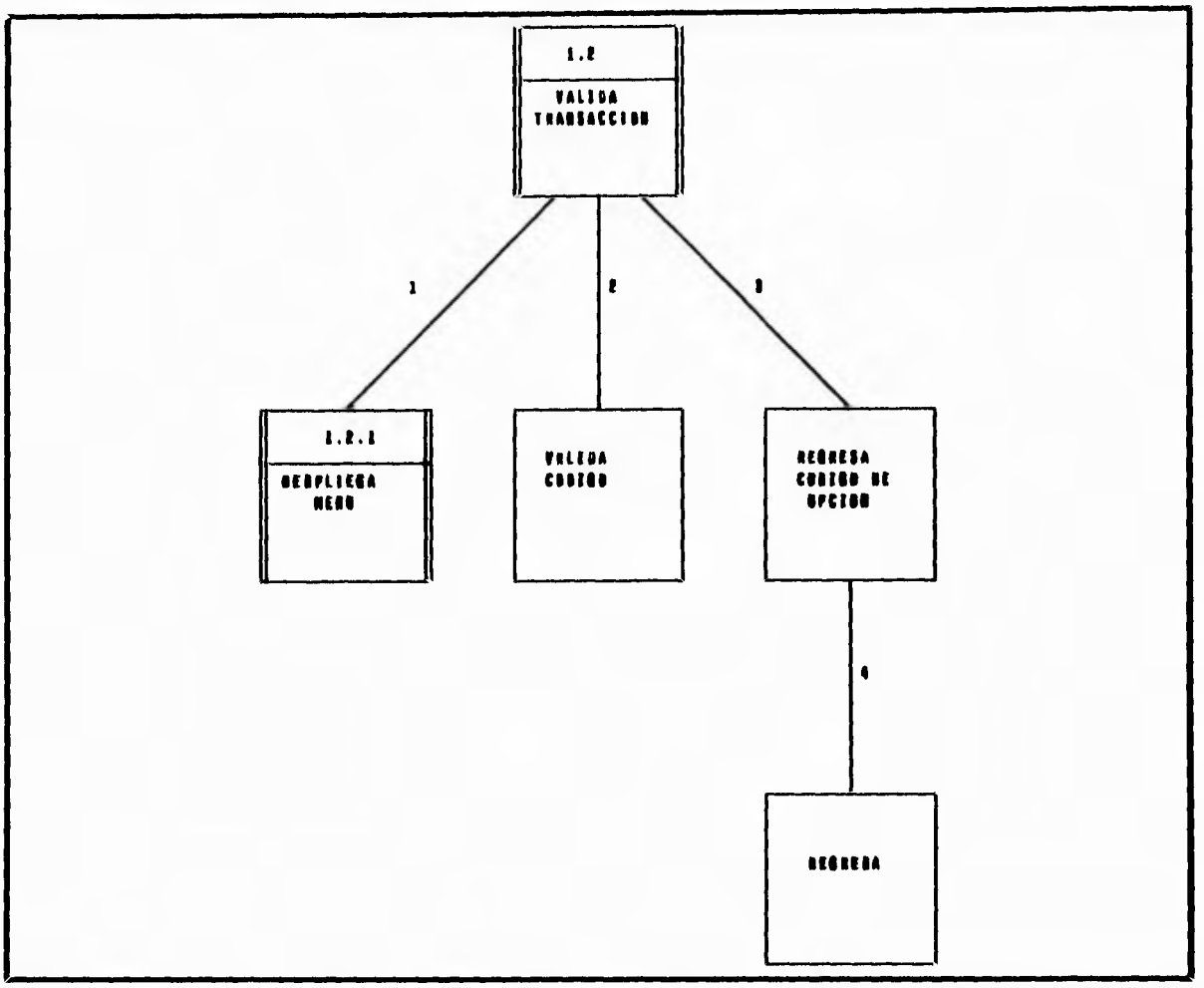


Fig. 3.18 Diagrama estructura valida transacción

Una vez validado el código de transacción, será necesario rutear hacia el módulo de consultas para imprimir listados de impresoras o para consultar sólo la definición de impresoras. Se tendrá un módulo para aceptar el nombre de la impresora, el cual llamará a la rutina de validación de impresoras, la cual a su vez nos devolverá una bandera de existencia o error de la misma; finalmente será ruteada hacia el módulo correspondiente, es decir, alta, baja o modificaciones.

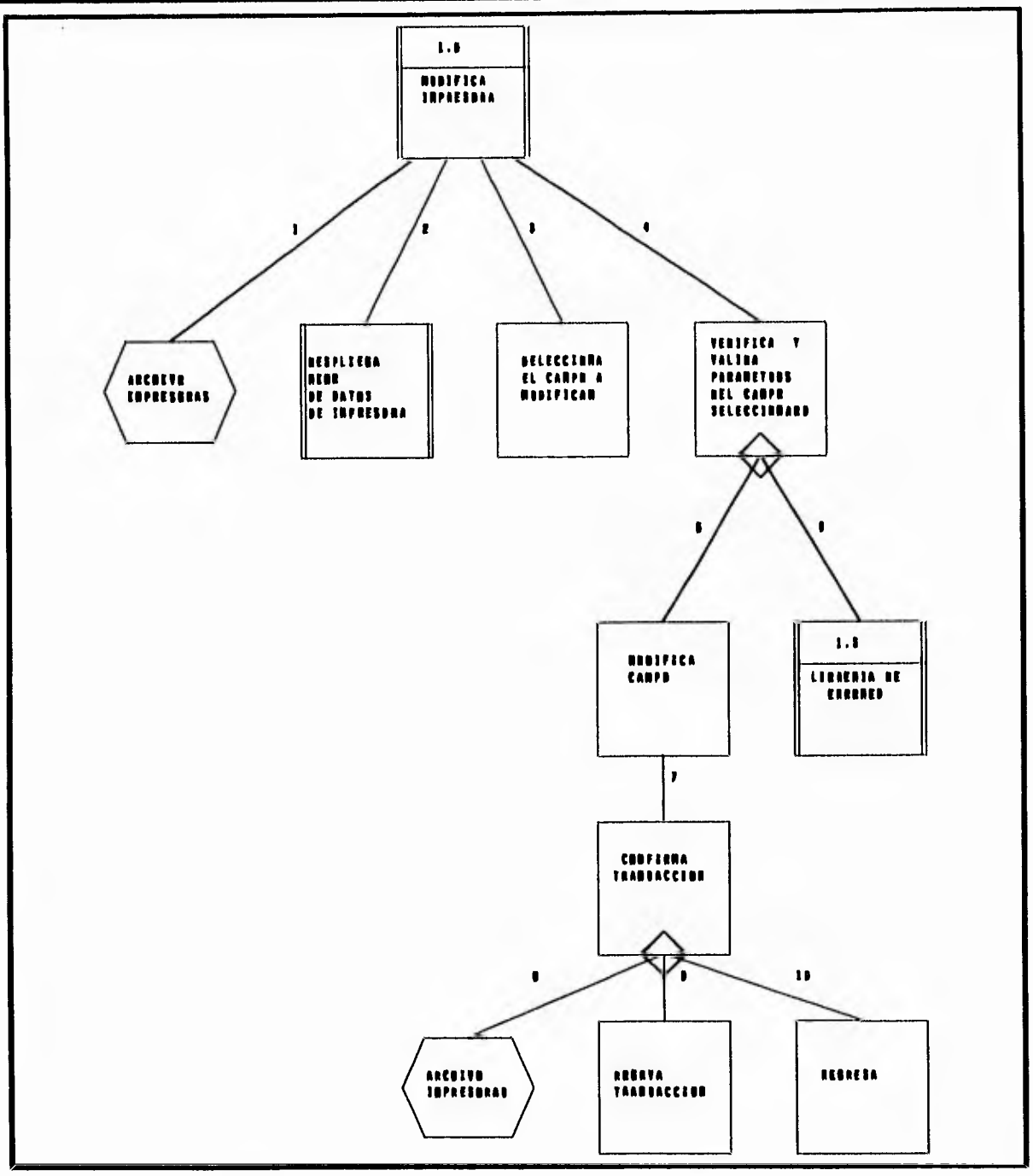


Fig. 3.19 Diagrama estructura modifica impresora

El pseudocódigo para este módulo en la rutina principal es como sigue:

```
inicio
Valida_licencia(l_lieval,l_nummen);
Acepta_transaccion(l_codope);
Valida_transaccion(l_codope,l_tranval);
Si transaccion valida
    inicio
        Si l_codope="consultar"
            inicio
                Consultar(l_codope);
                Si l_codope"listado"
                    inicio
                    Aceptar_impresor(CD_nomimp);
                        Llamar Lib03(CD_nomimp,l_valimp,l_nummen);(' Valida Impresora ')
                    Si impresora existe
                    Consultar(CD_nomimp);
                    Sino
                        Llamar Liberror(l_nummen); (' Llamar librería de errores ')
                    fin;
                Sino
                    Imprimir listado;
            fin;
        Si l_codope="salir"
            terminar;
        Sino
            inicio
                Aceptar_impresora(CD_nomimp);
                Llamar Lib03(CD_nomimp,l_valimp,l_nummen); (' Validar Impresora ')
                Si impresora no existe
                    Llamar Liberror(l_nummen); (' llamar libreria de errores')
                Sino
                    inicio
                        Si l_codope="Agregar"
                                Agregar(CD_nomimp);
                        Si l_codope="Borrar"
                                Borrar(CD_nomimp);
                        Si l_codope="Modificar"
                         Modificar(CD_nomimp);
                        fin;
            fin;
    fin;
Sino
    Llamar Liberror(l_nummmen); (' Llamar librería de Errores ')
fin;
```

3.5 Módulo utilería

El módulo de utilería de fonts reúne todos los elementos para realizar el proceso de descarga de fonts, la obtención de listados de fonts y de formas, así como la actualización de éstos en las respectivas librerías.

Para el diseño de este módulo se pretende a través de la carta estructurada definir cada una de sus funciones; donde cada acción representada, será detallada con el uso de pseudocódigo. Posteriormente, con estos elementos se procederá al diseño completo de dicho módulo, apoyándose para ello en un lenguaje de alto nivel.

Se ha establecido una división de trabajos para realizar las funciones que deben hacerse con esta utilería, por ello se observan los siguientes diagramas:

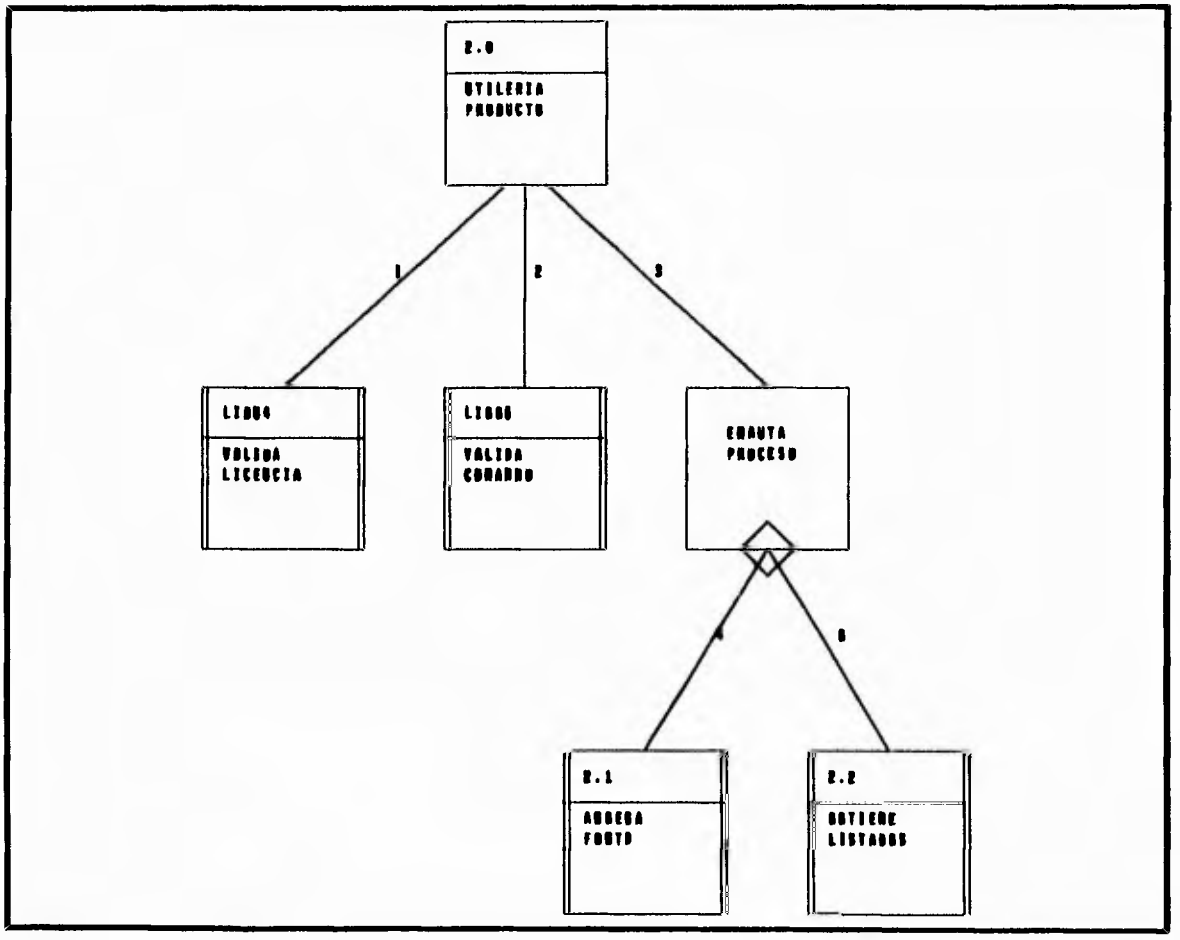


Fig. 3.20 Diagrama estructura utilería

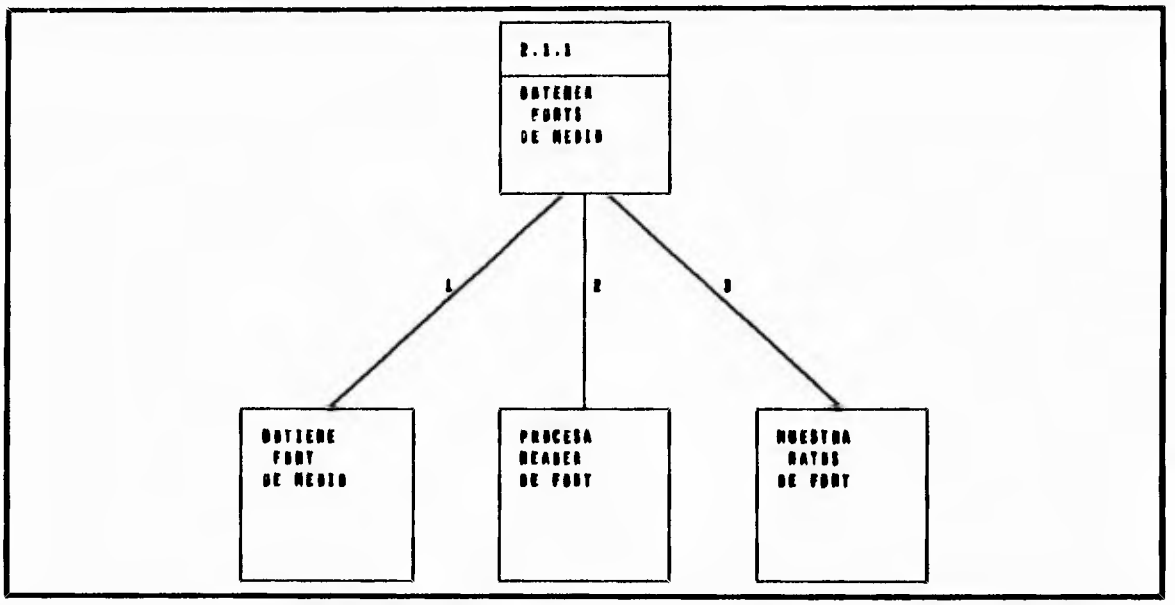


Fig. 3.21 Diagrama estructura obtener fonts de medio

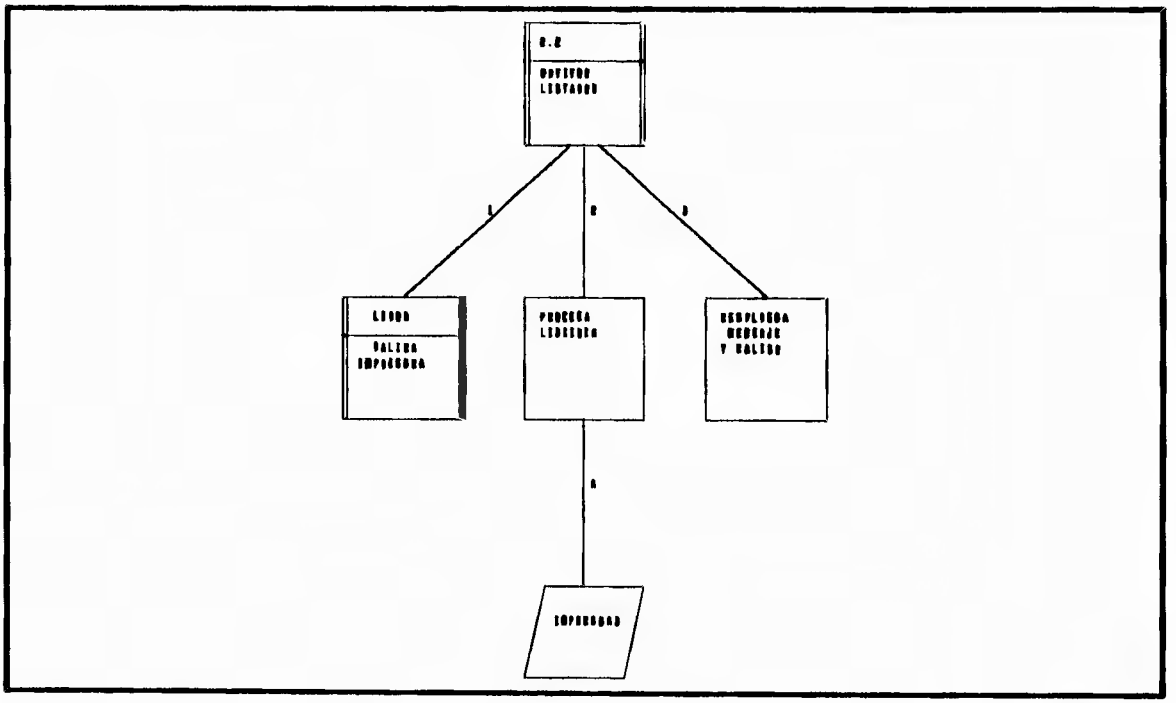


Fig. 3.22 Diagrama estructura obtener listados

A continuación se tiene el pseudocódigo de la rutina principal que constituye al módulo; con ella se valida la vigencia del producto y el tipo de transacción a realizar, haciendo uso de las rutinas de la biblioteca: Valida_Licencia y Valida_Comando.
Una vez que se verifica la licencia, el comando ruteará el proceso a seguir y que en este caso será el de agregar fonts u obtener listados; o bien llamará a la librería de errores para emitir el mensaje correspondiente.

/* Módulo principal de Utilería */

```
PROCESO PRINCIPAL Utilería_Fonts
DEFINE I_Transacción : ENTERO
INICIO
LLAMA FUNCION Valida_Licencia(I_LICVAL, I_DIFFEC, N_NUNMEN)
SI I_LICVAL es verdadera y ( I_DIFFEC es menor o igual a 30 y mayor que cero)
    DESPLIEGA Líneas_Comando
    LEE I_Transacción
    LLAMA FUNCION Valida_Comando
    SI I_Transacción := Utilería
        DESPLIEGA Línea_Comando
        LEE I_Transacción
        SI I_Transacción := Código_Agrega
            LLAMA PROCESO Agrega_Fonts
        SINO
            SI I_Transacción := Código_Listar
                LLAMA PROCESO Obtiene_Listados
            SINO
                DESPLIEGA Error(I_NUNMEN)
            TERMINA
        TERMINA
    SINO
        DESPLIEGA Error(I_NUNMEN)
    TERMINA
SINO
    DESPLIEGA Error(I_NUNMEN)
TERMINA
TERMINA Utilería_Fonts.
```

/* Proceso Agrega_Fonts */

```
PROCESO AGREGA_FONTS
INICIA
MIENTRAS EXISTA Medio(ARC_FONTS)
    LEE ARC_FONTS de Medio
    CREA Reg_DatoFont y define identificador de acceso LLave_Font
    PROCESA Reg_DatoFont en HEADER
CONTINUA
MIENTRAS Reg_DatoFont EXISTA
    LLAMA FUNCION Valida_Font(CD_NOMFON, I_NUNMEN)
    SI LLave_Font es diferente de CD_NOMFON
        AGREGA Reg_DatoFont en ARC_LIBFONTS
        COPIA Reg_DatoFont en Disco_Impresora
        GENERA Código_Termina
        DESPLIEGA I_NUNMEN('Termina copia de fonts')
    SINO
        DESPLIEGA I_NUNMEN('Ese font ya existe en librería')
    TERMINA
 CONTINUA
TERMINA Agrega_Font.
```

/* Proceso Obtiene_Listados */

```
PROCESO OBTIENE_LISTADOS
INICIA
DESPLIEGA Línea_Comando
LEE Datos Tipo_Librería y Nombre_Impresora
LLAMA FUNCTION Valida_Impresora(CD_NOMIMP, I_NUMMEN)
SI Nombre_Impresora es igual a CD_NOMIMP
    PROCESA Tipo_Librería
    SI Tipo_Librería es igual a Lista_Fonts
        LEE de Librería_Fonts el Reg_Fonts
        ENVIA Reg_Fonts a Impresora(CD_NOMIMP)
        DESPLIEGA I_NUNMEN('Termina impresión de listado de Fonts')
        SALIR de Proceso
    SINO
        LEE de Librería_Formas el Reg_Formas
        ENVIA Reg_Formas a Impresora(CD_NOMIMP)
        DESPLIEGA I_NUNMEN('Termina impresión de forma')
    TERMINA
SINO
    DESPLIEGA I_NUNMEN('Impresora Inexistente')
TERMINA
RETORNA
TERMINA Obtiene_Listados.
```

3.6 Módulo interfase de impresión

Como se mencionó en el capítulo anterior, la interfase de impresión reúne todos los elementos del proceso de la impresión electrónica. En esta parte, se realizará la carta de estructura, así como el pseudocódigo de la interfase, con estos elementos se tendrá el diseño de este módulo para su posterior realización en el lenguaje de alto nivel

La carta de estructura estará constituida por varias funciones o módulos tal como se muestra en la siguiente figura, dichas funciones representan la realización de una acción determinada, de esta forma el proyecto tendrá involucrado el concepto de modularidad.

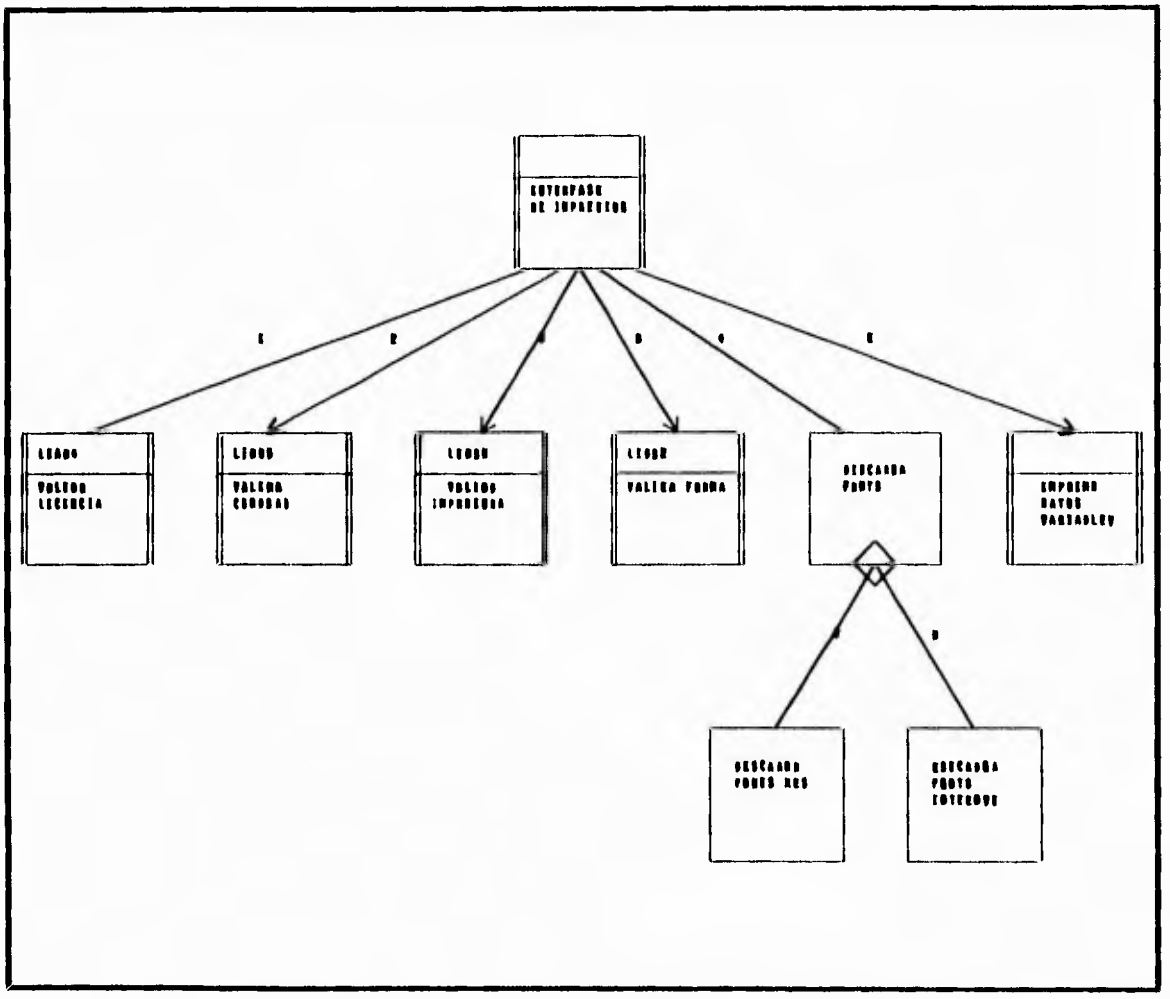


Fig. 3.23 Diagrama estructura interfase de impresión

El pseudocódigo de la carta de estructura para la interfase de impresión es el siguiente:

```
        IMPRE_FORM

PARAMETRO CD_LINENT


Valida Licencia (L_VALLIC, I_NUMMEN)
Si L_VALLIC entonces
    Valida comando (CD_LINENT, L_COMVAL, I_NUMMEN, CD_NUMARC, CD_NOMFOR,CD_NOMIMP, I_NUMCOP,
I_SALARC)
    Si L_COMVAL entonces
        Valida impresora (CD_NOMIMP, L_VALIMP, CD_EMVIMP, I_NUMMEN)
        Si L_VALIMP entonces
            I_NUMMOD=2
            Abre Archivo Libreria de Formas
            Valida Forma(I_NUMMOD, CD_NOMFOR, CD_EMVIMP, I_VALFOR,I_NUMMEN,  CDA_FONT [],
I_TOTLIN)
            Si L_VALFOR entonces
                Imprime Datos (CD_NOMFOR, I_NUMCOP, CD_NOMARC,CD_EMUIMP, I_TOTLIN,
CDA_FONT [])
                Cierra Archivo Libreria de Formas
            Si no
                Despliega error (I_NUMMEN)
                Cierra Archivo Libreria Formas
            Fin_Si
        Si no
            Despliega Error (I_NUMMEN)
        Fin_Si
    Si no
        Despliega Error (I_NUMMEN)
    Fin_Si
Si no
    Despliega Error(I_NUMMEN)
Fin si
```

En esta parte del pseudocódigo, primero se analiza que la licencia sea válida mediante una variable lógica, si es válida, se verifica que el comando sea el adecuado ya que mediante ello se obtendrán las diversas variables que serán útiles en las siguientes funciones del programa. Posteriormente se checa que la impresora solicitada exista dentro de la librería de impresoras, para verificar que dicho equipo tenga las características solicitadas por la forma que se va a enviar.

Al termino de las anteriores verificaciones podemos estar seguros que nuestra impresión va a ejecutarse adecuadamente, por lo que a continuación se muestra el pseudocódigo de la función de descarga de fonts.

```
FUNCION DESCARGA_FONTS

Parametros CDA_FONT [], CD_EMVIMP
Si CD_EMUIMP es igual a "XES" entonces
    I_APUFON = 1
    Mientras CD_FONT (I_APUFON) diferente de blancos
        Copia ARC_FONT AIMP(CDA_FONT (I_APUFON))
    Fin_Mientras
Si No
    Selecciona Fonts Residentes
Fin_Si
Regresa
```

Esta función realiza tanto la descarga de fonts como la asignación de font dentro de la impresora, dado que el producto puede operar en 2 tipos diferentes de impresión electrónica, la misma función realiza dicha actividad para cada uno de estos equipos independientemente.

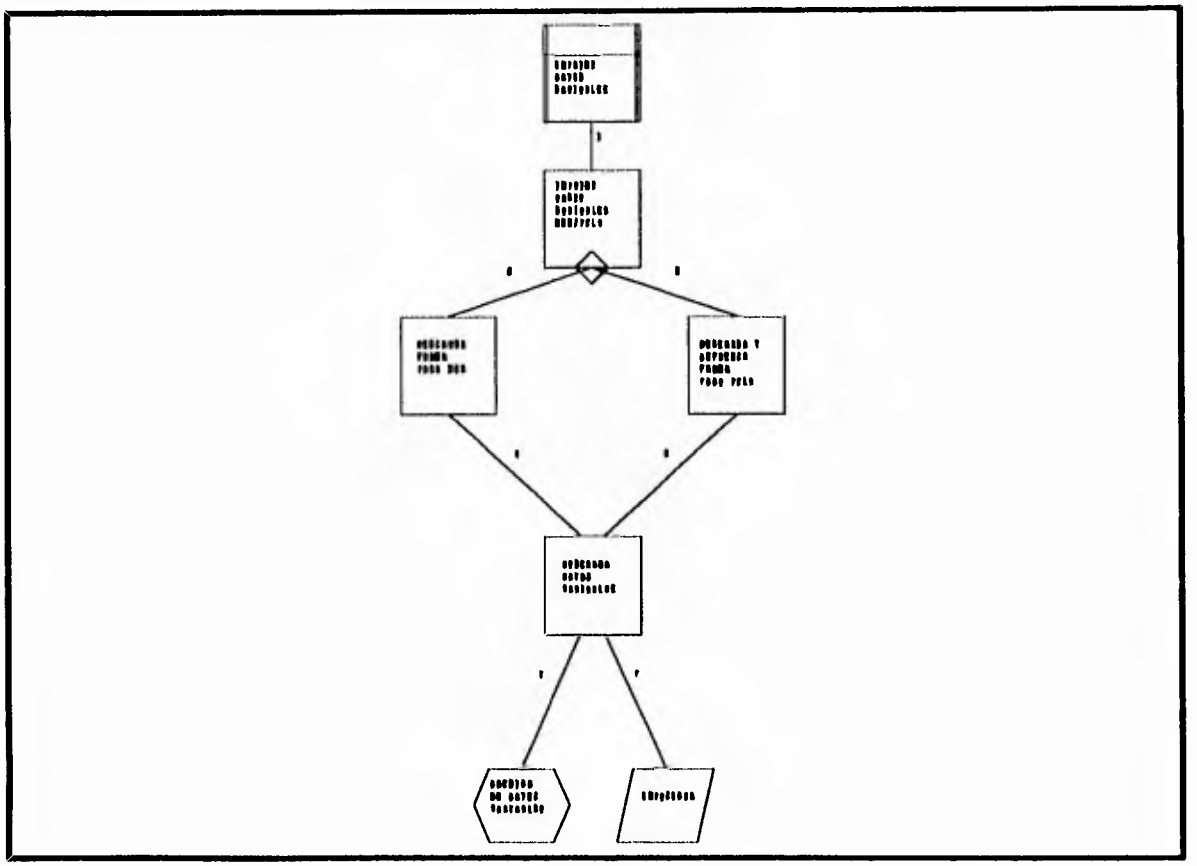


Fig. 3.24 Diagrama estructura imprime datos variables

Posteriormente se debe enviar la forma con los datos variables; el pseudocódigo de esta función se muestra a continuación.

FUNCION IMPRIME_DATOS

```
Parametros CD_NOMFOR, I_NUMCOP, CD_NOMARC, CD_EMUIMP, I_TOTLIN, L_SALARC, CDA_FONT ||
I_CONFOR = 0
I_NUMLIN = 0
Si CD_EMUIMP = "XES" entonces
    Descarga FONT (CDA_FONT ||, CD_EMUIMP)
    Copiar ARC_FORMA (CD_NOMFOR) a impresora
Si No
    Selecciona FONTS
Fin_Si
Si CD_NOMARC es blancos entonces
    Inicio
    Mientras I_CONFOR < I_NUMCOP
        Si CD_EMUIMP = "PCL4" entonces
            Copiar ARC_FORMA (CD_NOMFOR) a impresora
        Fin_Si
        Secuencia de escape para expulsar forma
        I_CONFOR = I_CONFOR + 1
    Fin_Mientras
    Fin
Si No
    Inicio
    EXISTE ARC_DATOS (CD_NOMARC, L_EXIARC)
    Si L_EXIARC
        Inicio
        Abrir ARC_DATOS (CD_NOMARC)
        Si Longitud ARC_DATOS = 0 entonces
            Despliega Error (#) ('No hay datos a imprimir')
        Si No
            Inicio
            Mientras I_CONFOR  = I_TOTLIN
                Inicio
                I_SALTA = Ultimo Caracter de R_ARCDATOS
                Si CD_EMUIMP = "PCL4" entonces
                    Copiar ARC_FORMA (CD_NOMFOR) a impresora
                Fin_Si
                Mientras no fin de R_ARCDATOS
                    Inicio
                    Si I_SALTA = Form feed o I_NUMLIN = I_TOTLIN ces
                        Inicio
                        Secuencia de Escape para expulsar hoja
                        Si CD_EMUIMP = "PCL4" entonces
                            Copiar ARC_FORMA (CD_NOMFOR) a impresora
                        Fin_Si
                        I_NUMLIN = 0
                        Fin
```

```
                        Fin_Si
                        Escribe R_ARCDATOS
                        I_NUMLIN = I_NUMLIN+1
                        Leer Siguiente Reg R_ARCDATOS
                        I_SALTA = Ultimo Caracter de R_ARCDATOS
                        Fin
                    Fin_Mientras
                    I_NUMLIN = 0
                    I_CONFOR = I_CONFOR+1
                    Fin
                Fin_Mientras
                Secuencia de Escape para expulsar hoja
                Fin
            Fin_Si
            Si L_SALARC es falso entonces
                Borrar archivo CD_NOMARC
            Fin_Si
            Fin
        Si no
            Despliega_Error(I?NUMMEN)
        Fin_Si
    Fin _Si
    Regresa
```

3.7 Instalación del producto

El proceso de instalación se realiza por única vez cuando necesitamos cargar el producto en el host en donde se llevará a cabo la operación del mismo. Como nota importante, se debe contemplar que el proceso de instalación es un programa independiente, esto es, no tiene funciones o procedimientos con todos los módulos que componen al producto de software

Este proceso se iniciará a través de un comando en el prompt del sistema operativo. El arranque de este proceso es a través de una pantalla de bienvenida, para posteriormente solicitar los siguientes datos iniciales.

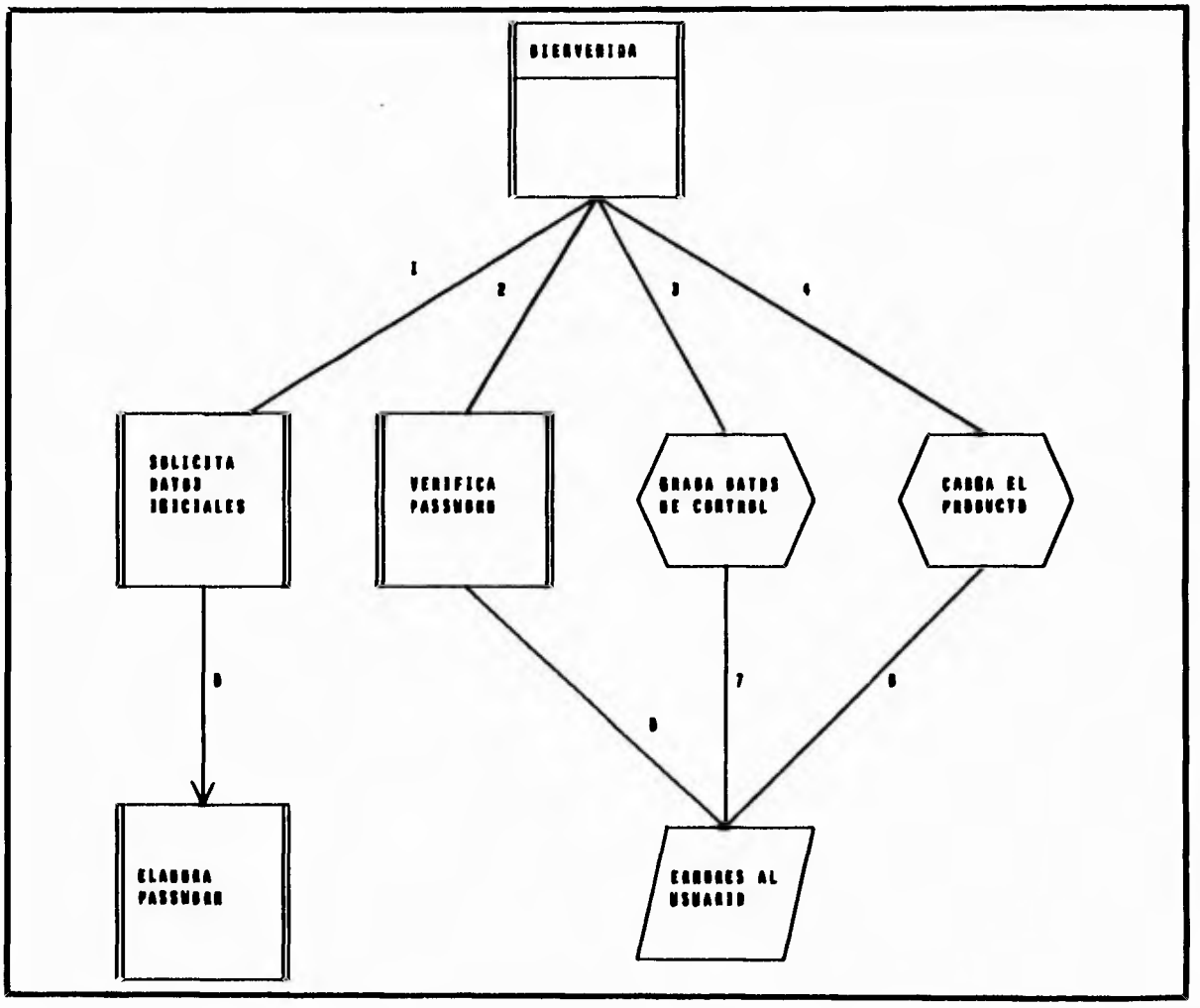


Fig. 3.25 Diagrama estructura instala producto

Dentro de los datos iniciales, es de suma importancia notar que se solicita la password que previamente el usuario deberá ingresar.

A través de estos datos iniciales, se procederá con el algoritmo que elaborará el password del producto, que debe coincidir con el que el usuario debió ingresar. En caso de no coincidir éstos, se enviará un mensaje alusivo y el programa terminará. Los datos de control se grabarán en el mismo lugar donde se guardarán los archivos de la aplicación, que se grabarán una vez que se concluya con los datos iniciales. Si se llevan a cabo con éxito los procesos anteriores, se concluye con el programa de instalación.

El pseudocódigo del módulo es el siguiente:

```
PUBLICAS CD_NOMEMP,CD_NUMSER
DESPLIEGA_BIENVENIDA
L_PASVAL = falso
SOLICITA CD_NOMEMP
SOLICITA CD_DIRPROD
SOLICITA CD_DIRARC
SOLICITA CD_LINST
SOLICITA CD_PASSW
D_FECINS = Fecha de hoy
CD_NUMSER = No. de serie HOST
D_VIGENCIA = D_FECINS
VALIDA_PASSWORD (CD_PASSWD,L_PASVAL,D_VIGENCIA)
SI L_PASVAL
   INICIO
    ESCRIBIR ARC_CONTROL
    SI ERROR_MEDIO
       INICIO
         DESPLIEGA MENSAJE(I_NUMERR)
         SALIR
       FIN
    ABRIR ARCHIVOS PRODUCTO
    SI ERROR_MEDIO
    INICIO
         DESPLIEGA MENSAJE(I_NUMERR)
         SALIR
       FIN
    ESCRIBIR ARCHIVOS PRODUCTO
    SI EROR_MEDIO
       INICIO
         DESPLIEGA MENSAJE(I_NUMERR)
         SALIR
       FIN
   FIN
SI NO
    DESPLIEGA MENSAJE(I_NUMERR)
SALIR
FUNCION VALIDA_PASSWORD
PARAMTEROS CD_PASSWD,L_PASVAL
```

```
I_PASS1= LONGITUD(CD_NOMEMP)
SI I_PASS1  50
     I_PASS = I_PASS + 50
CD_PASS1= ASCII(I_PASS1)
CD_INI1 = INICIALES (CD_NOMEMP)
I_INI = VALOR(CD_INI1)
CD_INI1 = HEXADECIMAL(I_INI)
I_NUMSER = VALOR(CD_NUMSER)
CD_NUMSER = ASCII(I_NUMSER)
CD_VIG1= CARACTERES 1,3,5 (D_VIGENCIA)
I_VIG1= VALOR (CD_VIG1)
CD_VIG1= ASCII(I_VIG1)
CD_PASS02 = CD_PASS1 + CD_INI1 + CD_VIG1
```

3.8 Diseño de pruebas de aceptación.

Para poder garantizar que un sistema de software esta libre de fallas o errores, es necesario definir un plan de pruebas de aceptación del sistema, cuyo objetivo principal, es el de aplicar una serie de pruebas para evaluar el sistema, dentro de cada uno de los criterios establecidos en la fase de definición de criterios de aceptación.

Esta serie de pruebas se llevarán a cabo durante la etapa de diseño e implementación, concluyendo con la etapa de pruebas finales del sistema, en la cual se mostrará un resumen de las fallas y correcciones detectadas durante el desarrollo del sistema.

3.8.1 Pruebas de funcionalidad.

Este tipo de pruebas son para garantizar que el sistema haga lo que debe hacer y no haga lo que no debe hacer.

Secuencias válidas de Altas, Bajas y Cambios.

Aplicado al módulo de mantenimiento de impresoras.

- Dar de alta 3 ó 4 impresoras con diferentes características.
- Consultar los datos de los registros para ver si se dieron de alta correctamente.
- Modificar el primero y último registro.
- Consultar para validar que los cambios se hayan realizado correctamente.
- Modificar registros intermedios.
- Consultar todos los registros para validar que las modificaciones se hayan realizado correctamente en cada uno de los casos.
- Eliminar el primero y último registro.
- Consultar el primero y último registro para verificar que se realizó la eliminación.
- Dar de alta registros previamente eliminados.
- Consultar que los registros se hayan dado de alta.
- Intentar dar de alta 2 ó 3 impresoras ya existentes
- Verificar que no sea permitida dicha acción y que sea enviado el mensaje correspondiente ante tal situación.
- Consultar los datos de las impresoras que se intentaron dar de alta para validar que no se alteró su información.
- Intentar dar de baja 2 ó 3 impresoras que previamente habían sido dadas de baja.

- Verificar que se despliegue un mensaje de que no existen esos registros.
- Consultar los registros para observar que no se ha efectuado ninguna irregularidad.
- Intentar realizar 2 ó 3 modificaciones sobre registros inexistentes o dados de baja.
- Verificar que los mensajes correspondientes sean desplegados.
- Consultar los registros para observar lo sucedido.

Ejecución de opciones validas en cada transacción.

Aplicado al módulo de compilador de formas.

- Escribir el comando con las opciones mínimas p.e. SIXCDF datos.FDL.
- Verificar que la transacción realice lo que está establecido para ese comando p.e verificar sintaxis y crear archivo con secuencias de escape.
- Escribir comando con opciones p.e. SIXCDF datos.FDL-l
- Verificar que la transacción realice lo que está establecido p.e. Compilar el archivo datos.fld, y generar un listado de la formas resultante.

Aplicado al módulo de utilería.

- Escribir el comando con las opciones mínimas p.e. SIXUTL add o
- Verificar que la transacción realice lo que está establecido para ese comando p.e. Agregar fonts externos a la librería de fonts.
- Escribir comando con el máximo de opciones p.e. SIXUTL -lfo impresora o SIXUTL -lfr impresora
- Verificar que la transacción realice lo que esta establecido p.e. Enviar un listado de fonts o formas existentes y dirigirlo hacia la impresora definida en el comando.

Aplicado al módulo de interfase de impresión.

- Escribir el comando con las opciones mínimas p.e. SIX -fr default -pr laserl
- Verificar que la transacción realice lo que está establecido para ese comando p.e. Enviar sólo un ejemplo de la forma definida como default e imprimirla en la impresora laserl
- Escribir comando con el máximo de opciones p.e. SIX -fl notas.prt -fr default -pr laserl -cp 2 -sv
- Verificar que la transacción realice lo que está establecido p.e. Mezclar el archivo de datos notas.prt con la forma default e imprimirlos en la impresora laserl con original y copia y almacenar el archivo de datos.

Seguridad de acceso y facultades operativas.

Aplicado al módulo de instalación del producto y a cada uno de los módulos que contemplan la validación de la licencia de uso.

- Intentos de instalación del producto con password válido y licencia de uso permitida.
- Verificar que la instalación se realice sin ningún problema.
- Intentos de instalación del producto con password no válido.
- Verificar que la instalación no proceda.
- Intentos de instalación del producto con password válido pero licencia de uso no permitida.
- Verificar que la instalación no proceda.
- Intento de acceso a cada uno de los módulos con licencia de uso vigente.
- Verificar que el acceso sea permitido sin ningún problema.
- Intento de acceso a cada uno de los módulos con licencia de uso a unos días de vencerse.
- Verificar que el acceso sea permitido sin problema pero desplegando un mensaje notificando reno vación de licencia de uso.
- Intento de acceso a cada uno de los módulos con licencia de uso vencida.
- Verificar que el acceso sea denegado.

Campos obligatorios no tecleados.

Aplicados a los módulos en donde esto aplique.

- Para cada campo obligatorio dejarlo en blanco.
- Validar que el sistema exija que esos datos sean suministrados.

Valores no válidos en campos.

Aplicados a los módulos en donde esto aplique.

- Dar 2 ó 3 datos por encima del rango permitido para cada campo en donde esto aplique.
- Dar 2 ó 3 datos por debajo del rango permitido para cada campo en donde esto aplique.
- Para campos numéricos dar 2 ó 3 datos alfabéticos.
- Para campos alfabéticos dar 2 ó 3 datos numéricos.
- Introducir caracteres especiales (#, %, , +, &) en los campos.

A cada una de los puntos anteriores verificar que el sistema responda de manera apropiada no permitiendo su entrada.

Detección de errores de operación.

Aplicado al módulo utilería (agregar fonts).

- Intentar agregar fonts desde diskette pero vacío o en otro formato
- Verificar que el módulo correspondiente detecte esta situación y sea desplegado el mensaje indicando tal irregularidad.

Aplicado al módulo utilería (listar formas o listar fonts).

- Solicitar el envío de listados hacia impresora no definida o inexistente.
- Verificar que sea solicitado un nombre de impresora válido o existente.

Aplicado al módulo mantenimiento de impresoras (listar impresoras).

- Solicitar el envío de listados hacia impresora no definida o inexistente.
- Verificar que sea solicitado un nombre de impresora válido o existente.

Aplicado al módulo Interfase de impresión (enviar forma ejemplo o datos y forma).

- Solicitar el envío de datos y forma o sólo forma hacia impresora no definida o inexistente.
- Verificar que sea solicitado un nombre de impresora válido o existente.

Página dejada en blanco intencionalmente

INSTRUMENTACION

La etapa de instrumentación de un producto de software, consiste básicamente en traducir todas las especificaciones de diseño en código fuente y realizar una documentación interna del producto; tanto el código fuente como la documentación deberán concordar con las especificaciones iniciales del sistema con la finalidad de verificar y facilitar la depuración, modificación y pruebas; para lograr esto es importante utilizar un modo de programación claro y sencillo, evitando lo menos posible utilizar técnicas de programación oscuras y complejas. El uso de técnicas de programación estructurada, buen estilo de programación, estándares, documentación y uso de comentarios nos dará como resultado un software de calidad.

Para este capítulo se incluirán parte de los listados de algunos de los programas principales que conforman el proyecto, así como las técnicas para la realización de los programas.

4.1 Estándares de programación en "C"

Los siguientes párrafos comprenden estándares de programación en lenguaje "C" que se implantaron para el proyecto, mismos que se siguieron durante la realización del mismo.

4.1.1 Nombre de archivos de programa

En primer lugar se establece el nombre de los archivos de programas los cuales para ambiente operativo MS-DOS, que es en donde se desarrolla en primer lugar el producto deberán ser máximo de 8 caracteres, el objetivo es tener portabilidad de los archivos en otras plataformas; así mismo, los nombres de los archivos deberán hacer referencia a lo que hace el programa a través de abreviaciones o nombres cortos.

Por otro lado la extensión de los archivos "Header" o de definiciones deberán componerse de una sola letra "h", mientras que los archivos de programa o fuente tendrán como extensión la letra "c".

A los nombres de los archivos se les antepondrán los caracteres "six" indicando "software de impresión para unix" complementándose con los siguientes 5 caracteres ya que estos harán referencia al propósito del archivo.

Ejemplos:

SIXBIB.C

SIXAPR.C

SIXCDF.C

SIXUTL.C

SIXKEYWD.H

Para obtener claridad en el código fuente se siguieron técnicas de programación estructurada para facilitar el mantenimiento del sistema a futuro.

En cuanto a las librerías tendrán como extensión "lib", los archivos de errores tendrán extensión "msg", el archivo de control incluirá la extensión "ctl", para los archivos que contienen la forma objeto tendrán como extensión "xes" para Xerox "pcl" para Hewlett Packard.

Ejemplos:

SIXFONT.LIB

SIXFORM.LIB

SIXMSG.MSG

SIXCTRL.CTL

4.1.2 Identación de código

La identación de código se realizará dejando 3 espacios en blanco y en el cuarto espacio comenzar a escribir, de la misma forma para los subbloques para darle una mayor legibilidad al código del programa; por ejemplo:

```
if (condición1)
{
   if (condición2)
        expresión;
}
```

Las instrucciones compuestas deben comenzar con la llave de apertura "{" en la siguiente línea del bucle o ciclo y alineada con la condición que le precede, mientras que la llave que cierra "}" deberá estar en la misma columna de la llave de apertura.

4.1.3 Compilación

Todo programa liberado a producción no deberá contener en su compilación "WARNINGS" de alerta al programador, ya que esto no cumpliría con la calidad que se espera tenga todo producto lanzado al mercado.

4.1.4 Ambiente de trabajo

En el sistema operativo UNIX la forma en que se organiza una aplicación se describe como sigue:

a) Se crea un subdirectorio en el directorio del usuario responsable de la aplicación, este debe de hacer referencia al nombre del proyecto.

b) En el subdirectorio de la aplicación se crean los siguientes 5 subdirectorios: bin, examples, include, lib, source.

c) En el subdirectorio bin se almacenan los archivos ejecutables de la aplicación.

d) En el subdirectorio examples se almacenan los programas o archivos de pruebas.

e) En el subdirectorio include se tienen los archivos de definición o "Header" (.h).

f) En el subdirectorio lib estarán las librerías que se generen para la aplicación.

g) Finalmente el subdirectorio source contendrá los archivos fuente (.c).

4.1.5 Archivos de definición o "Header"

Los archivos "header" en la sección de "includes" deben de ir entre los signos de menor que y mayor que < >. Ejemplo:

```
#include <nombre del archivo de definición >
```

4.1.6 Comentarios

Comentarios para encabezados de información

Los archivos "Header" y los archivos de implementación o archivos fuente constan de un encabezado de información como el siguiente:

```
/*****************************************************************************
 *  Sistema         :Software para Unix                                      *
 *                   Módulo de validación de licencia                        *
 *  Archivo         : sixbibl.c                                              *
 *  Propósito       : Rutina para validar la licencia de uso del producto.   *
 *  Autor           : Julio Bueyes Arteaga.                                  *
 *  Fecha creación  : Enero de 1994.                                         *
 *  Supervisión     : Julio Bueyes Arteaga.                                  *
 *  Historia        : Ultima  modificación 24 de Abril  1994.                *
 *****************************************************************************/
```

La primera línea del comentario debe iniciar con una diagonal (/) seguida de los asteriscos que sean necesarios.

En los siguientes renglones se comienza con un espacio en blanco, un asterisco (*), dos espacios en blanco y en el quinto espacio se escribe el comentario.

Los dos puntos (:) de separación entre el comentario y el texto de referencia deben ir alineados de acuerdo al texto más grande de los comentarios.

La última línea comienza con un espacio en blanco seguido de asteriscos (*) terminando con una diagonal(/).

Comentarios de documentación

Los comentarios que se hacen dentro del código se definen de dos formas:

a) Comentarios que requieren de más de una línea para documentar.

ejemplo:

```
/*
 Estructura para implementar la tabla de almacenamiento
 de tokens o palabras clave utilizado por el Scanner
*/
```

La primera línea debe comenzar con una diagonal (/) y un asterisco (*), los siguientes renglones se utilizan para describir el comentario alineado con el asterisco superior, finalmente la última línea comenzará con un asterisco (*) y una diagonal (/).

b) Comentarios que requieren una sola línea para documentar

ejemplo:

```
/* Estructura del registro del archivo de librería de Formas */
```

La línea comienza con la diagonal (/) seguida por el asterisco (*), un espacio en blanco, el comentario, un espacio en blanco y finalmente un asterisco (*) y una diagonal (/).

Para comentarios en las funciones auxiliares se deben documentar de la siguiente forma:

```
/*************************************************************
 *                                                           *
 * Función para validación de fonts: val_font(numero,font)   *
 *                                                           *
 *************************************************************/
```

La primera línea del comentario debe iniciar con la diagonal (/) seguida por los asteriscos (*) que sean necesarios.

En la segunda y cuarta línea se deja un espacio en blanco y se escribe un asterisco (*).

En la tercera línea se deja un espacio en blanco, un asterisco (*), un espacio en blanco y en el cuarto espacio se escribe el nombre de la función.

4.1.7 Nombres de identificadores

Este punto se refiere a la importancia de estandarizar los nombres de los identificadores en C, el objetivo es que en el código fuente se identifique inmediatamente el concepto que representa el identificador solo por la forma en que dicho identificador está escrito.

Los nombres de los identificadores del tipo que sean no deben exceder de 32 caracteres. Se deben emplear datos locales el mayor tiempo posible, emplear muchas variables globales es una mala técnica de programación, ya que dificulta la depuración y mantenimiento de los programas.

4.1.8 Variables globales y macros

Los nombres de las variables globales y macros (declaradas con #define) comenzaran siempre con una letra mayúscula y se construirán utilizando letras mayúsculas, dígitos, y/o underscores (_) para separar nombres compuestos; no deberá haber letras minúsculas en el nombre de una variable global o de una macro.

La gramática para la construcción de nombres de variables globales y macros es como sigue:

<mayúscula>= A| B | ... | Y | Z

<underscore>=_

<dígito>= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<variable_global_o_macro>=<mayúscula> {<mayúscula >| <underscore> |<d ígito>}$_{14}$

Ejemplo:

```
#define ARC_LIBFONT    "c:\\tesis\\source\\fontlib.lib"
#define LONCAD     10
#define TRUE       1
```

4.1.9 Variables locales y miembros de estructuras C

Para las variables locales y los miembros de estructuras C se construirán utilizando minúsculas, dígitos, y underscores para separar nombres compuestos.

La gramática para la construcción es la siguiente:

<mayúscula>= A| B | ... | Y | Z

<minúscula>= a | b | ... | y | z

<underscore>=_

<dígito>= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<variable_local>=<minúscula> {<minúscula >| <underscore> |<d ígito>}1+

Ejemplo:

```
char s_printer[LONCAD];       /* Nombre de impresora */
char s_nforma[LONCAD];        /* Nombre de forma
 int  I_save;             /* Salvar archivo
```

Un aspecto importante es que para todos los programas se siguieron estos estándares de programación, es decir que cada uno de ellos se tuvieron que apegar a las siguientes reglas en cuanto a la declaración de los nombres de las variables que se utilizaron en los programas.

Para nombres de variables de cadena de caracteres se antepusieron los símbolos "s_" y luego el nombre de la variable a utilizar

Para variables de tipo numérico los símbolos "i_" o "f_" para enteros o reales respectivamente.

Para los arreglos de enteros los símbolos "ia_".

Para constantes los símbolos "k_".

Para los arreglos de caracteres los símbolos "c_"

Y para las variables lógicas los símbolos "l_".

Además de lo anterior se procuró seguir las sugerencias propias del lenguaje "c" tales como nombres de variables con minúsculas y las constantes simbólicas con mayúsculas.

4.1.10 Funciones

Las funciones sean públicas o privadas deben de tener un prototipo, el cual consta del tipo de datos regresado por la función, el nombre de ésta y la lista de los tipos de los parámetros recibidos por la función.

La gramática para la construcción de los prototipos de funciones es el siguiente:

<tipo_dato>= cualquier tipo de dato válido en C, incluyendo "void" y tipos definidos por el usuario.

<prototipo>=<tipo_dato><nombre_func_c>(tipo_dato {, tipo_dato}0+);

ejemplos:

```
int ValidaForma(i_nummod,s_word,s_printer)
int i_nummod;
char s_word;
char s_printer;

ActualizaRegistro ( )
```

Los nombres de las funciones pueden contener letras y/o dígitos y deben comenzar siempre con una mayúscula. Los nombres de funciones no deben contener underscores(_). Si la función es un nombre compuesto, se pondrá en mayúscula el inicio de cada palabra.

La gramática de construcción de nombres de funciones es la siguiente:

<dígito>= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<nombre_func_e>={<mayúscula >{ <minúscula >| <dígito> }$_{1+}$}$_{1+}$

4.2 Características de implantación

Como se describe en el capítulo III el proyecto se dividió en varios módulos los cuales son:

- Compilador.
- Definición de impresoras.
- Utilería.
- Interfase de impresión .
- Instalación del producto.

Los detalles de cada uno de éstos se pueden revisar en el capítulo mencionado.

Una de las técnicas que se utilizaron durante la instrumentación fue la de enfocarse en las posibles rutinas que fueran comunes para varias partes del producto, lográndose con ello la construcción de una "biblioteca de funciones", la cual se convierte en un nuevo módulo dentro de los ya existentes, la finalidad de la construcción de este módulo es por la sencilla razón de que estas funciones son llamadas por el o los módulos que las necesiten, evitando con esto la codificación y repetición de líneas de código dentro de los programas.

Esta biblioteca de funciones comunes consta de las siguientes rutinas:

- Rutina para "validar licencia de uso", esta función es utilizada por todos los módulos del proyecto.
- Rutina para "validar impresoras" éste es utilizado por todos los módulos excepto por el "compilador".
- Rutina para "validar formas" utilizado por todos excepto por el modulo "utilería" y "definición de impresoras".
- Rutina para "validar fonts" es usado por el "compilador" y la "utilería".
- Rutina para "validar comandos" llamado por todos los módulos menos por el módulo de "definición de impresoras".
- Rutina para "validar un archivo", llamado por los modulos de compilador e Interfase de impresión
- Finalmente una rutina para "manejo de errores" utilizado por todos los módulos del proyecto.

4.2.1 Listado de la biblioteca de funciones comunes

```
/*********************************************************************
* SISTEMA          :  SOFIX, Software de Impresion para UNIX        *
*                  Rutinas de Validacion del Sistema                *
* Archivo          :  SIXBIB.C                                      *
* Proposito        :  Contiene rutinas de validacion del sistema    *
* Creada por       :  JULIO BUEYES ARTEAGA                          *
* Fecha Creacion   :  Enero 1994                                    *
* Supervision      :  JULIO BUEYES ARTEAGA                          *
* Historia         :  Ultima Modificacion ABRIL 1995                *
*********************************************************************/

/*********************************************************************
 * Funcion de Validacion de Licencia de Producto: val_lic(fecha)    *
 *********************************************************************/
/*                                                                   */
/*  Llamada:                                                         */
/*    numero = val_lic(fecha)                                        */
/*       donde:                                                      */
/*            fecha: es la fecha del dia tomada del sistema          */
/* La funcion regresara uno de los siguientes valores                */
/*       0   La licencia es invalida                                 */
/*       1   La licencia es valida                                   */
/*********************************************************************/

int val_lic(d_fechoy)
struct r_date d_fechoy;
{
      int i_diffec = CERO;
      FILE *f_lib;
      if ((f_lib = fopen(ARC_CONTROL,READ)) == NULL)
          {env_msg(0,20," Control File");
          exit(FALSE);}
      fread(&r_control,sizeof(r_control),UNO,f_lib);
      c_message=r_control.c_lenmen;
      fclose(f_lib);
      i_diffec = dif_fechas(r_control.d_feclic,d_fechoy);
      if (i_diffec <= CERO)
          {env_msg(0,1," ");
          return(FALSE);}
      if ((i_diffec <= TREINTA )&& (i_diffec > CERO ))
          {env_msg(0,2," ");
          return(TRUE); }
      else
          return(TRUE);
}

/*  Rutina que calcula la diferencia de fechas  */

int dif_fechas (d_fecha1, d_fecha2)
struct r_date d_fecha1;
struct r_date d_fecha2;
{
      int i_difdia, i_dia1, i_dia2, i_difanio;
      i_dia1 = dia_anio(d_fecha1);
      i_dia2 = dia_anio(d_fecha2);
```

```
        i_difanio = d_fecha1.i_anio - d_fecha2.i_anio;
        i_difdia=(i_dia1+(ANIO*i_difanio))-i_dia2;
        return(i_difdia);
}

/*  Rutina que obtiene los dias transcurridos del anio  */

int dia_anio(d_argfec)
struct r_date d_argfec;
{
        int i_counter,i_leap,i_dias;
        i_dias=d_argfec.i_dia;
        i_leap = d_argfec.i_anio % CUATRO == CERO && d_argfec.i_anio % CIEN != CERO
                || d_argfec.i_anio % CIEN4 == CERO;
        for (i_counter = UNO; i_counter  d_argfec.i_mes; i_counter++)
            i_dias += k_tabdias[i_leap][i_counter];
        return(i_dias);
}
```

```
/*************************************************************************
 * Funcion de Validacion de Existencia de Impresora: val_imp(s_word)     *
 *************************************************************************/
/*                                                                       */
/*  Llamada:                                                             */
/*    numero = val_imp(impresora)                                        */
/*  donde:                                                               */
/*       impresora:  nombre de la impresora a validar                    */
/* La funcion regresara los siguientes valores                           */
/*        0   La impresora es invalida                                   */
/*        1   La impresora es valida                                     */
/*       emulacion de la impresora encontrada si es valida               */
/*************************************************************************/

int val_imp(s_word)
char *s_word;
{
      int l_finval = FALSE;
      int l_existe = FALSE;
      int i_count;
      char s_impreg[11];
      FILE *f_lib;
      if ((f_lib = fopen(ARC_LIBIMPR,READ)) == NULL)
           {env_msg(0,20," Printers Library");
           exit(FALSE);}
      do {
           memset(s_impreg,'\0',11);
           fgetpos(f_lib,&i_posreg);
           fread(&r_printer,sizeof(r_printer),UNO,f_lib);
           if (feof(f_lib) != CERO)
                     {env_msg(0,3,s_word);
                     l_finval= TRUE;}
           else
                {
                strncpy(s_impreg,r_printer.s_nomimp,LONCAD);
                if (strcmp(s_word,s_impreg) == CERO)
                     {l_existe=TRUE;
                     l_finval=TRUE;}
                }
           }
       while (l_finval == FALSE );
      fclose(f_lib);
      return(l_existe);
}
```

```
/*************************************************************************
 * Funcion de Validacion de Existencia de Forma:                         *
 *          val_form(i_nummod,s_forma,s_emula)                           *
 *************************************************************************/
/*                                                                       */
/*  Llamada:                                                             */
/*    numero = val_form(modulo,forma,emulacion)                          */
/*  donde:                                                               */
/*          modulo  : numero de modulo llamador                          */
/*                  0  : modulo compilador                               */
/*                  1  : modulo impresion                                */
/*          forma:  es el nombre de la forma                             */
/*          emulacion: es la emulacion de impresora                      */
/* La funcion regresara uno de los siguientes valores                    */
/*       0   La forma es invalida                                        */
/*       1   La forma es valida                                          */
/*************************************************************************/

int val_form(i_nummod,s_forma,s_emula)
int  i_nummod;
char *s_forma;
char *s_emula;
{
      int l_finval = FALSE;
      int l_existe = FALSE;
      int i_count;
      char s_forreg[11];
      char s_emureg[5];
      FILE *f_lib;
      if ((f_lib = fopen(ARC_LIBFORM,READ)) == NULL)
          {env_msg(0,20," Forms Library");
          exit(CERO);}
      do {
          memset(s_forreg,'\0',11);
          memset(s_emureg,'\0',5);
          fgetpos(f_lib,&i_posreg);
          fread(&r_forma,sizeof(r_forma),UNO,f_lib);
          if (feof(f_lib) != CERO)
                {if(i_nummod == CERO)
                    env_msg(0,5,s_forma);
                else
                    env_msg(0,6,s_forma);
                l_finval= TRUE;}
          else
                {
                strncpy(s_forreg,r_forma.s_nomfor,LONCAD);
                strncpy(s_emureg,r_forma.s_impfor,CUATRO);
                if ((strcmp(s_forma,s_forreg) == CERO)
                    && (strcmp(s_emula,s_emureg) == CERO))
                    {l_finval=TRUE;
                    if (i_nummod == CERO)
                        env_msg(0,4,s_forma);
                    l_existe=TRUE;}
                }
          }
       while (l_finval == FALSE);
      fclose(f_lib);
      return(l_existe);
}
```

```
/*********************************************************************
 *  Funcion de validacion de existencia de font                      *
 *              val_font(i_nummod,s_nombre,s_estilo)                 *
 *********************************************************************/
/*                                                                   */
/*  Llamada                                                          */
/*  numero = val_font(numero,font,estilo)                            */
/*  donde:                                                           */
/*          numero: numero do modulo llamador                        */
/*                  0   : modulo utileria                            */
/*                  1   : modulo compilador                          */
/*          font :  es el nombre del font                            */
/*          estilo: es el tipo de impresora                          */
/*  La funcion regresara uno de los siguientes valores               */
/*      0   El font es invalido o no existe                          */
/*      1   El font es valido                                        */
/*********************************************************************/

int val_font(i_nummod,s_nombre,s_estilo)
int  i_nummod;
char *s_nombre;
char *s_estilo;
{
      int l_finval = FALSE;
      int l_existe = FALSE;
      char s_font[26];
      char s_tifon[5];
      FILE *f_lib;
      if ((f_lib = fopen(ARC_LIBFONT,READ)) == NULL)
          {env_msg(0,20," Fonts Library");
          exit(CERO);}
      do {
          memset(s_font,'\0',26);
          memset(s_tifon,'\0',5);
          fgetpos(f_lib,&i_posreg);
          fread(&r_font,sizeof(r_font),UNO,f_lib);
          if (feof(f_lib) != CERO)
                {if(i_nummod == CERO)
                    env_msg(0,8,s_nombre);
                else
                    env_msg(0,9,s_nombre);
                l_finval= TRUE;}
          else
                {
                strncpy(s_font,r_font.s_nomfon,25);
                strncpy(s_tifon,r_font.s_tipo,CUATRO);
                if ((strcmp(s_nombre,s_font) == CERO) &&
                    (strcmp(s_estilo,s_tifon) == CERO))
                    {l_finval=TRUE;
                    if (i_nummod == CERO)
                        env_msg(0,7,s_nombre);
                    l_existe=TRUE;}
                }
          }
      while (l_finval == FALSE );
      fclose(f_lib);
      return(l_existe);
}
```

```
/*****************************************************************
 * Funcion de Envio de Mensajes: env_msg(i_mod,i_num,s_opcion) *
 *****************************************************************/
/*                                                               */
/*  Llamada                                                      */
/*    env_msg(mod,num,cadena)                                    */
/*          donde:                                               */
/*               mod: modulo que llama a la biblioteca           */
/*                      0 = Bibliotecas                          */
/*                      1 = Compilador CDF                       */
/*                      2 = Utileria                             */
/*                      3 = Impresion                            */
/*                      4 = Admon de Impresoras                  */
/*               num: numero de mensaje                          */
/*               cadena: (opcional)                              */
/*****************************************************************/
int env_msg(i_mod,i_num,s_opcion)
int i_mod,i_num;
char s_opcion[];
{
      int i_fs;
      long i_desp;
      int i_count;
      FILE *f_lib;
      switch (i_mod) {
      case 0: if ((f_lib = fopen(ARC_BIBMSG,READ)) == NULL)
                {fprintf(stderr,"No puedo abrir archivo BIB Messages Library");
                exit;}
                break;
      case 1: if ((f_lib = fopen(ARC_BIBMSG1,READ)) == NULL)
                {fprintf(stderr,"No puedo abrir arcivo CDF Messages Library");
                exit;}
                break;
      case 2: if ((f_lib = fopen(ARC_BIBMSG2,READ)) == NULL)
                {fprintf(stderr,"No puedo abrir UTL Messages Library");
                exit;}
                    break;
      case 3: if ((f_lib = fopen(ARC_BIBMSG3,READ)) == NULL)
                {fprintf(stderr,"No puedo abrir archivo IMP Messages Library");
                exit;}
                break;
      case 4: if ((f_lib = fopen(ARC_BIBMSG4,READ)) == NULL)
                {fprintf(stderr,"No puedo abrir archivo APR Messages Library");
                exit;}
                break;
      )
      i_desp=(i_num-1) * sizeof(r_libmsg);
      if ((i_fs=fseek(f_lib,i_desp,SEEK_SET)) != 0)
          {env_msg(0,21," ");
          exit;}
      i_poserr=ftell(f_lib);
      fread(&r_libmsg,sizeof(r_libmsg),UNO,f_lib);
      fprintf(stderr,"SIX%d%.3s : ",i_mod,r_libmsg.s_numsg);
      for (i_count=0;i_count < strlen(r_libmsg.s_msg);i_count++)
          fprintf(stderr,"%c",r_libmsg.s_msg[i_count]);
      fprintf(stderr," %-s \n",s_opcion);
      fclose(f_lib);
      return(i_mod);
}
```

```
/*************************************************************************
 * Funcion de Validacion de Existencia de Archivos: val_arch(s_word) *
 *************************************************************************/
/*                                                                       */
/*  Llamada:                                                             */
/*    numero = val_arch(archivo)                                         */
/*  donde:                                                               */
/*       archivo:  nombre del archivo a validar                          */
/* La funcion regresara los siguientes valores                           */
/*       0   El archivo solicitado no existe                             */
/*       1   El archivo solicitado existe                                */
/*************************************************************************/

int val_arch(s_word)
char *s_word;
{
      int l_existe = FALSE;
      struct stat filestat;
      if( !stat( s_word, &filestat ) )
          l_existe=TRUE;
      else
          env_msg(0,19,s_word);
      return(l_existe);
}
```

```
/*************************************************************************
 *    Funcion de Validacion de Comando: val_comd(i_nummod)               *
 *************************************************************************/
/*                                                                       */
/*  Llamada entero = val_comd(num_mod)                                   */
/*       donde:                                                          */
/*             num_mod - numero de programa                              */
/*                 1 - programa del compilador                           */
/*                 2 - programa de utileria                              */
/*                 3 - programa de interfase                             */
/* La funcion regresara uno de los siguientes valores                    */
/*       0   Si el comando es invalido                                   */
/*       1   Si el comando es valido                                     */
/*************************************************************************/

int val_comd(i_nummod)
int i_nummod;
{
      int i_numedo = CERO;
      int i_numtok = CERO;
      int l_finsca = FALSE;
      char c_carcom = BLANCO;
      if(strlen(s_commando)==CERO || strlen(s_commando)==UNO)
          return(FALSE);
      clear_token(s_token);
      for (i_cotoka=CERO;i_cotoka < LONCAD1;i_cotoka++)
          ia_tokcom[i_cotoka][CERO]=CERO;
      i_cotoka=CERO;
      i_cocaco=CERO;
      i_cotoken=CERO;
      do {
          if (i_numedo == CERO)
                {
                c_carcom=get_charcom(s_commando);
                if (isalpha(c_carcom) != CERO)
                     {
                     s_token[i_cotoken++]=c_carcom;
                     i_numedo=TRES;
                     }
                else if (isdigit(c_carcom) != CERO)
                          {
                          s_token[i_cotoken++]=c_carcom;
                          i_numedo=CUATRO;
                          }
                     else
                     switch (c_carcom) {
                     case GUION:
                          s_token[i_cotoken++]=c_carcom;
                          i_numedo=UNO;
                          break;
                     case CR:
                          l_finsca=TRUE;
                          i_numtok=130;
                          ia_tokcom[i_cotoka++][CERO]=i_numtok;
                          break;
                     case BLANCO:
                          i_numedo=CERO;
                          break;
                     default:
```

```
                i_numedo=CINCO;
                break;}
        }
if (i_numedo == UNO)
        {
        c_carcom=get_charcom(s_commando);
        if (isalpha(c_carcom) != CERO)
                {
                s_token[i_cotoken++]=c_carcom;
                i_numedo=DOS;
                }
        else
                {
                i_numedo=CINCO;
                env_msg(0,10," ");
                exit(FALSE);
                }
        }
if (i_numedo == DOS)
        {
        c_carcom=get_charcom(s_commando);
        if (isalpha(c_carcom) != CERO)
                {
                s_token[i_cotoken++]=c_carcom;
                i_numedo=DOS;
                }
        else
        switch (c_carcom) {
        case CR:
                ter_token('a');
                l_finsca=TRUE;
                i_numtok=130;
                ia_tokcom[i_cotoka++][CERO]=i_numtok;
                break;
        case BLANCO:
                i_numedo=CERO;
                ter_token('a');
                break;
        default:
                clear_token(s_token);
                env_msg(0,10," ");
                exit(FALSE);
                break;}
        }
if (i_numedo == TRES)
        {
        c_carcom=get_charcom(s_commando);
        if (isalpha(c_carcom) != CERO)
                {
                s_token[i_cotoken++]=c_carcom;
                i_numedo=TRES;
                }
        else if (isdigit(c_carcom) != CERO)
                        {
                        s_token[i_cotoken++]=c_carcom;
                        i_numedo=TRES;
                        }
                else
                switch (c_carcom) {
```

```
                    case SUBGUION:
                         s_token[i_cotoken++]=c_carcom;
                         i_numedo=TRES;
                         break;
                    case PUNTO:
                         s_token[i_cotoken++]=c_carcom;
                         i_numedo=TRES;
                         break;
                    case CR:
                         ter_token('a');
                         l_finsca=TRUE;
                         i_numtok=130;
                         ia_tokcom[i_cotoka++][CERO]=i_numtok;
                         break;
                    case BLANCO:
                         i_numedo=CERO;
                         ter_token('a');
                         break;
                    default:
                         clear_token(s_token);
                         env_msg(0,10," ");
                         exit(FALSE);
                         break;}
               }
          if (i_numedo == CUATRO)
               {
               c_carcom=get_charcom(s_commando);
               if (isdigit(c_carcom) != CERO)
                    {
                    s_token[i_cotoken++]=c_carcom;
                    i_numedo=CUATRO;
                    }
               else
                    switch (c_carcom) {
                    case CR:
                         l_finsca=TRUE;
                         ter_token('b');
                         i_numtok=130;
                         ia_tokcom[i_cotoka++][CERO]=i_numtok;
                         break;
                    case BLANCO:
                         ter_token('b');
                         i_numedo=CERO;
                         break;
                    default:
                         clear_token(s_token);
                         env_msg(0,11," ");
                         exit(FALSE);
                         break;}
               }
          if (i_numedo == CINCO)
               {
               i_numedo=CERO;
               env_msg(0,12,&c_carcom);
               exit(FALSE);
               }
          }
     while ((c_carcom != CR) || (l_finsca==FALSE));
     if (parcommand(i_nummod) == TRUE)
```

```
            return(TRUE);
      else
            return(FALSE);
}

void clear_token(s_token)
char s_token[LONCAD1];
{
      memset(s_token,'\0',LONCAD1);
      return;
}

char get_charcom(s_commando)
char s_commando[];
{
      return(s_commando[i_cocaco++]);
}

void ter_token(c_tipo)
char c_tipo;
{
      int i_numtok,
          i_numpos;
      i_numtok=CERO;
      i_numpos=CERO;
      i_numtok=keyword_com(s_token);
      if (i_numtok != CERO)
          ia_tokcom[i_cotoka++][CERO]=i_numtok;
      else if (c_tipo == 'a')
                  { i_numtok=110;
                    i_numpos=insert_tabla(s_token,i_numtok);
                    ia_tokcom[i_cotoka][CERO]=i_numtok;
                    ia_tokcom[i_cotoka++][UNO]=i_numpos;}
            else
                  { i_numtok=120;
                    i_numpos=insert_tabla(s_token,i_numtok);
                    ia_tokcom[i_cotoka][CERO]=i_numtok;
                    ia_tokcom[i_cotoka++][UNO]=i_numpos;}
      clear_token(s_token);
      i_cotoken=CERO;
      return;
}

int keyword_com(s_word)
char s_word[];
{
      int i_potato,i_vacomp;
      for (i_potato=CERO;i_potato  <= LONCAD1;i_potato++)
          if ((i_vacomp=strcmp(s_word,k_tokcom[i_potato])) == CERO)
            break;
      if (i_vacomp == CERO)
                return(i_potato*DIEZ);
          else
                return (CERO);
}

int insert_tabla(s_word,i_numtok)
char s_word[];
int i_numtok;
```

```
{
      int i_count;
      switch (i_numtok) {
          case 110:
                for (i_count=CERO;i_count <= LONCAD1;i_count++)
                    sa_ident[i_nupoti][i_count]=s_word[i_count];
                return(i_nupoti++);
                break;
          case 120:
                ia_enter[i_nupoen]=atoi(s_word);
                return(i_nupoen++);
                break;
          }
}

int parcommand(i_nummod)
int i_nummod;
{
      int i_numtok = CERO;
      int i_numedo = CERO;
      int i_confor = CERO;
      int i_confil = CERO;
      int i_conpri = CERO;
      int i_consav = CERO;
      int i_concop = CERO;
      int i_constx = CERO;
      int i_conlst = CERO;
      int i_numpos = CERO;
      int i_count = CERO;
      int l_finpar = FALSE;
      int l_comval = FALSE;
      int i_numtoken = CERO;
      i_cotoka=CERO;
      do {
          switch (i_numedo) {
          case CERO:
                 i_numtok=get_tokcom(i_numtoken);
                 switch (i_numtok)
                  {case 110:
                     i_numpos=extract_tabla(i_cotoka);
                     for (i_count=CERO;i_count < LONCAD1;i_count++)
                        r_accion.s_file[i_count]=sa_ident[i_numpos][i_count];
                     i_numedo = UNO;
                     break;
                case  30:
                     i_numedo = CINCO;
                     break;
                case 40:
                     i_numedo = SIETE;
                     r_accion.l_opc1_lfo=1;
                     break;
                case 50:
                     i_numedo = NUEVE;
                     r_accion.l_opc2_lfr=1;
                     break;
                     default:
                          i_numedo = DIEZ;
                          break;
                     }
```

```
        break;
case UNO:
        i_numtok=get_tokcom(i_numtoken);
             switch (i_numtok)
             {case 10:
                  i_conlst++;
                  i_numedo=TRES;
                  r_accion.l_opc1_lfo=TRUE;
                  break;
             case 20:
                  i_constx++;
                  i_numedo=CUATRO;
                  r_accion.l_opc2_lfr=TRUE;
                  break;
             case 130:
                  i_numedo=DOS;
                  break;
             default:
                  env_msg(0,13,"SIXCDF");
                  return(FALSE);
             }
             if (i_conlst  UNO || i_constx  UNO)
                  {env_msg(0,14," ");
                  return(FALSE);
                  }
        break;
case DOS:
         if (i_nummod == UNO)
             return(TRUE);
         else
             {if (i_nummod == DOS)
                  env_msg(0,13,"SIXUTL");
             else
                  if (i_nummod == TRES)
                    env_msg(0,13,"SIX");
                  else
                      env_msg(0,13,"SIXCDF");
             return(FALSE);
             }
             break;
case TRES:
        i_numtok=get_tokcom(i_numtoken);
        switch (i_numtok)
        {case 20:
             i_constx++;
             i_numedo=CUATRO;
             r_accion.l_opc2_lfr=TRUE;
             break;
        case 130:
             i_numedo = DOS;
             break;
        default:
             env_msg(0,13,"SIXCDF");
             return(FALSE);
        }
        if (i_constx  UNO)
             {env_msg(0,14," ");
             return(FALSE);}
        break;
```

```
case CUATRO:
      i_numtok=get_tokcom(i_numtoken);
      switch (i_numtok)
      {case 10:
           i_conlst++;
           i_numedo=TRES;
           r_accion.l_opc1_lfo=TRUE;
           break;
      case 130:
           i_numedo = DOS;
           break;
      default:
           env_msg(0,13,"SIXCDF");
           return(FALSE);
      }
      if (i_conlst  UNO)
           {env_msg(0,14," ");
            return(FALSE);}
      break;
case CINCO:
      i_numtok=get_tokcom(i_numtoken);
      switch (i_numtok) {
      case 130:
           i_numedo=SEIS;
           break;
      default:
           env_msg(0,13,"SIXUTL");
           return(FALSE);}
      break;
case SEIS:
      if (i_nummod==DOS)
           return(TRUE);
      else
           {if (i_nummod == UNO)
                env_msg(0,13,"SIXCDF");
           else
                if (i_nummod == TRES)
                  env_msg(0,13,"SIX");
                else
                     env_msg(0,13,"SIXUTL");
           return(FALSE);
           }
      break;
case SIETE:
      i_numtok=get_tokcom(i_numtoken);
      switch (i_numtok) {
      case 110:
           i_numedo=OCHO;
           i_numpos=extract_tabla(i_cotoka);
           for (i_count=CERO;i_count < LONCAD1;i_count++)
           r_accion.s_printer[i_count]=sa_ident[i_numpos][i_count];
           break;
      default:
           env_msg(0,13,"SIXUTL -lfo");
           return(FALSE);}
      break;
case OCHO:
      i_numtok=get_tokcom(i_numtoken);
      switch (i_numtok) {
```

```
        case 130:
             i_numedo=SEIS;
             break;
        default:
             env_msg(0,14," ");
             return(FALSE);}
        break;
case NUEVE:
        i_numtok=get_tokcom(i_numtoken);
        switch (i_numtok) {
        case 110:
             i_numedo=OCHO;
             i_numpos=extract_tabla(i_cotoka);
             for (i_count=CERO;i_count < LONCAD1;i_count++)
             r_accion.s_printer[i_count]=sa_ident[i_numpos][i_count];
             break;
        default:
             env_msg(0,13,"SIXUTL -lfr");
             return(FALSE);}
        break;
case DIEZ:
        if (i_nummod == TRES)
        { switch (i_numtok) {
        case 60:
             i_numedo=11;
             i_confil++;
             break;
        case 70:
             i_numedo=17;
             i_confor++;
             break;
        case 80:
             i_numedo=19;
             i_conpri++;
             break;
        default:
             env_msg(0,13,"SIX");
             return(FALSE);
             break;}
        if (i_confil  UNO || i_confor  UNO || i_conpri  UNO)
             {env_msg(0,14," ");
             return(FALSE);}
             }
        else
             if (i_nummod == UNO) {
                  env_msg(0,13,"SIXCDF");
                  return(FALSE);}
             else
                  { env_msg(0,13,"SIXUTL");
                  return(FALSE);}
        break;
case 11:
        i_numtok=get_tokcom(i_numtoken);
        switch (i_numtok) {
        case 110:
             i_numedo=12;
             i_numpos=extract_tabla(i_cotoka);
             for (i_count=CERO;i_count < LONCAD1;i_count++)
             r_accion.s_file[i_count]=sa_ident[i_numpos][i_count];
```

```
            break;
        default:
            env_msg(0,15," ");
            return(FALSE);
            break;}
        break;
case 12:
        i_numtok=get_tokcom(i_numtoken);
        switch (i_numtok) {
        case 90:
            i_numedo=14;
            i_concop++;
            break;
        case 100:
            i_numedo=16;
            r_accion.l_savefile=TRUE;
            i_consav++;
            break;
        case 130:
            i_numedo= 13;
            break;
        default:
            i_numedo=10;
            break;}
        if (i_concop  UNO || i_consav  UNO)
            {env_msg(0,14," ");
             return(FALSE);}
        break;
case 13:
        if (i_nummod = TRES)
            return(TRUE);
        else
            {if (i_nummod == DOS)
                env_msg(0,13,"SIXUTL");
            else
                if (i_nummod == UNO)
                  env_msg(0,13,"SIXCDF");
                else
                    env_msg(0,13,"SIX");
            return(FALSE);}
        break;
case 14:
        i_numtok=get_tokcom(i_numtoken);
        switch (i_numtok) {
        case 120:
            i_numedo=15;
            i_numpos=extract_tabla(i_cotoka);
            if (ia_enter[i_numpos] <= UNO)
                r_accion.i_copias=UNO;
            else
                r_accion.i_copias=ia_enter[i_numpos];
            break;
        default:
            env_msg(0,16," ");
            return(FALSE);
            break;}
        break;
case 15:
        i_numtok=get_tokcom(i_numtoken);
```

```
        switch (i_numtok) {
        case 100:
             i_numedo=16;
             r_accion.l_savefile=TRUE;
             i_consav++;
             break;
        case 130:
             i_numedo=13;
             break;
        default:
             i_numedo=10;
             break;}
        if (i_consav  UNO)
             {env_msg(0,14," ");
             return(FALSE);}
        break;
   case 16:
        i_numtok=get_tokcom(i_numtoken);
        switch (i_numtok) {
        case 90:
             i_numedo=14;
             i_concop++;
             break;
        case 130:
             i_numedo=13;
             break;
        default:
             i_numedo=10;
             break;}
        if (i_concop  UNO)
             {env_msg(0,14," ");
             return(FALSE);}
        break;
  case 17:
        i_numtok=get_tokcom(i_numtoken);
        switch (i_numtok) {
        case 110:
             i_numedo=18;
             i_numpos=extract_tabla(i_cotoka);
             for (i_count=CERO;i_count < LONCAD1;i_count++)
             r_accion.s_forma[i_count]=sa_ident[i_numpos][i_count];
             break;
        default:
             env_msg(0,17," ");
             return(FALSE);
             break;}
        break;
  case 18:
        i_numtok=get_tokcom(i_numtoken);
        if (i_numtok == 130 )
             i_numedo=13;
        else
             i_numedo=10;
        break;
  case 19:
        i_numtok=get_tokcom(i_numtoken);
        switch (i_numtok) {
        case 110:
             i_numedo=18;
```

```
                    i_numpos=extract_tabla(i_cotoka);
                    for (i_count=CERO;i_count < LONCAD1;i_count++)
                    r_accion.s_printer[i_count]=sa_ident[i_numpos][i_count];
                    break;
               default:
                    env_msg(0,18," ");
                    return(FALSE);
                    break;}
               break; }
     }
     while (l_finpar==FALSE);
}

int get_tokcom(i_num)
int i_num;
{
     i_num=ia_tokcom[i_cotoka++][CERO];
     return(i_num);
}

int extract_tabla(i_postab)
int i_postab;
{
     int i_numtok;
     i_numtok=ia_tokcom[--i_postab][CERO];
     switch (i_numtok) {
          case 110:
               return(ia_tokcom[i_postab][UNO]);
               break;
          case 120:
               return(ia_tokcom[i_postab][UNO]);
               break;}
}

void copycom(s_parcom)
char s_parcom[];
{
     int i_count;
     for (i_count=CERO; i_count <= strlen(s_parcom); i_count++)
          s_commando[i_cocaco++]=s_parcom[i_count];
     return;
}
```

```
/*****************************************************************
 * Funcion de Conversion a Lower case: cadlwr(s_cadena)          *
 *****************************************************************/
/*                                                               */
/*  Llamada                                                      */
/*    cadlwr(cadena)                                             */
/*          donde:                                               */
/*               cadena: es la cadena que se desea convertir     */
/*                                                               */
/* La funcion regresa la cadena convertida a un apuntador        */
/*                                                               */
/*****************************************************************/


char * cadlwr(s_oricad)
char *s_oricad;
{
      char *ps_tstcad;
       /* Obtengo las minusculas de la cadena */
       for( ps_tstcad = s_oricad; *ps_tstcad; ps_tstcad++ )
            if( isupper( *ps_tstcad ) )
                  *ps_tstcad = tolower( *ps_tstcad );
       return(s_oricad);
}
```

```
/*****************************************************************
 * Funcion de eliminacion de espacios: cadtrim(s_cadena)         *
 *****************************************************************/
/*                                                               */
/*  Llamada                                                      */
/*    cadtrim(cadena)                                            */
/*               cadena: es la cadena que tiene los espacios     */
/*                                                               */
/* La funcion regresa la cadena sin espacios a un apuntador      */
/*                                                               */
/*****************************************************************/


char * cadtrim(s_oricad)
char *s_oricad;
{
      char s_tstcad[50];
      char *ps_tstcad;
      int i_count,i_loncad;
       for(i_count=0;i_count2;i_count++)
           s_tstcad[i_count]='\0';
       i_count=0;
       i_loncad=strlen(s_oricad);
       /* Obtengo la cadena sin espacios */
       do {
            if(!isspace(*s_oricad))
                     s_tstcad[i_count++] =  *s_oricad++ ;
            else
                     *s_oricad++;
       }
       while(  i_count  i_loncad );
       ps_tstcad=strdup(s_tstcad);
       return(ps_tstcad);
}
```

4.3 Relación entre archivos y programas

Como se mencionó en el párrafo 4.2 (características de implantación) existe una relación entre cada uno de los módulos que componen el sistema, pero falta por especificar los nombres físicos de esos archivos para tener una referencia más rápida al momento de dar mantenimiento o mejorar el sistema; ya que como es muy común en el ámbito de los sistemas de computación y más que nada entre los programadores la de olvidar o desatender los nombres de los archivos y cual era su función. A continuación se describe la relación entre archivos y programas.

El sistema SOFIX esta formado por los primeros 5 programas que se listan, los demás nombres se explicarán en su momento.

1.- SIXCDF(compilador de formas)

2.- SIXUTL (Utilería)

3.- SIX (Impresión de formas y con datos variables)

4.- SIXAPR (Administrador de impresoras)

5.- SIXINSTALL (Instalación del producto)

6.- SIXBIB (Biblioteca de funciones comunes a todos los programas)

7.- SIXTIPOS (Archivo de definiciones utilizado por SIXBIB)

8.- SIXBIB.MSG (Archivo de mensajes de error para la biblioteca SIXBIB)

9.- SIXFORM.LIB (Archivo librería de formas)

10.- SIXFONT.LIB (Archivo librería de fonts)

11.- SIXIMP.LIB (Archivo librería de impresoras)

Cada uno de los programas antes mencionados se encargan de una función en especial; esa función se describe a detalle en la parte del análisis de requerimientos y en el diseño del producto. Por lo tanto sólo recordaremos brevemente que hacen cada uno de ellos.

4.3.1 SIXCDF(compilador de formas)

Se encarga de compilar el archivo que contiene la forma previamente diseñada, para generar el código necesario para la impresora y fusionarla con los datos variables.

SIXCDF tiene relación con SIXBIB el cual es un programa que se encarga de validar entre otras cosas la licencia de uso del producto, el comando de compilación y sus opciones, la existencia del archivo (en este caso la forma fuente) y la existencia de la forma.

4.3.2 SIXUTL (Utilería)

También conocido como interfase de adquisición de fonts o de listado de fonts o formas; el cual se encarga de mantener actualizada la librería de fonts (leer fonts de disco flexible y grabarlos si así se desea en la librería de fonts SIXFONT.LIB), este programa se relaciona con SIXBIB que en este caso valida la licencia de uso, valida el comando correspondiente y en caso de solicitar listado ya sea de fonts o de formas SIXBIB valida la existencia de fonts o de formas respectivamente (es decir hace referencia a SIXFONT.LIB o SIXFORM.LIB).

4.3.3 SIX (Impresión de formas y con datos variables)

Se encarga de tomar el archivo con el código que generó el programa SIXCDF que junto con los archivos de las librerías de formas (SIXFORM.LIB), la librería de fonts (SIXFONT.LIB), el archivo de la librería de impresoras (SIXIMP.LIB) y el programa SIXBIB (que valida el comando para tal fin) produce como resultado una forma impresa con datos variables y los fonts que se hayan seleccionado.

4.3.4 SIXAPR (Administrador de impresoras o definición de impresoras)

Tiene la función de mantener actualizado el archivo SIXIMP.LIB es decir definir nuevas impresoras, dar de baja o modificar datos así como consulta de los mismos. Este programa tiene relación con SIXBIB el cual valida la licencia de uso, valida el nombre de la impresora.

4.3.5 SIXINSTALL (Instalación del producto)

Este programa se encarga de la instalación del producto es decir copia los programas y archivos necesarios para que el sistema funcione adecuadamente.

4.3.6 SIXBIB (Biblioteca de funciones comunes a todos los programas)

Este a su vez tiene relación con los archivo SIXTIPOS y SIXBIB.MSG los cuales contienen las definiciones necesarias de todas las rutinas internas, así como los mensajes de error que se despliegan en el momento de detectarse.

Así como SIXBIB tiene relación con un archivo de mensajes SIXBIB.MSG algunos de los programas como SIXCDF, sixapr tienen a su vez archivos similares, además de que cada uno de los programas tienen o están formados por mas de una subrutinas cuyos nombres no viene al caso mencionar en este momento ya que esa información se dejará en un manual de desarrollo.

Con el fin de garantizar que el producto cumple con los criterios de aceptación definidos en el capítulo de Análisis de requerimientos, se llevaron a cabo tanto pruebas unitarias, durante la etapa de desarrollo, como pruebas integrales, durante la etapa de pruebas. Las pruebas realizadas fueron de tipo funcional, de desempeño, de tensión y de estructura, con estas pruebas fué posible determinar si el producto desarrollado cumplía o no con los factores de calidad definidos como criterios de aceptación (correctividad, confiabilidad, eficiencia, solidez, usabilidad, integridad, flexibilidad y portabilidad).

5.1 Pruebas funcionales y de estructura:

5.1.1 Compilación de formas

Para probar esta función del producto se compilaron diversas formas, con errores y sin errores, usando el comando "sixcdf" con sus diferentes opciones:

sixcdf forma.fdl	- Compila forma fuente (forma.fdl)
sixcdf forma.fdl -l	- Compila y genera listado de la forma fuente
sixcdf forma.fdl -s	- Revisa sintaxis
sixcdf forma.fdl -l -s	- Revisa sintaxis y genera listado
sixcdf forma.fdl -s -l	- Revisa sintaxis y genera listado

Cuando se procesaron formas fuente con algún tipo de error, sintáctico o semántico, el compilador fué capaz de detectarlos y reportarlos con un mensaje de error, los mensajes generados siempre fueron los esperados. Para las formas fuente sin errores el compilador generó las secuencias de escape, cuando se uso el comando sin la opción -s, o generó un mensaje indicando que no se habían detectado errores de sintaxis, cuando se uso el

comando con la opción -s. Con la opción -l el compilador generó un archivo conteniendo un listado de la forma fuente.

5.1.2 Altas, bajas, cambios y consultas de impresoras.

En primer lugar se dieron de alta varias impresoras con diferentes características, posteriormente se realizaron consultas para verificar que quedaron dadas de alta correctamente. Después de esto se hicieron algunos cambios en las características de las mismas impresoras y para checar estos cambios también se realizaron las consultas adecuadas. El siguiente paso fué dar de baja algunas impresoras y, de igual forma a los pasos anteriores, se hicieron las consultas correspondientes para verificar que efectivamente dichas impresoras quedaron dadas de baja. Finalmente se intentaron dar de alta impresoras que ya existían y se intentaron dar de baja impresoras que no existían, en estas dos últimas actividades se recibieron los mensajes de error correspondientes.

5.1.3 Altas de fonts

Las pruebas de esta función consistieron en dar de alta los fonts externos desde diskette a la librería de fonts, el comando utilizado fué "sixutl add "

5.1.4 Generación de listados de fonts y formas

Para probar estas funciones se utilizaron los comandos "sixutl -lfo impresora" y "sixutl -lfr impresora" respectivamente, en ambos casos se imprimió, en la impresora indicada en el comando, un listado de fonts o formas existentes en la librería correspondiente.

5.1.5 Impresión de formas

Con el uso del comando "six" se llevaron a cabo las pruebas de esta función, se imprimió un ejemplo de la forma definida como default, se mezeló un archivo de datos con la forma de default con original y copia.

5.1.6 Instalación del producto

Aunque la instalación del producto no es en realidad una función, fué importante realizar las pruebas para garantizar que el usuario no tendrá problemas al instalarlo. Se realizaron intentos de instalación con password y licencia válidas e inválidas, en el primer caso la instalación se pudo realizar sin ningún problema y en el segundo caso, como se esperaba, no fué posible realizar la instalación.

Después de concluir las pruebas funcionales del producto podemos asegurar que éste cumple con los siguientes factores de calidad:

Correctividad

Se logró el objetivo propuesto, es decir, este producto es la solución efectiva al problema propuesto.

Confiabilidad

Su funcionamiento siempre fué adecuado, los resultados esperados en cada prueba fueron los obtenidos.

Solidez

El producto fué capaz de detectar todos los errores que se cometieron durante las pruebas y de seguir funcionando correctamente.

Usabilidad

Durante las pruebas se pudo comprobar que el producto es fácil de usar, todos los comandos usados en el producto son sencillos de escribir y los mensajes de error generados orientan al usuario para corregir sus errores.

Integridad

El producto cuenta con los mecanismos necesarios de seguridad, password y licencia, que garantizan que solo personas autorizadas puedan hacer uso del producto y de la información que este procesa.

Portabilidad

En virtud de que el producto fué desarrollado bajo el sistema operativo MS-DOS y que fué instalado bajo el sistema operativo UNIX realizando un mínimo de cambios o adaptaciones, podemos decir que cumple con esta caraterística.

5.2 Pruebas de desempeño y tensión

Las pruebas funcionales y de estructura realizadas también sirvieron para hacer simultaneamente las pruebas de desempeño y tensión, se realizaron acciones como las siguientes:

- Compilación de formas muy elaboradas.

- Impresión de formas complejas

Con este tipo de pruebas se pudo comprobar que el producto hace un uso mínimo de recursos de cómputo y tiempo cuando procesa volúmenes grandes de información, es decir, cumple con el factor de calidad denominado Eficiencia.

CONCLUSIONES

La llegada de Los Sistemas de Impresión Electrónica han evolucionado la forma de imprimir los documentos provenientes de un equipo de cómputo, ya que permiten eliminar el uso de papel pre-impreso y en consecuencia los problemas que esto representa. A lo largo de este trabajo, hemos analizado ampliamente a estos sistemas, con la finalidad de obtener formas electrónicas que permitan a los usuarios el diseño de las mismas de manera transparente y la reducción de sus costos.

Con el uso de un Lenguaje de Descripción de Formas en un Sistema de Impresión Electrónica es posible diseñar formas e imprimir éstas junto con la información requerida en un mismo paso, en el propio site o lugar de trabajo, sin necesidad de depender de una imprenta o algún otro método de impresión a gran escala, para las formas requeridas. Las formas que se diseñen con este lenguaje pueden ser almacenadas y modificadas las veces que sea necesario, mejorando con esto la productividad y disminuyendo considerablemente los gastos en la impresión de documentos.

Como consecuencia de lo anterior, encontramos en el mercado aplicaciones que permiten la elaboración de formas electrónicas de manera transparente; pero que por los requerimientos necesarios para su operación, elevan sus costos y sólo se hacen accesibles a las grandes corporaciones empresariales. Con ello surge la inquietud de crear y difundir un producto de software, capaz de contemplar en su diseño, la tecnología empleada en impresoras de alto volumen (impresiones masivas) para crear formas electrónicas en impresoras de bajo volumen (menor capacidad) y que a su vez permita la reducción de costos. Adicionalmente al objetivo principal del presente proyecto se desea difundir el uso y correcta aplicación de los métodos, programas y herramientas (hardware) empleados por los grandes corporativos para impresiones masivas en impresoras de menor capacidad.

Actualmente en el ambiente Unix no existen productos de software comerciales que permitan diseñar formas de una manera sencilla, sino que deben usarse secuencias de escape que complican el uso transparente y eficiente de los recursos de impresión. El producto de software desarrollado durante este trabajo permite incorporar la tecnología de Impresión Electrónica al ambiente Unix, ya que el Lenguaje Descriptor de Formas que forma parte de este producto permite crear formas electrónicas de calidad de una manera sencilla y fácil, además de poder ser almacenadas y modificadas sin perder oportunidad, permitiendo el uso inmediato de los dispositivos de impresión.

En el transcurso de este proyecto, se pudieron claramente notar las diferencias entre los lenguajes de impresión XES que se emplea en impresoras XEROX y PCL para impresoras HEWLETT PACKARD o compatibles con las mismas. Es importante mencionar que la facilidad de uso es uno de los puntos prioritarios en cualquier lenguaje de programación, que en el caso de XES es muy bien alcanzado, ya que a pesar de ser secuencias de escape para la impresión de documentos, es muy fácil de entender y usar.

En el caso de PCL, aunque es más complicado de entender, su uso es más extendido y por lo mismo es sumamente importante que nuestro proyecto lo haya contemplado. Las principales diferencias entre dichos lenguajes es el manejo tipográfico, el control del tamaño lógico del área de impresión y el control de formas residentes en la memoria de la impresora.

En el caso de XES, este lenguaje es capaz de dejar una forma a imprimir en la memoria de la impresora, mientras que en PCL lo que se almacena es un conjunto de secuencias de escape que cada vez que se haga un llamado a ellas se procesaran por la impresora. Esto, aunque de primera instancia pudiera parecer una desventaja, representa una ventaja porque se puede almacenar en la memoria de la impresora más de una forma a la vez.

Es importante mencionar que con respecto a la información necesaria sobre los lenguajes de impresioón XES y PCL4, se tenía un total desconocimiento por parte de la mayoría de los integrantes del equipo, por ello fue necesario realizar varias reuniones con personas conocedoras del tema, que permitiera al equipo conocer más a fondo el funcionamiento de las impresoras láser. Asimismo era imperativo obtener manuales o información técnica referente a los lenguajes mencionados, el relativo a XES fue bastante fácil, sin embargo para PCL4, no fue posible obtenerla fácilmente, debido a que ni en la misma empresa Hewlett-Packard estaba disponible comercialmente, esta información se consiguió por medio de otras personas, usuarios de impresoras HP.

El Producto de Software para Incorporar la Tecnología de Impresión Electrónica en el Ambiente Unix desarrollado permite manejar eficientemente los recursos de impresión, en un ambiente de desarrollo y producción, aumentando la productividad y la flexibilidad en el cambio. Este producto proporciona una solución integral al problema planteado al inicio de este trabajo, por lo tanto podemos concluir que el objetivo fijado se ha logrado con la terminación de este producto

La creación de un producto de software no disponible en el mercado, pone en evidencia la necesidad de que los programas hechos a la medida como se les conoce comúnmente no demeritan su calidad ya que se les ha llegado a considerar como programas que continuamente hay que estar modificando o poniendo parches para que funcionen adecuadamente; todo este trabajo adicional es muy fácil de evitar teniendo una buena planeación del trabajo y definir una estrategia de solución pero sobretodo tener un buen diseño, para que el mantenimiento del sistema no sea un verdadero caos; el producto se apego a los estándares de programación y a las técnicas de diseño y de programación disponibles hasta este momento.

El haber utilizado una metodología de desarrollo de sistemas ha hecho posible establecer una adecuada estrategia de solución y planeación de trabajo, logrando con ello que la implementación se realizara de una manera sencilla y con poco cambio en su estructura. Por lo que se puede observar que la creación de software es una actividad que basa su calidad en la utilización de herramientas adecuadas que permitan obtener los resultados esperados con una apropiada planeación.

El desarrollo de este proyecto, no sólo ha permitido reafirmar y aplicar aspectos técnicos que van desde el diseño de un compilador, el diseño de un lenguaje, el diseño de estructura de datos, la generación de código, etc.; sino también, ha permitido conocer y establecer las diferencias existentes en impresoras láser y lenguajes de impresión. Con ello se ha podido obtener, el control de las secuencias de escape requeridas para el diseño de formas y datos variables de cada impresora, a través de un solo lenguaje de diseño de formas, transparente para el usuario.

El producto de software para impresoras de bajo volumen, requiere aún de ciertas mejoras; sin embargo posee los elementos esenciales para el diseño de formas y de impresión de manera transparente. Si se pensara en la comercialización de este producto, restaría aún bastante trabajo para mejorar su presentación, simplificar y extender rutinas de programación tanto para XEROX como para HP; así como para otros equipos y plataformas de operación, lo cual sería esencial para este objetivo.

Se considera que si bien esta es la primera experiencia en realizar un producto de software no se desea que se quede en solo una experiencia académica, sino que se vea la aplicación práctica del mismo, lo cual motiva a continuar realizando el esfuerzo para trabajar en cualquier otra actividad, con el mismo interés y dedicación en el futuro.

Si bien el producto de software no ha sido instalado en un ambiente de producción, los resultados obtenidos en el transcurso de las pruebas ha demostrado que es de calidad, ya que cumple con los requisitos establecidos.

Con respecto a la instalación del producto, el cual fue desarrollado en un ambiente de PC y posteriormente transportado a la plataforma UNIX, nos permitió observar que no siempre es fácil realizar la migración debido en gran parte a que, aunque la documentación de Microsoft C indica que al compilar con la opción ANSI no se tendrán problemas, en la práctica esto no es cierto, se realizaron algunas modificaciones a los programas y algunas rutinas tuvieron que realizarse en el ambiente UNIX, éstas últimas existen y funcionan bien en ambiente de PC pero que no existen para el ambiente requerido

Finalmente cabe aclarar que la experiencia más positiva de este trabajo, ha sido el de diseñar y desarrollar un producto practicamente en su totalidad, haciendo uso de un mínimo de recursos y equipo. El producto de software creado, por lo tanto, ha permitido establecer la intervención de varios puntos importantes, gramáticas, lenguajes, compiladores, interfase con sistema operativo y administración de recursos de impresión, cumpliendo con el principal objetivo para el cual fue desarrollado. Por ello es posible mencionar que el desarrollo de software en México puede estar al nivel de cualquier país del mundo.

Hemos mencionado los aspectos técnicos que ha aportado el desarrollo del presente trabajo, pero también intervienen los aspectos humanos en cualquier relación laboral, es por ello que sobre los aspectos humanos se puede comentar que no es fácil ponerse de acuerdo, sin embargo es posible. Gracias a una adecuada dirección y voluntad de superación se pudo terminar el trabajo ya que sin empeño no se hubiera alcanzado el objetivo inicial.

Es importante resaltar que aunque es difícil la relación humana, esta tiene como beneficio el conocer puntos de vista diversos que permiten la generación múltiple de ideas (sinergía) y puntos de vista, para alcanzar un conocimiento más amplio.

Página dejada en blanco intencionalmente

ANEXO A

Los diagramas de flujo de datos que se definen para el producto, los cuales se encuentran en el Capítulo II (Análisis de Requerimientos), deben ser expandidos generando con ello la necesidad de incursionar en cada uno de los procesos identificados, así se obtienen nuevos diagramas de flujo, que permitan ver con más detalle lo que se desea hacer.

Es por ello que este anexo se ha desarrollado para explicar y mostrar cada uno de los subprocesos.

A.1 Interfase del usuario para mantenimiento de impresoras.

La interfase de mantenimiento de impresoras tiene como objetivo permitir al usuario la actualización dinámica del archivo de impresoras, como se puede observar en el siguiente diagrama de flujo de datos; la expansión de cada uno de los módulos que lo componen se encuentran en las páginas posteriores

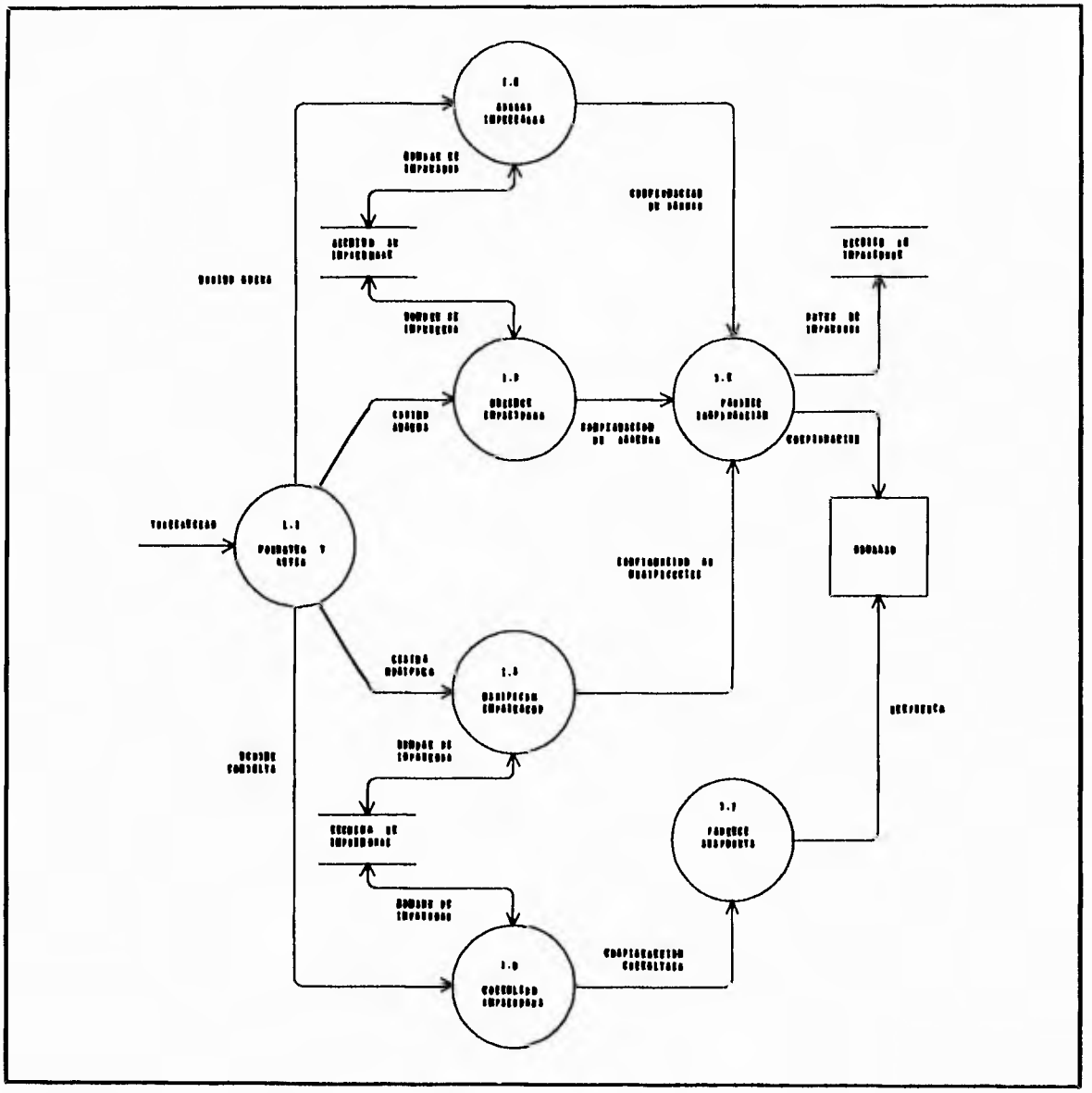


Fig. 2.3 Mantenimiento de impresoras

Formatea y Rutea.

La función de Formatea y Rutea del módulo de mantenimiento de impresoras, será la de presentar en pantalla las opciones para que el usuario seleccione el tipo de movimiento que desea realizar sobre la librería de Impresoras. Cualquier acción tomada por el usuario será validada por el módulo, si se detecta algún error se emitirá el mensaje adecuado en pantalla, de lo contrario se invocará al módulo correspondiente según la selección del usuario.

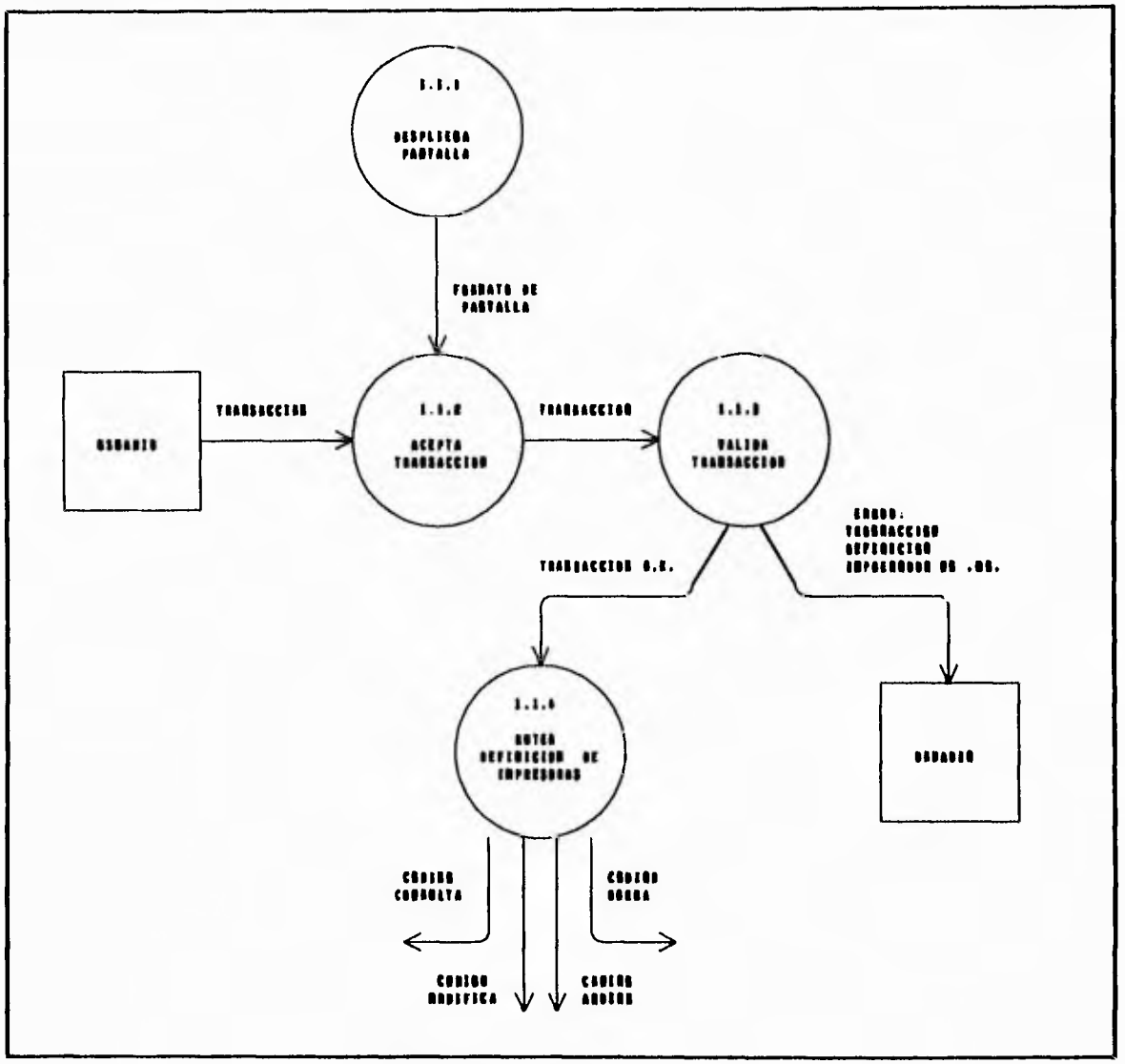


Fig. 2.3a Módulo formatea y rutea

Borrar la definición de una impresora.

El proceso de borrar dentro del módulo de mantenimiento de impresoras, solicitará al usuario el nombre de la impresora que desea dar de baja, después validará la existencia del registro correspondiente en la librería de impresoras. Si la impresora existe, entonces se desplegará en pantalla la información relacionada con la impresora y se le pedirá al usuario que confirme el borrado del registro. Si la impresora no existe, se le informará al usuario por medio de un mensaje en pantalla.

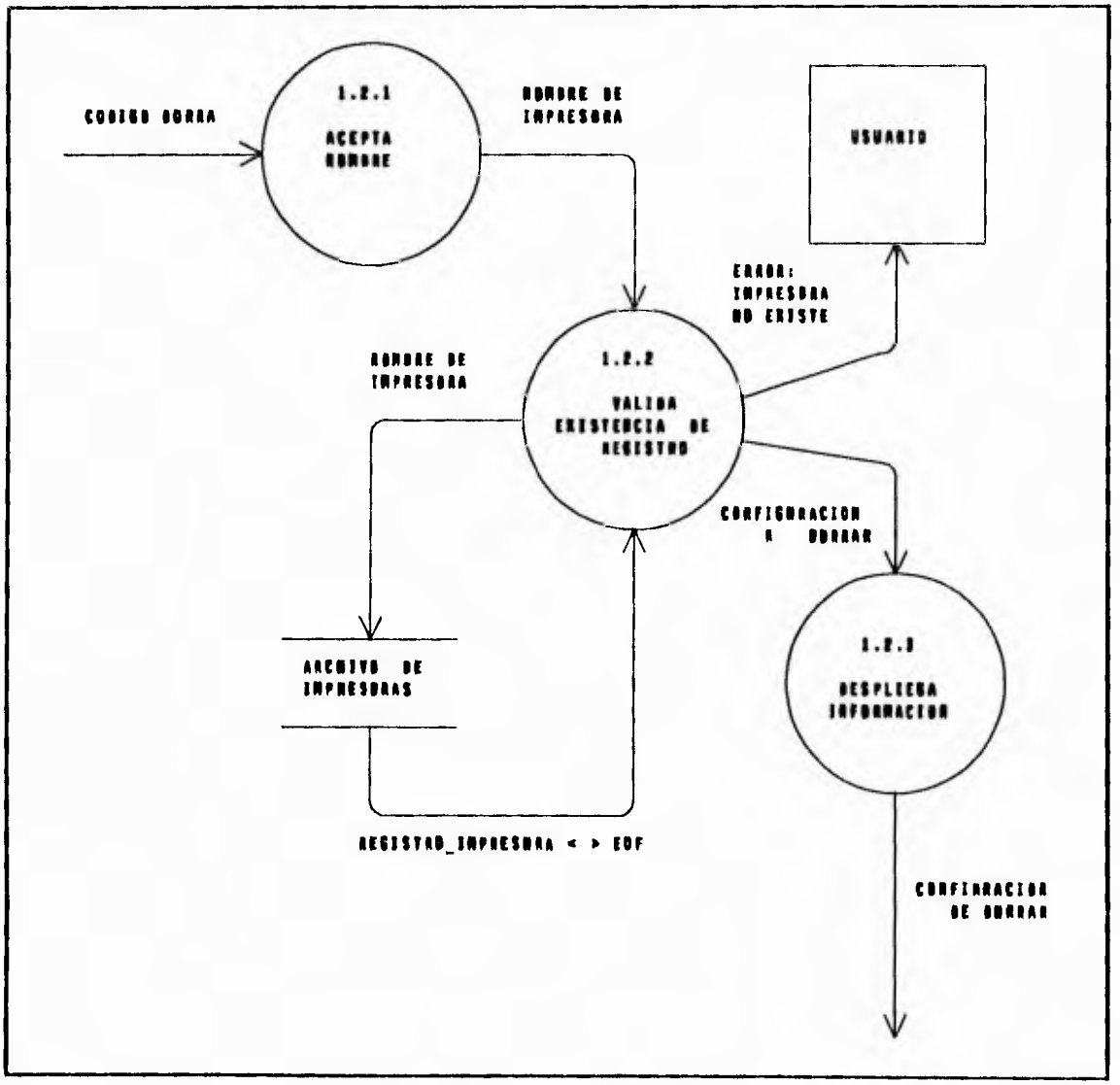


Fig. 2.3b Módulo para borrar impresoras

Agregar una definición de impresora.

El proceso para agregar información al módulo de mantenimiento de impresoras, primero solicitará el nombre de la impresora a dar de alta, posteriormente validará la no existencia del registro correspondiente en la librería de impresoras, si la impresora ya existe se emitirá un mensaje de error. Si la impresora no existe entonces el módulo solicitará los datos de la impresora como son nombre, tipo, conexión, cantidad de memoria y velocidad de transmisión y la confirmación para darla de alta.

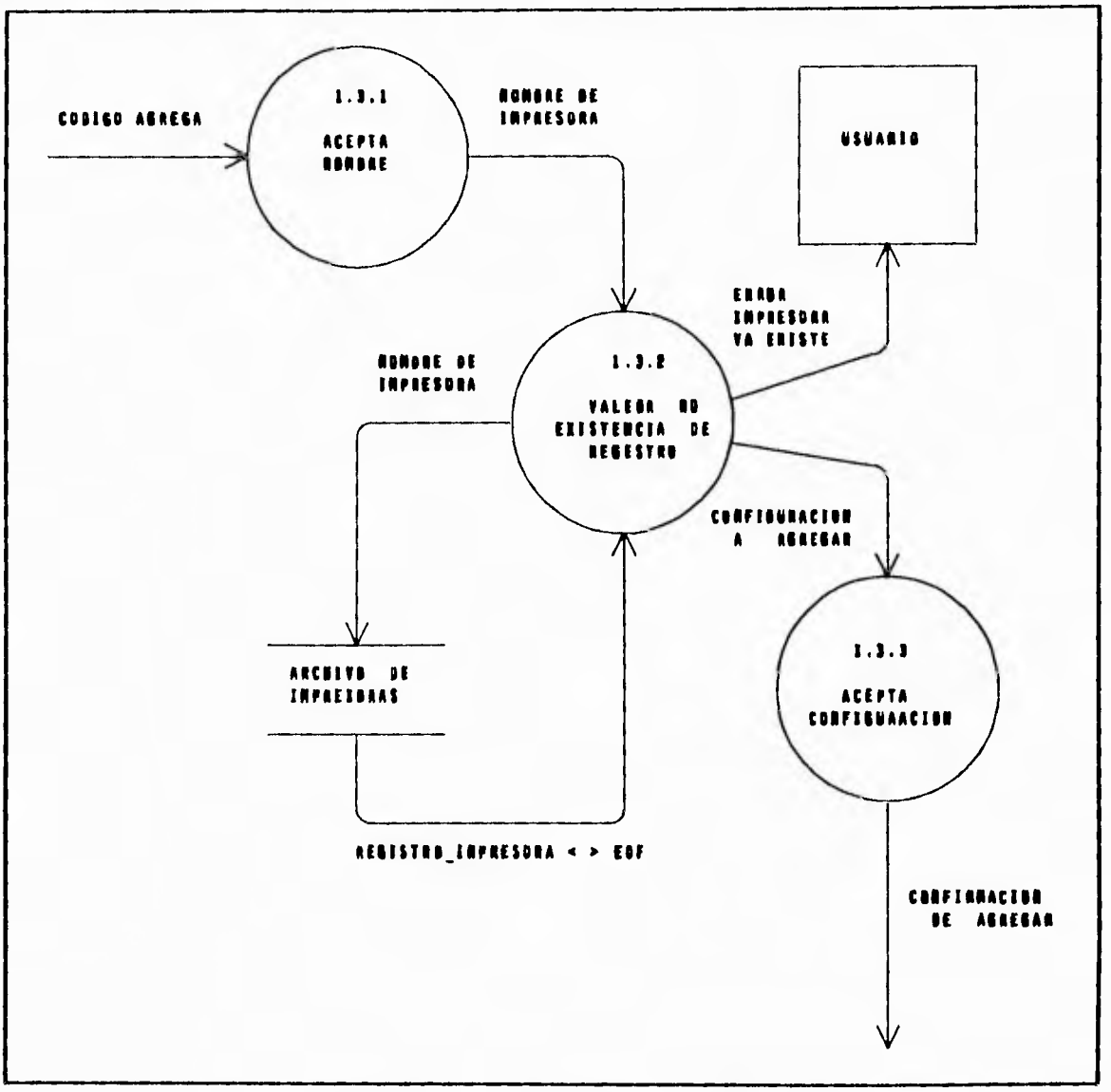


Fig. Módulo para agregar impresoras

Modificar la definición de impresora.

El proceso de modificar del módulo de mantenimiento de impresoras, solicitará el nombre de la impresora a la cual se modificarán sus datos, después validará la existencia del registro correspondiente en la librería de impresoras, si la impresora no existe, entonces se emitirá un mensaje de error a la pantalla. Si la impresora existe, entonces se desplegarán los datos de ésta para que puedan ser modificados. Antes de actualizar el registro en la librería, el módulo solicitará al usuario una confirmación.

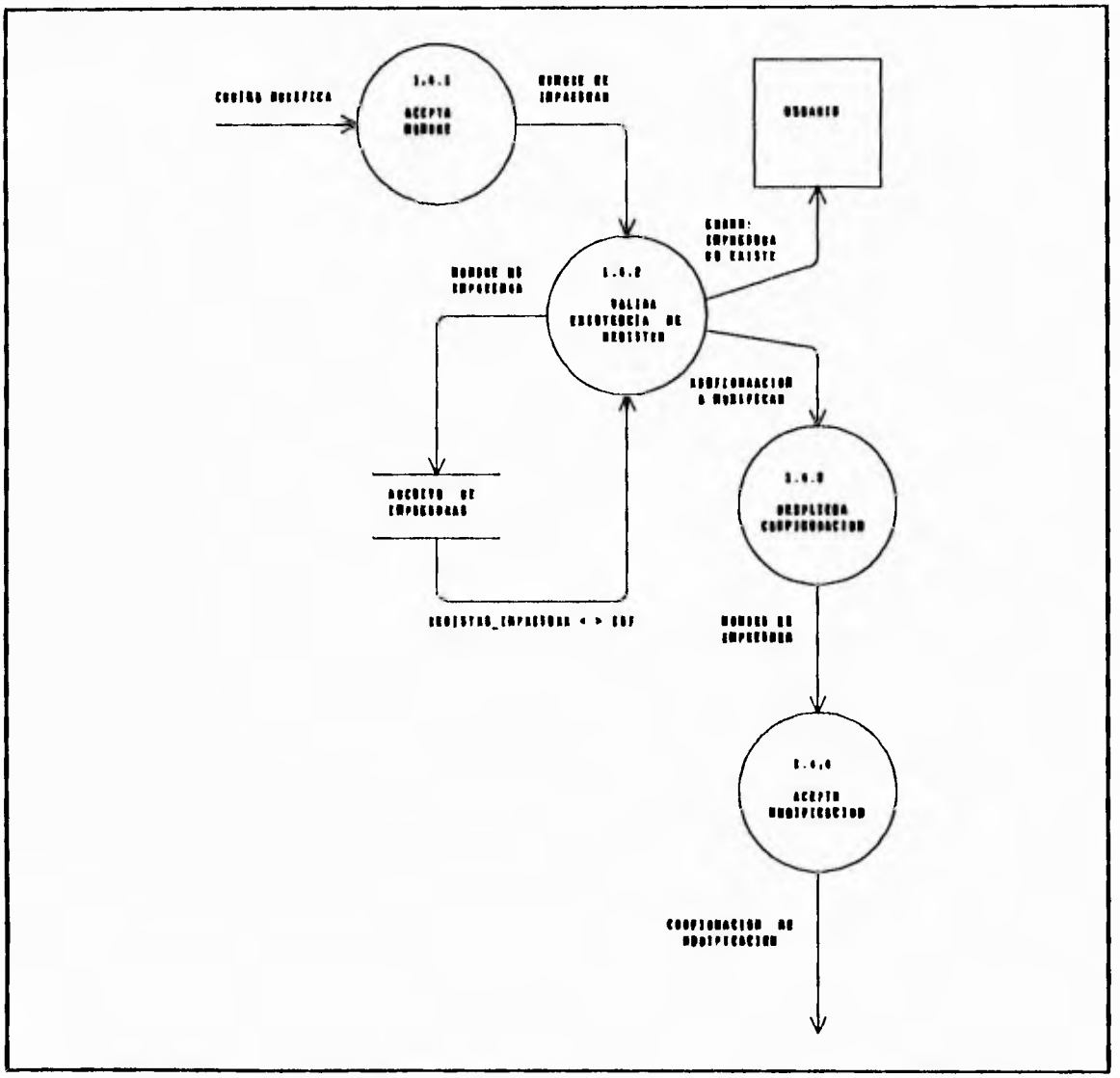


Fig. 2.3d Módulo para modificar datos de impresoras

Consultar la definición de impresora.

La primera acción del proceso de consulta para el módulo de mantenimiento de impresoras, será solicitar el nombre de la impresora a consultar, posteriormente validará la existencia del registro correspondiente en la librería de impresoras, si la impresora no existe, entonces se emitirá un mensaje de error. Si la impresora existe, entonces se desplegarán en la pantalla los datos de ésta.

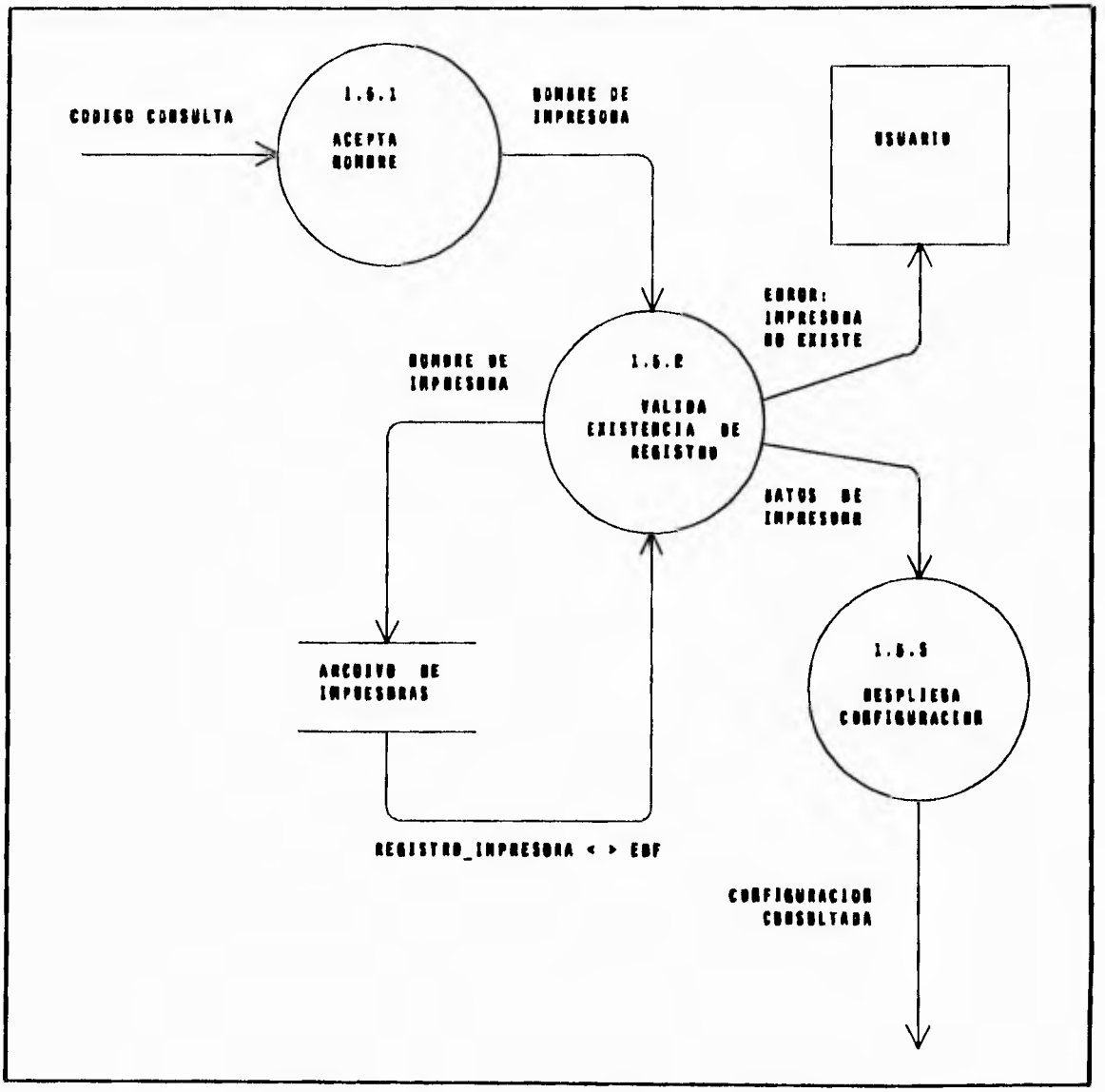


Fig. 2.3e Módulo para consultar impresoras

A.2 Interfase de utilería.

La interfase de utilería tiene como objetivo agregar fonts en el sistema para su posterior uso en las impresoras, además proporcionará un listado de fonts y formas existentes. A continuación se muestra el diagrama de flujo de datos de esta interfase, y la expansión de sus módulos en las páginas posteriores.

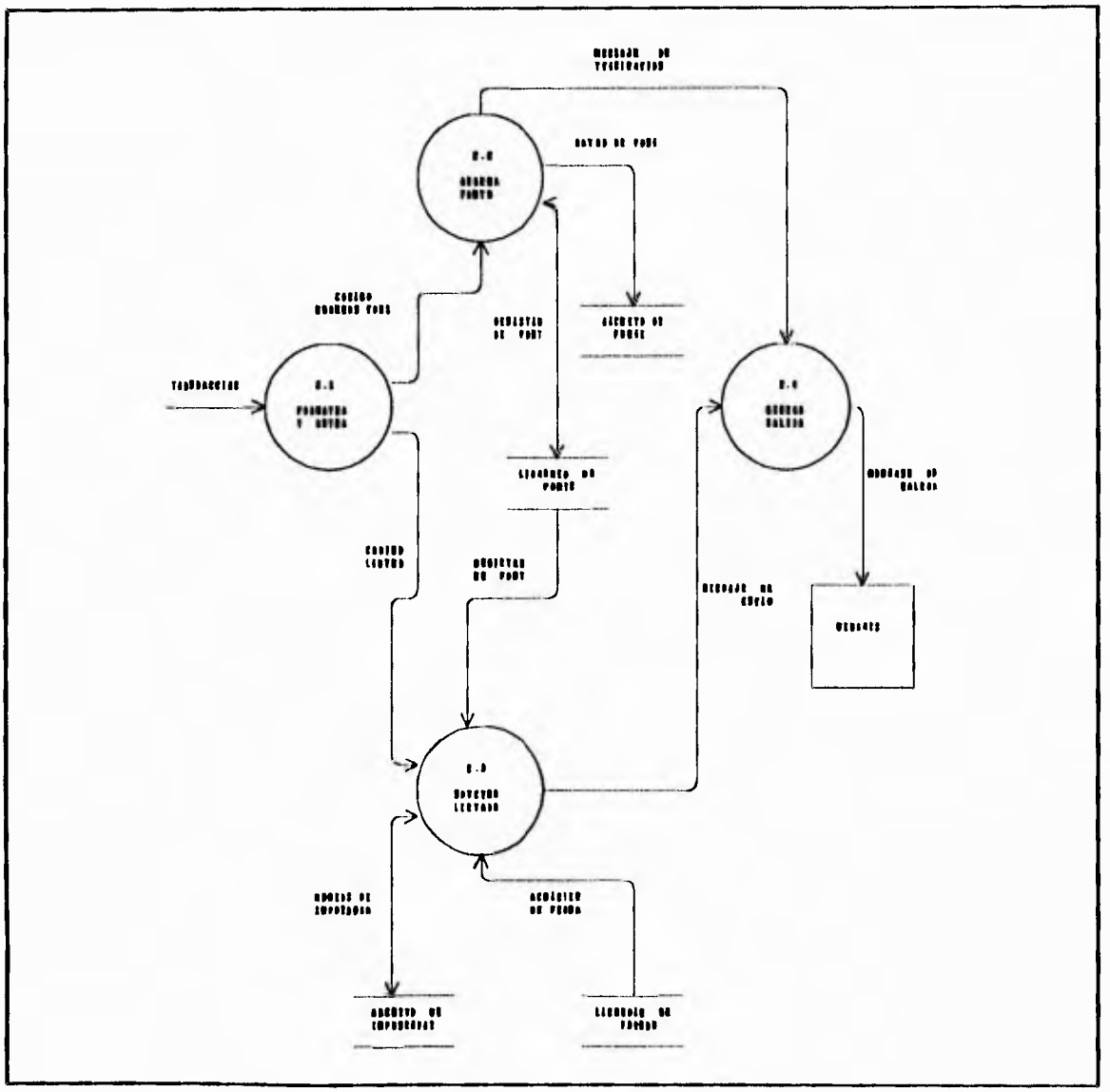


Fig. 2.4 Interfase de utilería

Formatea y rutea.

La función de formatea y rutea del módulo de interfase de utilería, será la de presentar en pantalla un menú con el tipo de transacciones que el usuario podrá efectuar y que va desde agregar fonts a la librería de fonts hasta la de abandonar el módulo. Cualquier acción tomada por el usuario será validada por el módulo y en el caso de incurrir en algún error, éste emitirá un mensaje de error a la pantalla. En el caso contrario, invocará la acción correspondiente a la selección del usuario.

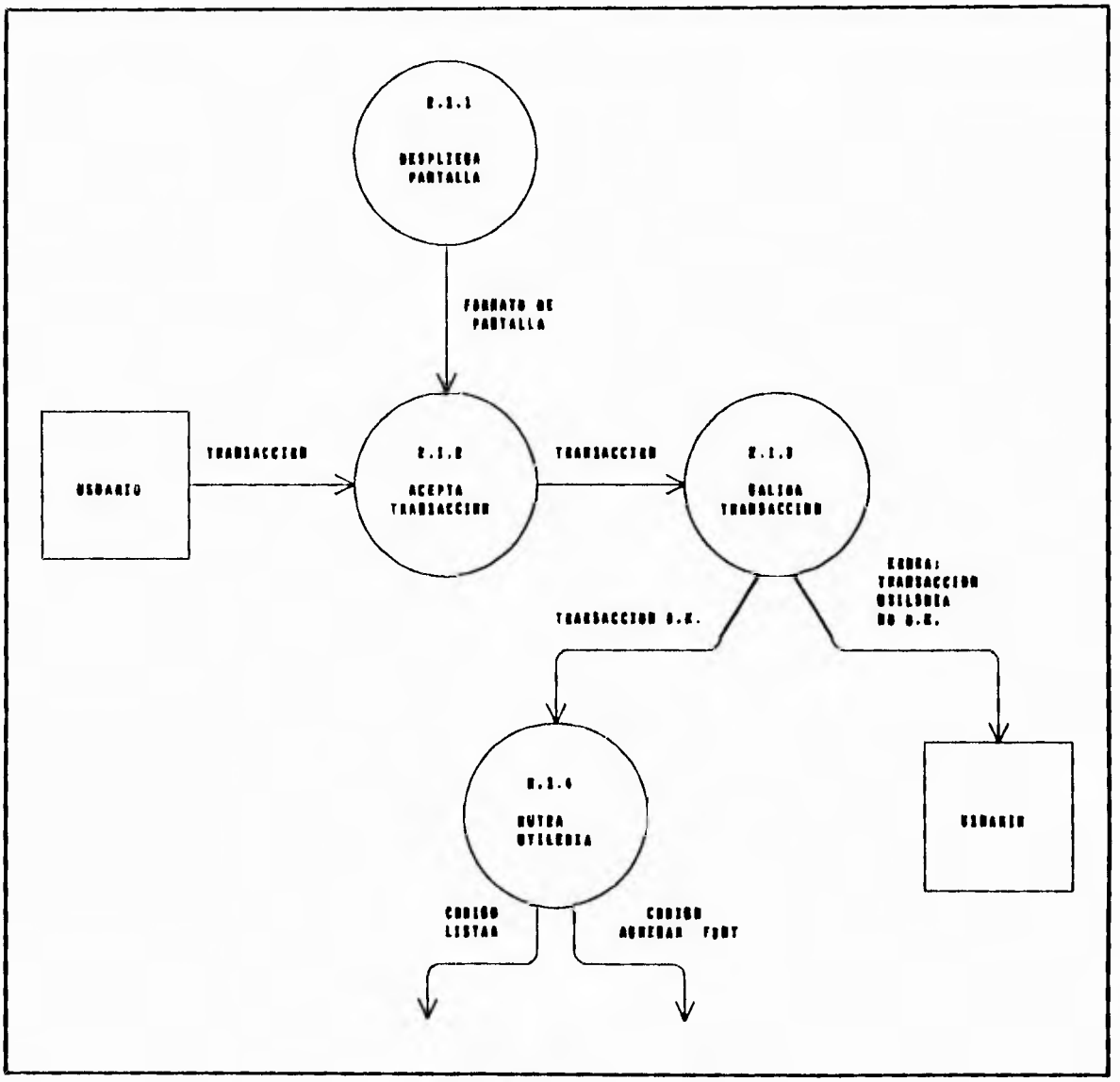


Fig. 2.4a Módulo formatea y rutea

Agrega Fonts.

La primera acción del proceso para agregar fonts, será solicitar al usuario el medio (diskette o cinta) de donde tomará los datos del font. Una vez que el registro es leído, se procesa la descripción del font y los datos del mismo son mostrados al usuario. El proceso actualizará la librería de fonts y el archivo completo de font será copiado al host. Una vez que se efectúe el proceso se enviará al usuario un mensaje de terminación.

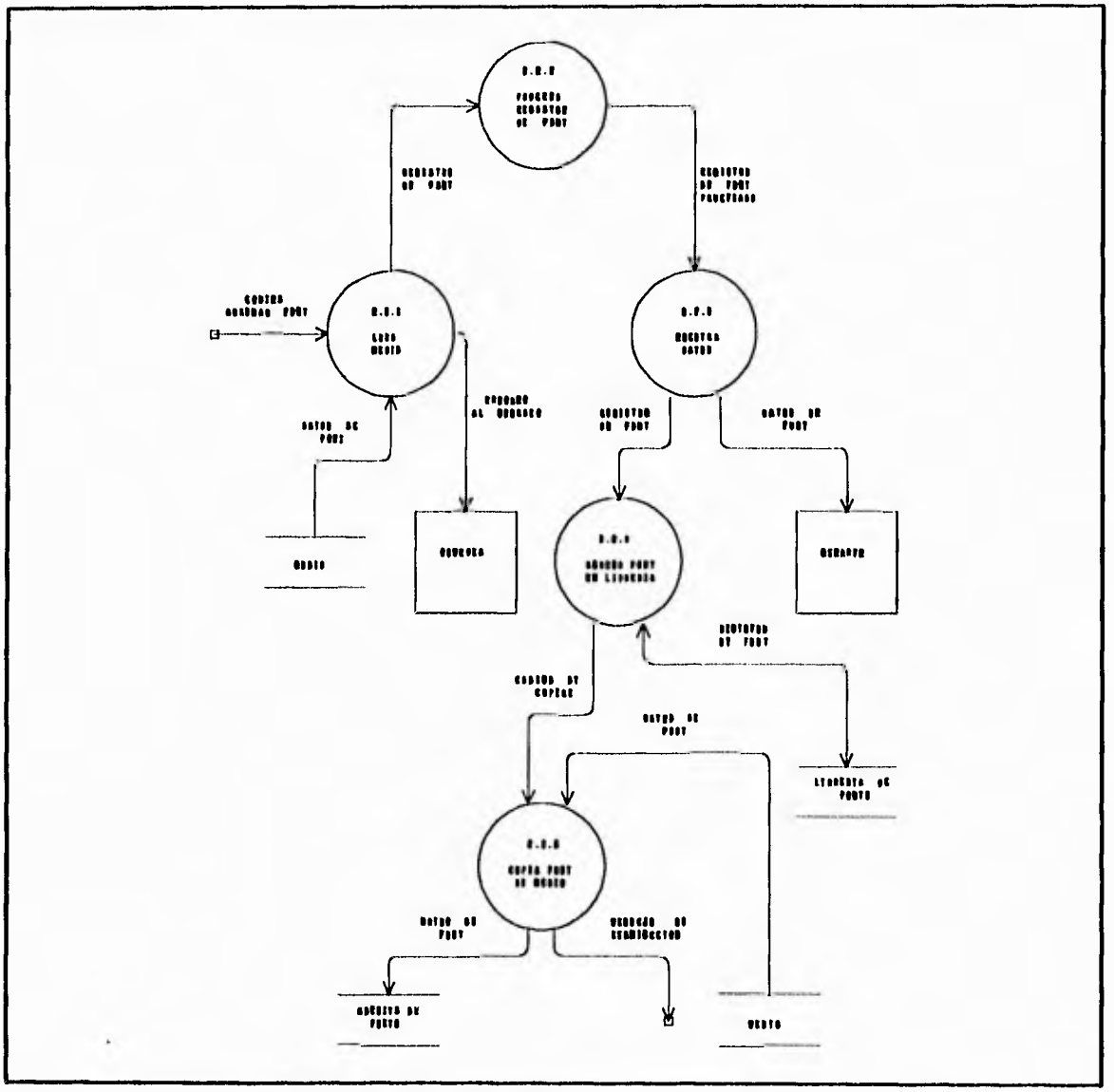


Fig. 2.4b Módulo para agregar fonts

Obtención de listados.

El proceso para la obtención de listados tanto de fonts como de formas, solicitará al usuario como primer paso, el nombre del dispositivo de salida. Una vez definida la impresora, esta será validada dentro del archivo de impresoras para ratificar su existencia. En el caso de que no exista la impresora, el proceso emitirá un mensaje al usuario, y de lo contrario se procederá a tomar el registro de la librería respectiva (fonts o formas), lo procesará y generará un listado de registros que estará disponible para ser enviado a impresora.

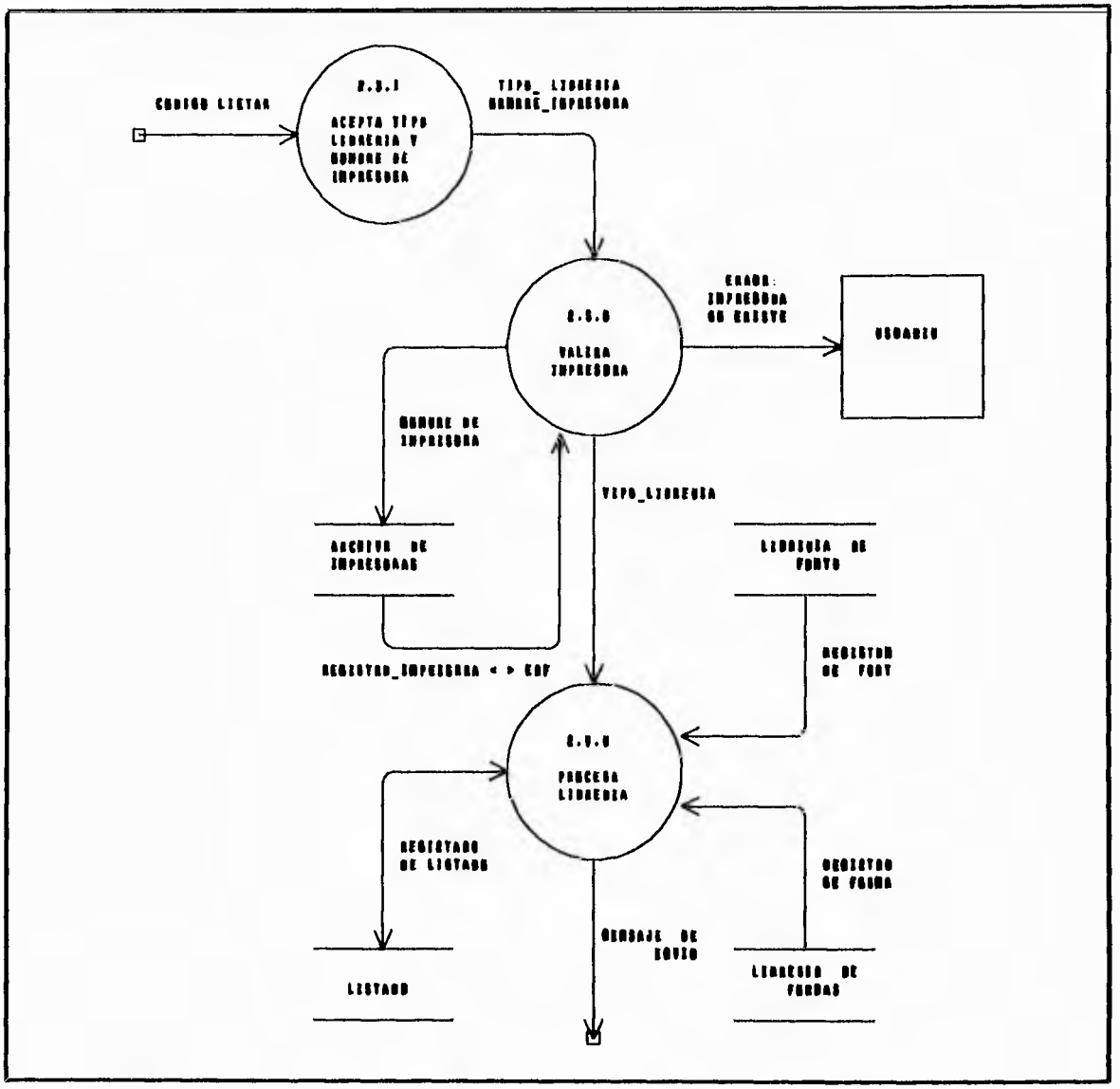


Fig. 2.4c Módulo de listado de fonts y formas

Generación de Salida.

El proceso para la generación de salida del módulo de interfase de utilería, se lleva a cabo a través del ruteo de: mensaje de terminación y mensaje de envío. Cuando el proceso detecta un mensaje de terminación, genera a través del ruteo un código de terminación, lo liga al despliegue de mensajes y emite al usuario el mensaje de salida. El procedimiento es similar para el caso de mensaje de envío, sólo que al generarse el código de envío, éste es emitido a impresión junto con los registros de listado. El código de envío será ligado con el despliegue de mensajes y emitirá al usuario el mensaje correspondiente al momento de terminar.

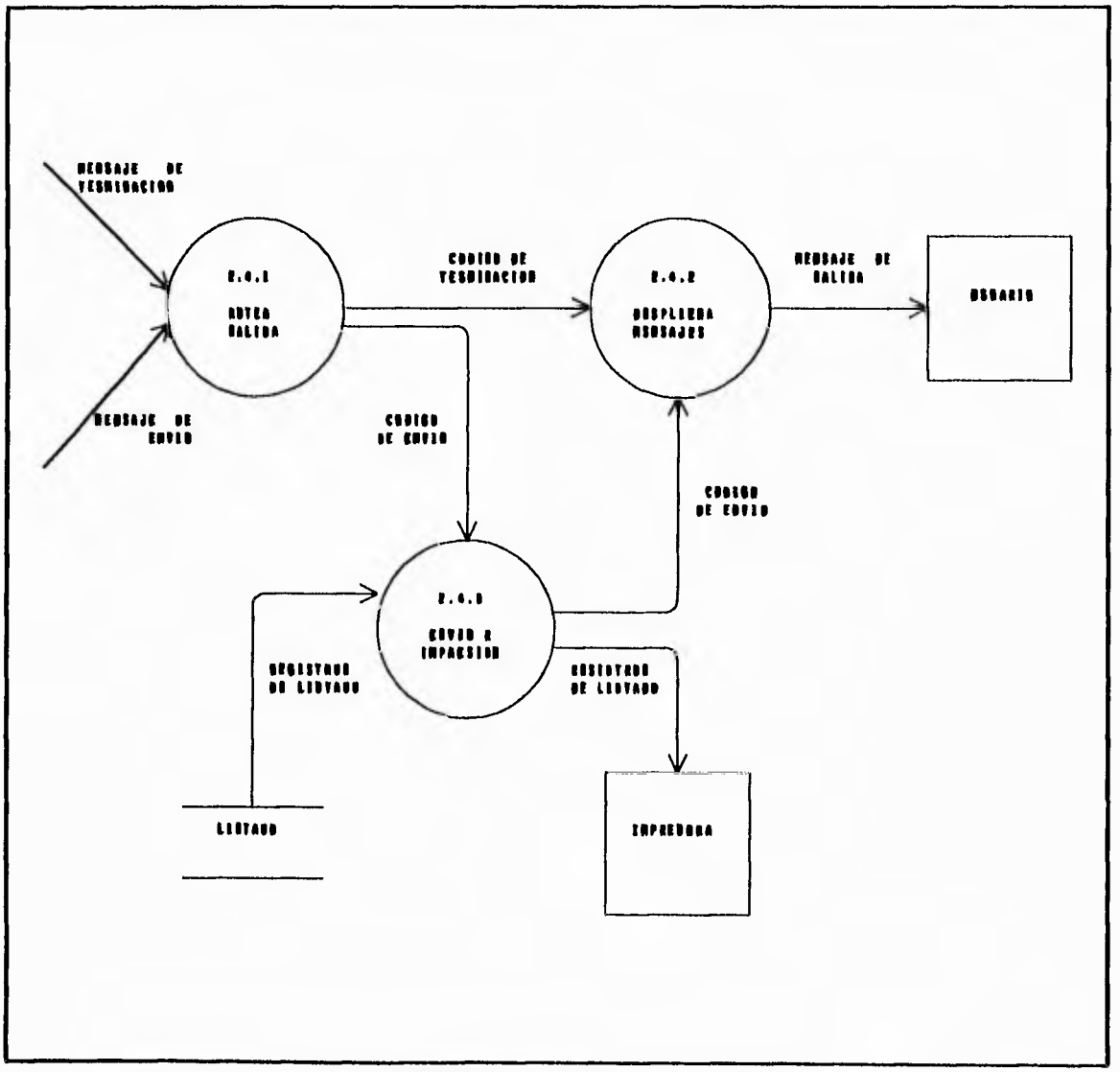


Fig. 2.4d Módulo de generación de salidas

A.3 Interfase de Impresión

El objetivo de la interfase de impresión es enviar la forma diseñada por el usuario a la impresora con o sin datos variables, además de interactuar con las librerías de fonts y formas respectivas. El siguiente diagrama muestra los procesos de esta interfase y la expansión de sus respectivos módulos se encuentran en las páginas posteriores.

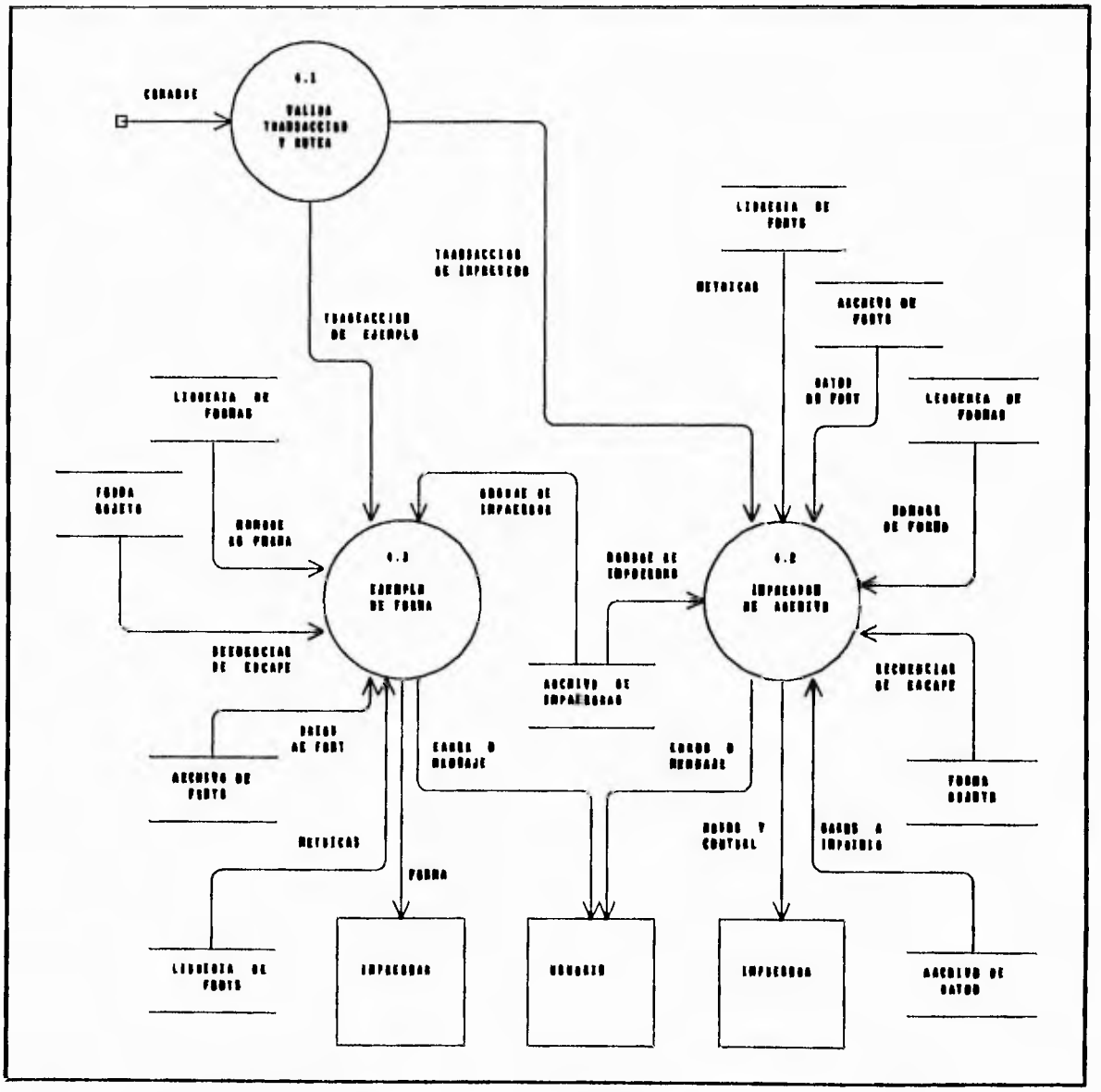


Fig. 2.6 Interfase de impresión

Valida transacción y rutea interfase de impresión

La función del módulo de valida transacción y rutea es tomar el comando de impresión y validar su sintaxis por medio de un scanner y un parser dedicados exclusivamente a esa función, después de esto se enviarán al usuario mensajes de error o se continuará con el ruteo de la transacción al módulo correspondiente.

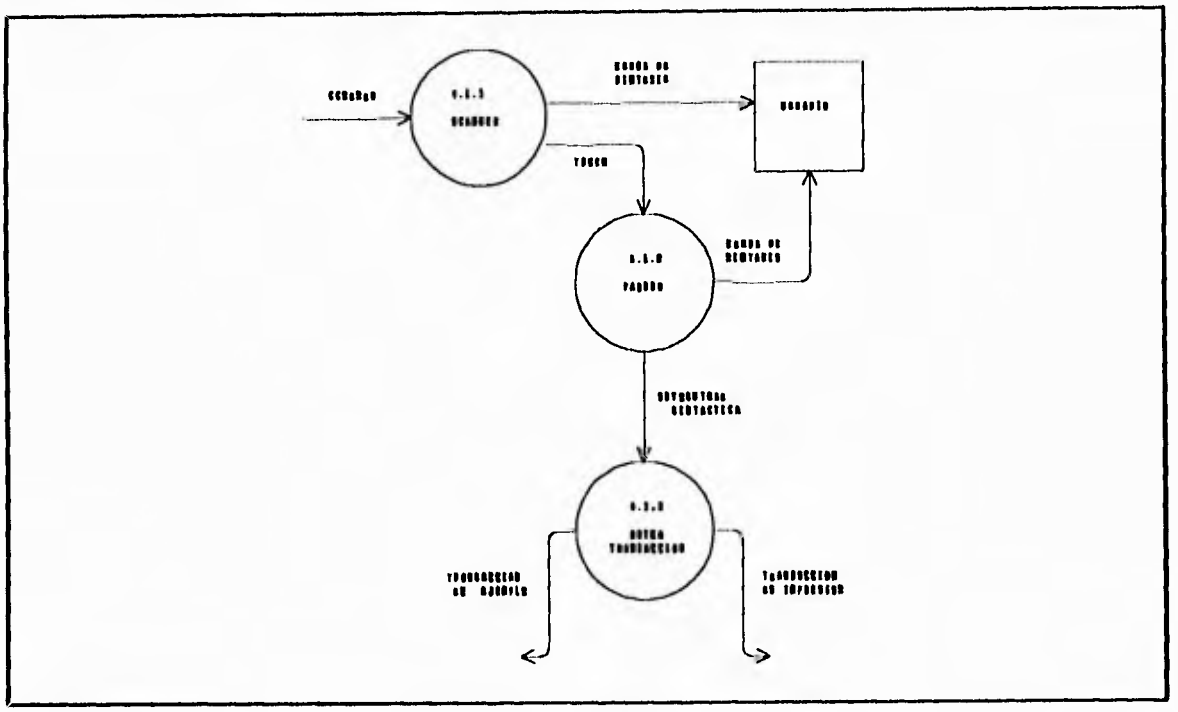


Fig. 2.6a Módulo de validación de transación y ruteo

Impresión de archivo

El objetivo del módulo de impresión de archivo será validar la existencia de la forma, después validar la existencia de la impresora a la cual se desea descargar la forma diseñada, en seguida se valida que exista el archivo físico que contiene los datos variables; en caso de detectar algún error durante los procesos antes descritos se generarán y enviarán los mensajes de error correspondientes.

En caso de no detectar ningún error, se procederá a descargar los datos de los fonts, la forma como secuencias de escape y los datos a imprimir a la impresora.

El módulo descarga fonts obtiene de la librería de formas los nombres de los fonts que se requieran para imprimir la forma deseada, conociendo los nombres de los fonts se obtendrá el nombre del archivo físico de cada font a descargar en la impresora .

El módulo descarga forma obtendrá de la librería de formas el nombre del archivo físico de la forma a descargar en la impresora, este archivo físico (forma objeto) contiene las secuencias de escape generadas por el compilador.

El módulo de inclusión de control se encargará de insertar los caracteres de control necesarios en los datos a imprimir y enviar un mensaje de terminación de proceso al usuario.

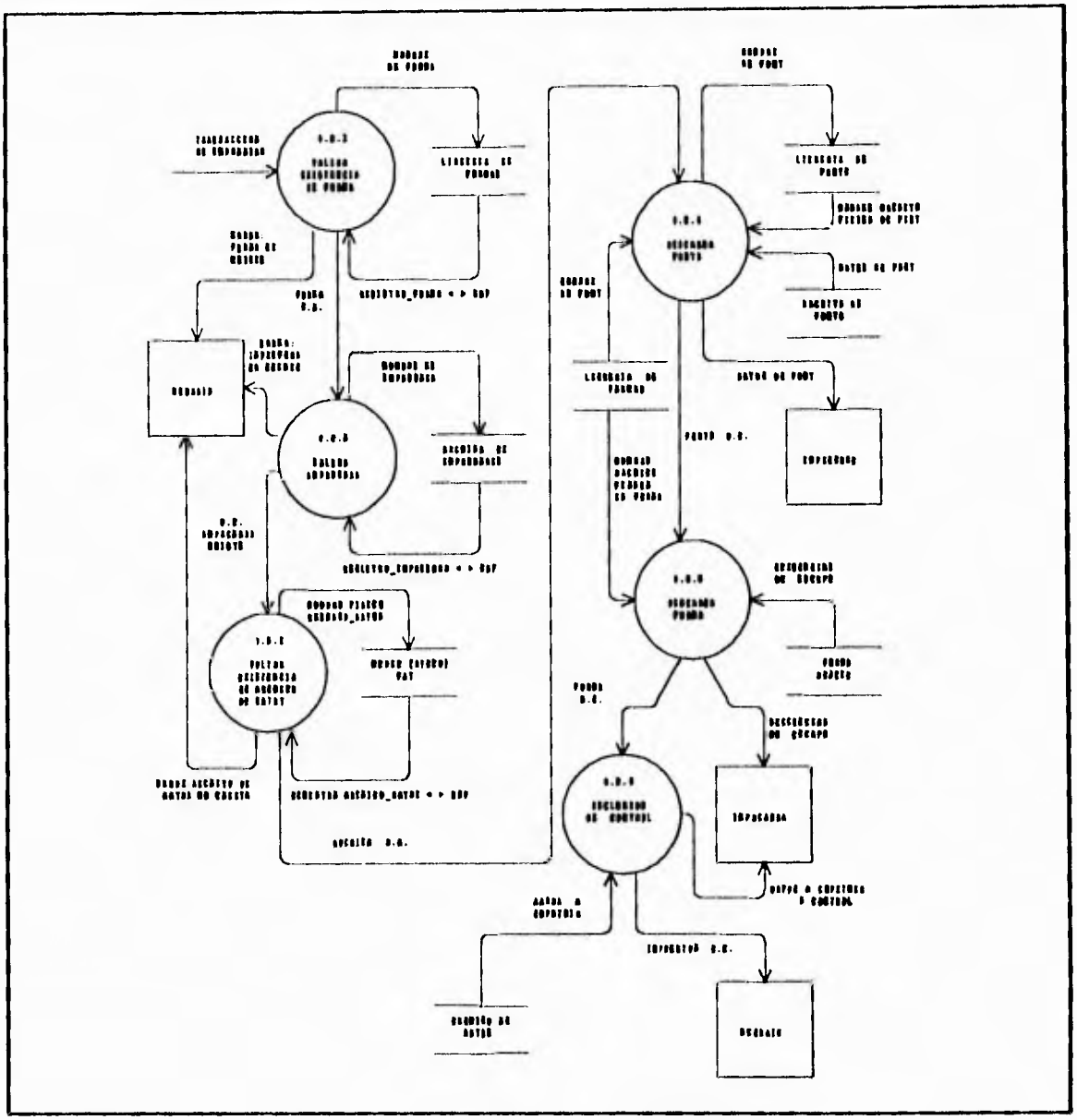


Fig. 2.6b Módulo de impresión de archivo

Inclusión de control.

Una vez que la forma se descargó satisfactoriamente a la impresora el módulo de inclusión de control se encargará de enviar línea por línea los datos a imprimir hasta encontrar el final del archivo; incluyendo el código de control form feed cada vez que se llene la página.

El archivo de impresión será impreso tantas veces como se haya indicado en el comando de impresión, finalmente se tendrá la posibilidad de borrar o mantener el archivo de impresión, lo cual se indicará con un parámetro en el comando de impresión.

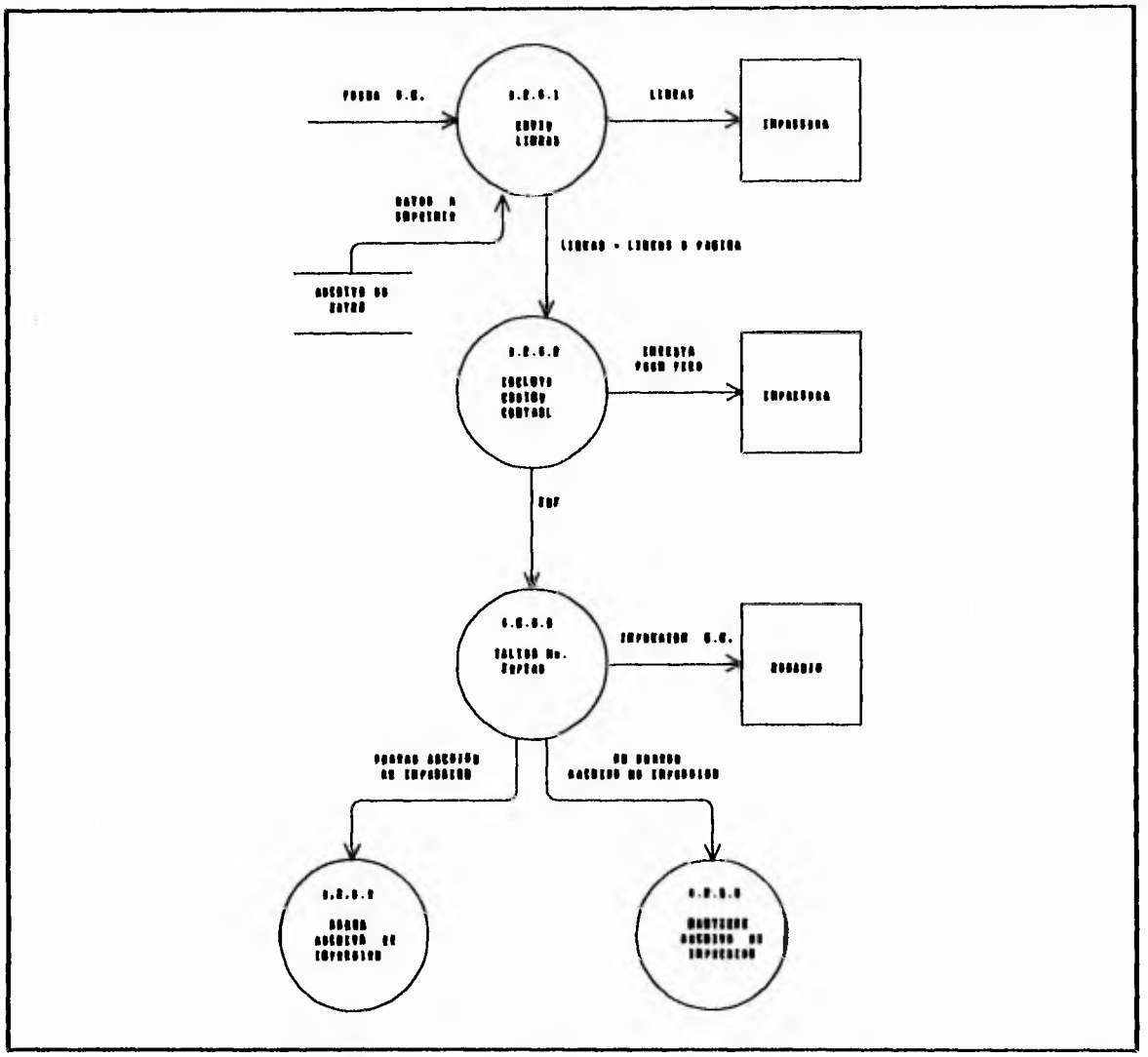


Fig. 2.6c módulo de inclusión de códigos de control

Ejemplo de forma

El objetivo del módulo de ejemplo de forma será validar la existencia de la forma, después validar la existencia de la impresora a la cual se desea descargar la forma diseñada, en caso de detectar algún error durante los procesos antes descritos se generarán y enviarán los mensajes de error correspondientes.

En caso de no detectar ningún error, se procederá a descargar los datos de los fonts y la forma como secuencias de escape a la impresora.

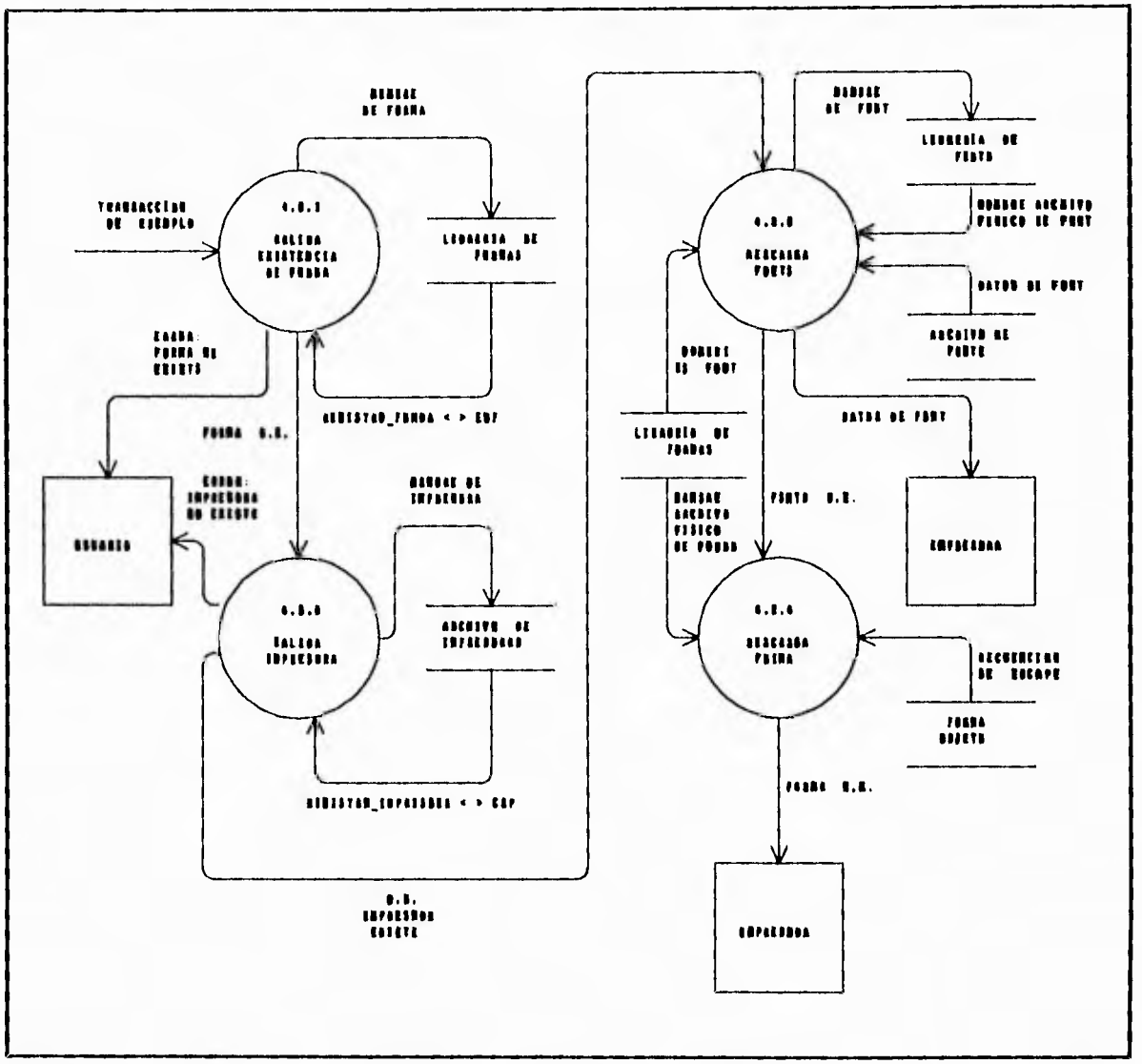


Fig. 2.6d Módulo de impresión de ejemplo de forma

El módulo descarga fonts obtiene de la librería de formas los nombres de los fonts que se requieran para imprimir la forma deseada, conociendo los nombres de los fonts se obtendrá el nombre del archivo físico de cada font a descargar en la impresora .

El módulo descarga forma obtendrá de la librería de formas el nombre del archivo físico de la forma a descargar en la impresora, este archivo físico (forma objeto) contiene las secuencias de escape generadas por el compilador.

BIBLIOGRAFIA

Aho, Alfred V.

"Compiladores: Principios, técnicas y herramientas"
Addison-Wesley Iberoamericana, Año1990, USA

Bohem, Barry W.

"Software Engineering"
IEEE Transactions on Computers, Dec, Año1976, USA

Brooks, Frederick P.

"The Mythical Man Month"
Addison-Wesley, Año1975, Philippines

Browning, Sally A.

"Unix System V Print Service Administration"
Prt & Prentice Hall, Año 1993, USA

Burch, John G. y otros

"Information Systems: Theory and Practice"
John Wiley & Sons, Año1979, USA

Ceballos, Fco. Javier

"Curso de programación con C : Microsoft C"
Macrobit, Año 1990, México

Ceballos, Fco. Javier

"Manual para Quick C2 Guia del Programador"
Macrobit, Año 1990, México

Clapp, Judith A.

"Designing Software for Maintainability Computer Design"
The Maganize of Based Systems, Vol 20, No 9, Sept, Año 1981, USA

Donovan, John J.

"Systems Programming"
McGraw Hill, Año 1972, Singapore

Fairley, Richard E.

"Software Engineering Concepts"
McGraw Hill, Año 1985, USA

Fischer, Le Blanc

"Crafting Compiler With C"
The Benjamin/Cummings Publishing Company, Inc., Año 1995, USA

Hewlett-Packard Company

"LaserJet III Printer Technical Reference Manual"
Hewlett-Packard Company, Año 1990, USA

Jamsa, Kris

"Microsoft C: secrets, shortcuts, and solutions"
Microsoft Press, Año 1989, USA

Jensen, Randall W

"Software Engineering"
Prentice-Hall, Año 1976, USA

Joyanes Aguilar Luis

"Microsoft C/C++ 7 Manual de Bolsillo"
McGraw Hill Interamericana, Año 1994, España

Kernighan, Brian W.

"El Lenguaje de Programación C"
Prentice-Hall, Año 1991, México

Kernighan, Brian W.

"El Entorno de la Programación UNIX"
Prentice-Hall, Año 1987, México

Lucas, Henry C.

"The Analysis, Design and Implementation of Information Systems"
McGraw Hill, Año 1976, USA

Maguire, Steve

"Writing Solid Code"
Microsoft Press, Año 1993, USA

Mak, Ronald

"Writing Compilers & Interpreters: An Applied Approach"
John Wiley & Sons, Año1991, USA

Parnas, David L.

"Designing Software for Ease of Extension and Contraction"
IEEE Transactions on Software Engineering, Vol SE-5, No 2, March, Año 1979, USA

Pressman, Roger S.

"Ingeniería del Software: Un enfoque práctico"
McGraw Hill, Año 1993, España

Roiz, Virginia

"Certificación de Calidad en Sistemas"
Tesis Profesional, Universidad del Valle de México, Año 1988, México

Sanders, Donald

"Informática: Presente y Futuro"
McGraw Hill, Año 1985, México

SCO UNIX Operating System

"User's Reference"
"System Administrator's Guide"
"System Administrator's Reference"
Año 1992, USA

Shooman, Martin L.

"Software Engineering: Design, Reliability, and Management"
McGraw Hill, Año 1985, Singapore

Templos, Alberto y otros

"Diseño y Aprovechamiento de Aplicaciones para Computadoras"
Tesis Profesional, Facultad de Ingeniería UNAM, Año 1985, México

Tenenbaum, Aaron M.

"Estructuras de Datos en C"
Prentice-Hall, Año 1993, México

Thomas, Rebecca

"Sistema Operativo UNIX: Guía de Usuario"
McGraw Hill, Año 1985, México

Tremblay, Jean-Paul

"The Theory and Practice of Compiler Writing"
McGraw Hill, Año 1985, Singapore

Tsun, S. Chow

"Tutorial Software Quality Assurance: A Practical Approach"
IEEE Transactions on Computers, pp. 13-20 Jan, Año 1985, USA

Xerox Corporation

"Xerox 4030 II Laser Printer; Xerox Escape Sequences Reference Manual"
Xerox Corporation, Año 1990, USA

Young, Michael J.

"Systems Programming in Microsoft C"
Sybex, Año 1991, USA