75
291.

UNIVERSIDAD NACIONAL
AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

Metodología para la conversión de sistemas en COBOL NOS/VE en equipos CYBER a RM/COBOL para UNIX en equipos HP 9000

TESIS PROFESIONAL
Que para obtener el título de

INGENIERO EN COMPUTACION

presenta

ERNESTO PAREDES ARREOLA

Dir. de tesis:
Ing. Víctor Hugo Amaya Galván

Codirector:
Ing. Lucila Patricia Arellano Mendoza

México, D.F. 1997

AGRADECIMIENTOS

Quiero expresar mi más sincero agradecimiento a todas las personas que colaboraron de una u otra manera en la realización de esta tesis:

Ing. Víctor Hugo Amaya Galván
Ing. Lucila Patricia Arellano Mendoza
Act. Arturo Beltrán Rangel
Act. José Luis Romero Caballero
Ing. Daniel Chacón Moreno

Gracias por sus sugerencias y por el tiempo dedicado a revisar este trabajo.

Ernesto Paredes Arreola

DEDICATORIAS

A mis padres

Luis Paredes Sánchez

Ernestina Arreola

Por todo el tiempo y esfuerzo que me han dedicado

A mis hermanos

Alma Delia

Patricia

Pilar

Citlali

Daniel

David

Javier

Roberto

Metodología para la conversión de sistemas en COBOL NOS/VE en equipos CYBER a RM/COBOL para UNIX en equipos HP 9000

Objetivos:

Diseñar la metodología para la conversión de archivos, programas y procedimientos de sistemas en COBOL NOS/VE en equipos CYBER a RM/COBOL para UNIX en equipos HP 9000, así como el desarrollo de herramientas que faciliten dicha conversión.

PRÓLOGO

El presente trabajo tiene el propósito de mostrar algunos de los problemas a los que se enfrentan los programadores cuando se tiene que cambiar un sistema desarrollado en COBOL en cierto sistema operativo (NOS/VE) a otro sistema operativo muy distinto (UNIX), así como la manera de resolverlos mediante una metodología.

Se hace uso de herramientas de UNIX como `sed`, `awk`, `YACC` y `LEX` para resolver varias de las tareas de la conversión.

El capítulo 1 es una introducción al problema. Se considera la situación actual del sistema y se analizan las diferencias entre NOS/VE y UNIX que afectan a la conversión, como diferencias en: la organización de archivos, el bloqueo de registros y de archivos, el manejo de pantallas y algunos otros aspectos. Se plantean las actividades necesarias para lograr la conversión.

El capítulo 2 considera la metodología usada para realizar la conversión y se usa el análisis estructurado como metodología de análisis de requerimientos para tratar el problema.

En el capítulo 3 se considera el diseño del sistema mediante el diseño estructurado y se convierten las especificaciones del sistema en cartas estructuradas.

En el capítulo 4 se analiza la forma de realizar la conversión de archivos, programas y pantallas y se considera el diseño de varias herramientas para realizar la conversión.

El capítulo 5 muestra los pasos que debe realizar un programador para realizar la conversión, usando las herramientas desarrolladas.

Al final de la tesis, se incluyó un glosario de términos y se presenta la bibliografía usada para desarrollar esta tesis.

INDICE

INDICE DE FIGURAS

1. INTRODUCCION.

1.1 Situación actual.

Se cuenta en Aseguradora Mexicana con sistemas desarrollados en COBOL NOS/VE en dos equipos CYBER 930. Estos equipos utilizan el sistema operativo NOS/VE (Network Operating System/Virtual Environment), el cual es un sistema operativo multiusuario y multitareas. Para el desarrollo de aplicaciones en COBOL se cuenta con COBOL NOS/VE.

COBOL NOS/VE es una implementación del estándar COBOL ANSI 1985, e incluye algunas extensiones al lenguaje.

Actualmente hay varios problemas que se tienen con estos equipos:

a) El tiempo de respuesta que tiene el equipo en cargar el programa principal y en efectuar algunas transacciones es muy lento, dependiendo de la carga del sistema.

b) Ya no se puede crecer en memoria (la capacidad máxima de memoria es de 64 Mb) y en otros aspectos que pudieran dar un mejor rendimiento al equipo.

c) El fabricante ya dejó de producir el equipo y tiene fallas frecuentes en disco.

Por lo tanto, surge la necesidad de convertir los sistemas desarrollados a otros equipos, en este caso, a equipos HP 9000 con sistema operativo HP-UX (versión de UNIX de HP) con RM/COBOL para UNIX. Esto se debe a que ya se cuenta con algunos equipos similares para dar atención a las sucursales.

Primeramente, se realizará la conversión del sistema de contabilidad.

1.2 Características de los equipos HP 9000

Los equipos HP 9000 tienen una arquitectura tipo RISC, con un formato fijo de instrucciones de 32 bits, y son escalables desde sistemas de escritorio hasta sistemas con la potencia de un mainframe.

El sistema operativo que manejan estos equipos es el HP-UX, el cual es el UNIX System V de AT&T más el valor añadido de HP, y tiene algunos rasgos del UNIX desarrollado en la Universidad de California en Berkeley.

Para el desarrollo de aplicaciones en COBOL, se cuenta con RM/COBOL-85 para UNIX. RM/COBOL-85, al igual que COBOL NOS/VE, es una implementación del estándar COBOL ANSI 1985 (designado X3.23-985), diseñado para un rendimiento óptimo y una portabilidad amplia a través de una diversidad extensa de equipos y sistemas operativos.

1.3 Diferencias entre COBOL NOS/VE y RM/COBOL 85 para UNIX

Aunque no hay un cambio en el lenguaje de programación (COBOL), hay diferencias significativas entre COBOL NOS/VE y RM/COBOL en:

1. El manejo de pantallas.
2. El manejo del bloqueo de registros y de archivos.
3. Utilerías propias (extensiones al lenguaje del proveedor).

En NOS/VE se usa la utilería Design Screen para diseñar las pantallas. En esta utilería se definen la pantalla, los letreros fijos y las variables, así como sus atributos (brillante, brillante protegido, video inverso, etc.) Cuando se ejecuta el programa interactivo, mediante algunas funciones se hace la llamada a la pantalla, y se presenta al usuario.

Aunque RM/COBOL cuenta con la sección *Screen Section* en la *Data División* del programa para describir pantallas y sus atributos asociados, se desea utilizar la utilería ASEMAP, un manejador y diseñador de pantallas propio en ambiente UNIX, ya que varios sistemas de sucursales de la empresa lo están usando, se conoce el formato en que éste maneja las pantallas y puede ser adaptado para que se asemeje al uso de pantallas que tienen los usuarios en los equipos CYBER.

La diferencia en el manejo de pantallas implica la conversión de todas las pantallas desarrolladas en el Design Screen de NOS/VE al formato leído por ASEMAP. El manejo de las pantallas en los programas interactivos es diferente, por lo que es necesario el desarrollo de herramientas que faciliten dicha conversión de programas.

También es necesario realizar una conversión de archivos, pues el formato en que se guardan los archivos es distinto entre NOS/VE y RM/COBOL.

Se considerarán a continuación las diferencias en la organización de archivos, el bloqueo de registros y de archivos, el manejo de pantallas y la ejecución de comandos del sistema operativo desde programas en COBOL, entre COBOL NOS/VE y RM/COBOL para poder analizar lo que implica la conversión de archivos y de programas.

1.3.1 Diferencias en la organización de archivos entre NOS/VE y RM/COBOL

Aunque COBOL NOS/VE y RM/COBOL soportan los mismos tres tipos de organización de archivos (secuenciales, relativos e indexados), hay diferencias en el formato en que se almacenan en NOS/VE y el que usa RM/COBOL. Por ejemplo, la forma de guardar un archivo indexado en NOS/VE es distinta a la de RM/COBOL; cada uno utiliza cierta información de control en un formato distinto, lo que no permite copiar directamente un archivo indexado de NOS/VE a RM/COBOL.

En ASEMEX se manejan solamente archivos secuenciales e indexados de longitud fija, por lo que nos concentraremos principalmente en éstos.

1.3.1.1 Organización de archivos en NOS/VE

Archivos secuenciales

Cada registro en un archivo secuencial es almacenado en el orden en el cual se escribió. Los registros en un archivo secuencial pueden ser de longitud fija o variable.

En cuanto a su estructura física, los registros son contiguos en un archivo de longitud fija.

Archivos indexados

En un archivo secuencial indexado, los registros son almacenados en orden ascendente por el valor el valor de la llave primaria Los registros pueden ser accesados secuencialmente o aleatoriamente.

Los registros pueden tener longitud fija o variable. Cada registro tiene una llave primaria única que permite un acceso rápido a cualquier registro en el archivo. También se pueden definir una o más llaves alternas para el registro.

Los registros son agrupados en bloques de datos. Los bloques de datos contienen registros y apuntadores internos a esos registros También se tienen bloques de índices.

1.3.1.2 Organización de archivos en RM/COBOL

Hay tres tipos de archivos soportados por RM/COBOL. Estos tipos de archivos son: secuenciales, relativos e indexados.

Archivos secuenciales

Los archivos secuenciales están organizados de tal modo que los registros son siempre leídos o escritos serialmente. Hay dos tipos de archivos secuenciales:

1. **Archivos secuenciales lineales.** Deben contener solo datos de texto ASCII. Cada registro lógico es variable en longitud y finaliza con un carácter salto de línea ('\n').
2. **Archivos secuenciales binarios.** Pueden contener cualquier tipo de datos. Los archivos secuenciales binarios pueden ser de longitud fija o variable.

Los archivos secuenciales binarios de longitud fija son grabados por RM/COBOL sin alguna estructura adicional; su longitud de registro suministra la información necesaria para manejar al archivo.

Archivos indexados

Los archivos indexados contienen datos y uno o más índices o llaves. Estos índices son usados para localizar datos en el archivo por medio de buscar el valor de la llave y así encontrar la ubicación en el archivo del registro correspondiente.

Cada registro en un archivo indexado contiene una llave primaria y puede contener varias llaves alternas.

RM/COBOL realiza una compresión de datos por omisión, tanto de registros de datos como de llaves.

Cada registro del archivo puede ser representado en un formato de datos comprimido o sin comprimir. La compresión de registros de datos reemplaza múltiples ocurrencias de los caracteres espacio, cero y nulo o caracteres repetidos con un carácter de compresión único. La compresión de registros de datos casi siempre mejora el rendimiento de los archivos indexados reduciendo el tamaño del archivo y permitiendo que más información sea leída en una transferencia física única.

En los archivos indexados se permite a las llaves estar comprimidas o sin comprimir. La compresión de llaves reemplaza los primeros caracteres de una llave que igualan a la llave precedente, y cualesquier caracteres espacio finales de una llave, con caracteres de compresión. Esto usualmente reduce la cantidad de espacio en disco ocupado por un archivo.

Debido al ahorro de espacio en disco y a la mejora en el rendimiento del acceso a archivos indexados, se decidió dejar habilitada la compresión de datos.

1.3.1.3 Tipos de datos numéricos en COBOL NOS/VE y en RM/COBOL, y sus diferencias.

Los datos numéricos se pueden representar en la computadora en dos modos básicos: como datos de caracteres o como datos numéricos.

En el modo de carácter, los datos numéricos se representan mediante esquemas de codificación alfanumérica, donde cada dígito decimal de un número se representa por medio de un grupo de 6 u 8 bits. Se pueden codificar mediante el código decimal de 6 bits BCD (Binary Coded Decimal), en ASCII o en EBCDIC

En el modo numérico se representan mediante la notación binaria de punto fijo, notación científica o de punto flotante, decimal empacado y decimal de zona.

En COBOL, los datos en modo carácter se describen en modo DISPLAY, mientras que los datos en modo numérico se describen en modo COMPUTATIONAL. DISPLAY es la condición que se da por omisión: todos los elementos de datos se supone que están en modo DISPLAY a menos que se declaren en modo COMPUTATIONAL.

Las computadoras efectúan las operaciones aritméticas con datos numéricos que están en modo numérico, no en caracteres. Si los datos numéricos se representan

en modo carácter, primero se deben convertir a modo numérico, antes de que se ejecuten las operaciones aritméticas.

Cada equipo usa una o más representaciones COMPUTATIONAL para sus operaciones aritméticas; algunas representaciones son: COMPUTATIONAL (abreviada COMP), BINARY y PACKED-DECIMAL. En general, la opción COMP es la que se espera que corresponda al modo computacional de una máquina dada.

Tanto en COBOL NOS/VE como en RM/COBOL, cada carácter de un elemento de datos DISPLAY es almacenado en código ASCII de 7 bits en un byte (8 bits) de memoria.

Se encontró que la forma de representación de los datos numéricos COMP en COBOL NOS/VE es distinta a la de RM/COBOL, mientras que la forma de representación de los datos en modo DISPLAY es la misma, por lo que es necesario convertir los datos numéricos en modo COMPUTATIONAL a modo DISPLAY en NOS/VE antes de transferirlos al equipo HP.

1.3.2 Diferencias en el bloqueo de registros y archivos entre COBOL NOS/VE y RM/COBOL

Tanto COBOL NOS/VE como RM/COBOL manejan el bloqueo de archivos y de registros. Este es un mecanismo que permite regular el acceso a los datos de un archivo indexado en un ambiente multiusuario.

El bloqueo de un archivo impide a otros usuarios tener cualquier acceso al archivo. El bloqueo de un registro prohibe el acceso a un único registro de un archivo de datos.

El bloqueo de archivos y registros se hace sobre la base de que el primer usuario que accede al archivo o registro impide el acceso a otros usuarios. Después de que el archivo o registro es actualizado, se permite el acceso a otros usuarios.

Por medio del bloqueo, se asegura la integridad de un archivo secuencial indexado que está siendo usado concurrentemente por más de un programa en ejecución. El bloqueo asegura que:

a) Las peticiones son procesadas en la secuencia en que fueron emitidas.
b) Cada operación es ejecutada en la versión mas actualizada del registro.

A continuación se examina el manejo del bloqueo de archivos y registros en COBOL NOS/VE y en RM/COBOL y sus diferencias.

1.3.2.1 Manejo del bloqueo de archivos y de registros en COBOL NOS/VE.

Más de un programa ejecutándose pueden leer o escribir concurrentemente un archivo secuencial indexado. Sin embargo, ésto es posible solo si es un archivo permanente y el modo de compartición del archivo lo permite. Por ejemplo, para la lectura compartida el modo de compartición debe incluir READ; para la escritura compartida, el modo de compartición debe ser WRITE

Modo de compartición	*Permite*
READ	Compartición en lectura
WRITE	Compartición en escritura

Si no se va a permitir el uso concurrente de un archivo, el archivo no debe permitir compartición (SHARE_MODE=NONE).

Concurrencia en la escritura

La concurrencia en la escritura se presenta cuando varios programas comparten un archivo y al menos uno de ellos puede escribir al archivo.

Se pueden usar diferentes clases de bloqueo para la concurrencia en la escritura:

a) Bloqueo por omisión.
a) Bloqueo implícito establecido por las rutinas de tiempo de ejecución de COBOL
b) Bloqueo explícito creado por las llamadas a rutinas de bloqueo.

En ASEMEX, algunos programas usan el bloqueo por omisión, aunque la mayoría de los programas interactivos usan el bloqueo implícito y el bloqueo explícito.

Bloqueo por omisión

Por omisión, cuando un programa COBOL abre un archivo compartido en el modo INPUT o I-O, una declaración READ al archivo solicita un bloqueo antes que lea el registro. El bloqueo permite a otros usuarios leer el registro, pero no hacer cualquier otra cosa con éste.

El bloqueo es solicitado con desbloqueo automatico, lo cual significa que el bloqueo es liberado cuando el programa ejecuta una operación de REWRITE o DELETE para el mismo registro, o cuando expira la condición de bloqueo, después de 60 segundos por omisión.

Cuando un programa usa el bloqueo por omisión, debe efectuar los siguientes pasos para reescribir o borrar un registro:

1. Abrir el archivo en el modo I-O.
2. Leer el registro a ser reescrito o borrado.
3. Reescribir o borrar el registro.

Bloqueo implícito

Si no vamos a utilizar el bloqueo por omisión suministrado por COBOL, podemos usar el bloqueo implícito, el cual nos permite cambiar las opciones del bloqueo usadas por siguientes declaraciones READ del archivo y se realiza mediante la rutina:

- CBP$SET_KEY_LOCKING_OPTIONS.

Rutina CBP$SET_KEY_LOCKING_OPTIONS

Esta rutina permite establecer las opciones del bloqueo usadas por siguientes declaraciones READ de ese archivo.

El formato de la llamada de esta rutina es:

CALL "CBP$SET_KEY_LOCKING_OPTIONS" USING *file-name lock wait unlock intent status*

A continuación se describen cada uno de los parámetros, y en el caso de las literales *lock*, *wait*, *unlock* e *intent*, se explican los valores que pueden tomar:

file-name: Nombre del archivo sobre el cual se establecerán las opciones.

lock: Indica si posteriores declaraciones READ del archivo solicitan bloqueos que persisten después de la lectura.

"LOCK": Posteriores declaraciones READ solicitan bloqueos que persisten.

"NO_LOCK": Subsiguientes declaraciones READ no solicitan bloqueos.

wait: Indica si el programa espera si otro programa tiene bloqueado el registro a ser leído.

"WAIT_FOR_LOCK": El programa espera hasta que el registro es disponible.

"NO_WAIT_FOR_LOCK": El programa no espera a un registro bloqueado.

unlock: Indica si posteriores declaraciones READ del archivo pedirán desbloqueo automático del bloqueo de COBOL.

"AUTOMATIC_UNLOCK": El bloqueo es automáticamente desbloqueado cuando el programa accesa otro registro (por omisión).

"WAIT_FOR_UNLOCK": El bloqueo persiste hasta que el bloqueo expire, el archivo sea cerrado o se ejecute la rutina CBP$UNLOCK_KEY para desbloquear explícitamente el registro.

intent: Especifica el tipo de bloqueo usado por subsiguientes declaraciones READ.

"EXCLUSIVE_ACCESS": No permite ningún acceso al registro por otros programas. Se debe usar cuando el programa mantiene un bloqueo sobre un registro que se va a borrar (DELETE) o reescribir (REWRITE) después de la declaración READ.

"PRESERVE_ACCESS_AND_CONTENT": Permite a otros programas leer el registro, pero previene de que escriban, reescriban o borren el registro, mientras el programa que mantiene bloqueado el registro tiene todas las formas de acceso disponibles.

Bloqueo explícito

Nos permite especificar directamente el bloqueo y desbloqueo de un registro o de todo el archivo y sólo se debería usar si no se piensa usar el bloqueo por omisión o implícito de COBOL. Para usar el bloqueo explícito se debe hacer lo siguiente:

- Usar la rutina CBP$SET_KEY_LOCKING_OPTIONS para terminar el bloqueo implícito de COBOL completamente. Se debe especificar el valor "NO_LOCK" a la literal *lock*, para que subsiguientes declaraciones READ del archivo no soliciten bloqueos que persisten después de la lectura
- Usar las rutinas CBP$LOCK_KEY y CBP$UNLOCK_KEY para bloquear y desbloquear un registro, respectivamente.
- Usar las rutinas CBP$LOCK_FILE y CBP$UNLOCK_FILE para bloquear y desbloquear todos los registros de un archivo, respectivamente

En los programas COBOL de ASEMEX se manejan los tres tipos de bloqueo para la concurrencia en la escritura

1.3.2.2 Manejo del bloqueo de archivos y de registros de RM/COBOL

En RM/COBOL se puede manejar el bloqueo tanto a nivel de archivos como a nivel de registros. A continuación se explica cómo se maneja cada uno de ellos.

Bloqueo de archivos

Un programa en ejecución puede negar el acceso concurrente a un archivo a otros programas en ejecución a través del bloqueo de archivos.

La frase LOCK puede ser especificada en la declaración OPEN, para indicar que la apertura del archivo es para obtener acceso exclusivo al archivo hasta que el archivo sea cerrado. La sintaxis de la declaración OPEN para archivos secuenciales e indexados es:

```
OPEN  { INPUT          file    [WITH LOCK ] }
      { OUTPUT         file    [WITH LOCK ] }
      { I-O            file    [WITH LOCK ] }
```

Cuando la declaración OPEN no especifica la frase WITH LOCK, la declaración OPEN abre el archivo en modo compartido, por omisión.

Bloqueo de registros

El bloqueo de registros solo ocurre cuando el archivo es abierto en el modo de entrada/salida compartido.

Cuando un archivo es abierto en el modo de entrada (INPUT), la ejecución de la declaración READ nunca intenta bloquear el registro accesado.

La frase LOCK puede ser especificada para controlar el bloqueo de registros para un archivo de entrada/salida compartido.

Si el modo de apertura es de entrada/salida al ejecutar la declaración READ, el registro obtenido es bloqueado hasta que el registro no sea reescrito o borrado; se puede especificar la frase WITH NO LOCK para suprimir el bloqueo de registro por omisión.

Cuando la declaración READ intenta bloquear un registro que ya está bloqueado por otro programa ejecutándose concurrentemente, RM/COBOL espera a que el otro programa desbloquee el registro antes de completar la ejecución de la declaración READ.

Por lo tanto, si el archivo es abierto en el modo de entrada/salida y un registro solamente será leído para consulta pero no va a ser actualizado, el registro debe ser leído con la frase WITH NO LOCK para suprimir el bloqueo del registro por omisión.

A continuación se muestra la sintaxis de la declaración READ para entrada/salida secuencial:

```
READ  file      [NEXT]  RECORD  [WITH  NO LOCK]
    [AT END  declaración-imperativa]  ...
```

Para la entrada/salida de archivos indexados, la sintaxis de la declaración READ tiene los siguientes formatos.

Formato 1:

```
READ  file      [NEXT]  RECORD  [WITH  NO LOCK]
    [AT END  declaración-imperativa]  ...
```

Formato 2:

```
READ  file  RECORD  [WITH  NO LOCK]
    [INVALID  KEY declaración-imperativa ]  ...
```

Otra declaración que puede usarse para controlar el bloqueo de registros es UNLOCK, la cual tiene la siguiente sintaxis:

```
    UNLOCK  file
```

La declaración UNLOCK hace disponible a otros programas el registro más recientemente leído y bloqueado de un archivo.

1.4 Diferencias en el manejo de pantallas entre COBOL NOS/VE y RM/COBOL.

1.4.1 Manejo de pantallas en programas interactivos de COBOL usando Screen Formatting de NOS/VE.

NOS/VE cuenta con al utilería Screen Formatting (SCRF) para diseñar y administrar pantallas en los programas de aplicación

Screen Formatting (SCRF) consiste de un conjunto de subrutinas y procedimientos con los cuales se puede diseñar una pantalla que el usuario de una aplicación ve y usa para interactuar con el programa La pantalla es desplegada y se regresa el control al programa COBOL cuando se presiona una tecla de función válida.

El programador diseña la pantalla con la utilería Design Screen, definiendo las constantes y variables de la pantalla y la pone en una libreria objeto de pantallas (SCREEN_LIBRARY).

Se administra la pantalla en el programa de aplicación incluyendo llamadas a procedimientos o subrutinas de COBOL para manejo de pantallas

Una pantalla contiene objetos de texto (letreros fijos) y objetos gráficos (una línea, un cuadro, etc.) a los que se les pueden definir atributos de despliegue (video inverso, protegido, etc.); también puede contener variables de entrada y/o salida, a las que se les puede asignar un atributo de despliegue.

Hay varios atributos disponibles en el diseñador que pueden asignarse a una variable de la pantalla, a cada uno de los cuales se le asignó un valor; estos atributos son:

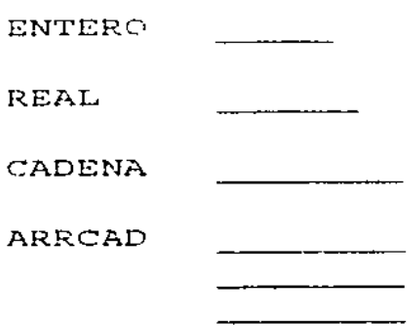
Valor	*Descripción del atributo*
01	Alta intensidad, modificable
02	Alta intensidad, protegido
03	Baja intensidad, modificable
04	Baja intensidad, protegido
05	Baja intensidad parpadeante, modificable
06	Baja intensidad parpadeante, protegido
07	Oculto, modificable
08	Oculto, protegido
09	Alta intensidad con video inverso, modificable1
10	Alta intensidad con video inverso, protegido
11	Baja intensidad con video inverso, modificable
12	Baja intensidad con video inverso, protegido
13	Baja intensidad con video inverso y parpadeante, modificable
14	Baja intensidad con video inverso y parpadeante, protegido
15	Oculto, con video inverso, modificable
16	Oculto, con video inverso, protegido

Las pantallas diseñadas con Design Screen siguen los siguientes estándares en ASEMEX:

1. El nombre de la pantalla debe tener el prefijo "PANTA_" seguido de un número de cinco dígitos.
2. Al definir una variable, su nombre debe comenzar con "CDM-", seguido del nombre de la variable y debe terminar con "-F": CDM-<*VARIABLE*>-F

Si el campo es una tabla, su nombre debe comenzar con "CDM-TAB-" seguido del nombre de la variable y debe terminar con "-F": CDM-TAB-<*VARIABLE*>-F.

Por ejemplo, se diseñó la pantalla PANTA_09998:



Cuando se crea la pantalla, SCRF genera un *registro de definición de pantalla* que contiene las definiciones de todas las variables de la pantalla. En el programa, se transfieren los valores de la pantalla a las variables de la pantalla en el programa, y viceversa, a través de este registro. El programador salva este archivo como un copy en una librería de fuentes llamada SCREEN_SOURCE_LIBRARY con la utilería SCU. El copy PANTA-09998 generado para esta pantalla es el siguiente:

```
01   PANTA-09998.
       03   CDM-TAB-ARRCAD-F OCCURS 3.
              05   CDM-ARRCAD-F PIC X(8).
       03   CDM-ENTERO-F PIC S9(18) COMP SYNC LEFT.
       03   CDM-REAL-F COMP-1.
       03   CDM-CADENA-F PIC X(8).
```

Se copia este registro de definición de la pantalla mediante la declaración COPY en el programa; el preprocesador inserta el archivo de la librería de fuentes en el texto del programa antes de compilar el programa.

En el programa COBOL se crean, por cada campo variable de la pantalla CDM-<*CAMPO*>-F, dos variables: una que almacena el valor del atributo del campo y otra que almacena el valor del campo. El nombre de las variables sigue la siguiente convención:

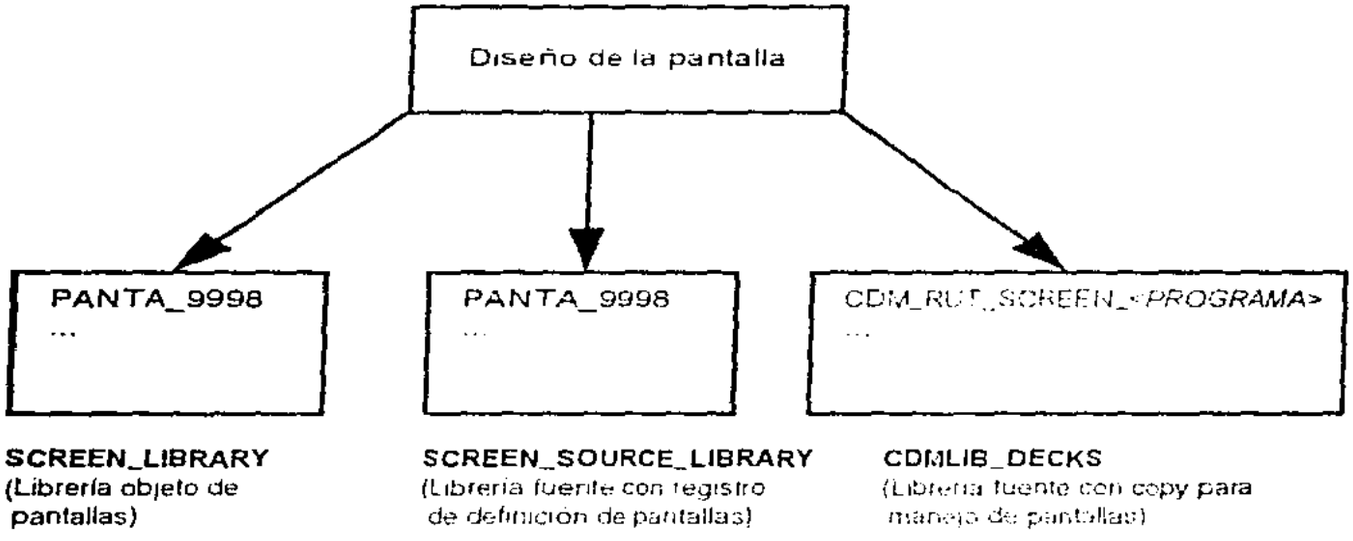
FACOF-<*CAMPO*>-P Almacena el valor del atributo del campo.
<*CAMPO*>-F Contiene el valor del campo.

Por ejemplo, el campo de la pantalla CDM-CADENA-F tiene asociadas dos variables en el programa en COBOL.

FACOF-CADENA-P: Variable que almacena el valor del atributo del campo.
CADENA-F: Variable que almacena el valor del campo.

Al diseñar la pantalla, también se genera un copy llamado CDM_RUT_SCREEN_<*PROGRAMA*>, donde <*PROGRAMA*> es el nombre del programa que utiliza la pantalla, en el cual se encuentran las rutinas para el manejo de todas las pantallas que usa el programa. Este copy se guarda en una librería de fuentes llamada CDMLIB_DECKS.

A continuación se presenta un diagrama que muestra los archivos generados al diseñar una pantalla:



En el copy CDM_RUT_SCREEN_<*PROGRAMA*> se efectúan los siguientes pasos para cada pantalla:

1. Mover a cada campo de la pantalla el valor que se desea que aparezca cuando se presente la pantalla, el cual lo tiene la variable <CAMPO>-F. Por ejemplo, para el campo CDM-ENTERO-F se tendría:

```
IF FACOF-ENTERO-P = 7 OR 9
   MOVE ZEROES  TO CDM-ENTERO-F
```

```
ELSE
    MOVE ENTERO-F TO CDM-ENTERO-F
END-IF
```

Con ésto se asigna cero al campo de la pantalla CDM-ENTERO-F si su atributo es oculto modificable o protegido (7 u 8), de otra manera se le asigna el valor de la variable ENTERO-F, la cual contiene el valor del campo de la pantalla en el programa.

2. Llamar a la rutina "FDP$XREPLACE_RECORD", la cual transfiere los valores del registro de definición de la pantalla al buffer de SCRF.

3. Llamar a la rutina "FDP$SET_CURSOR_POSITION" para establecer el atributo de despligue para el campo, cuyo valor lo contiene la variable FACOF-<CAMPO>-P.

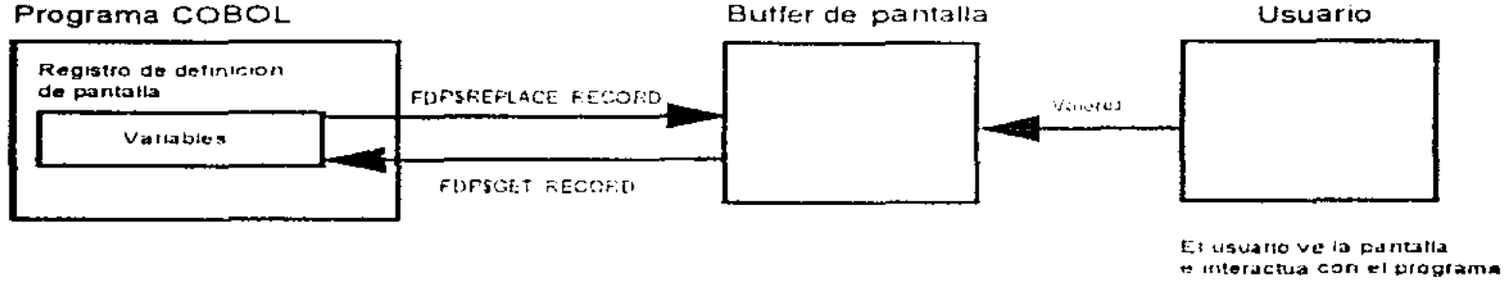
4. Presentar la pantalla hasta que no tenga errores en la captura o se presione alguna tecla de función permitida para salirse de la pantalla.

Para ésto se llamará primero a la rutina "FDP$XREAD_FORM", la cual presenta la pantalla en el monitor y después la rutina "FDP$GET_NEXT_EVENT", la cual regresa el valor de la tecla de función con la cual se salieron de la pantalla.

Después se llamará a la rutina "FDP$XGET_RECORD", la cual trae los datos capturados en la pantalla al registro de definición de la pantalla en el programa..

En caso de que se mueva algún dato inválido en un campo numérico, se mandará un mensaje y se volverá a presentar la pantalla, para que el usuario corrija los errores.

Esto se ilustra en la siguiente figura:



5. Al final, se tendrán todas las instrucciones MOVE para copiar el contenido del campo de la pantalla a la variable <CAMPO>-F, que contiene el valor del campo. Por ejemplo:

```
MOVE CDM-ENTERO-F TO ENTERO-F
```

1.4.2 Manejo de pantallas en COBOL usando la utilería ASEMAP para UNIX

En ASEMAP, por cada campo de la pantalla se tienen dos variables: una que almacena el valor del atributo del campo y otra que almacena el valor del campo. El nombre de las variables sigue la siguiente estructura:

VARIABLE-P Almacena el valor del atributo del campo

VARIABLE-F Almacena el valor del campo

donde:

VARIABLE: Es el nombre que se le dió al campo de la pantalla dentro del diseñador de pantallas.

Se diseñó con ASEMAP la misma pantalla que en el punto anterior (PANTA_09998).

El diseñador genera el copy PANT9998.scrn de la siguiente manera:

```
*****************************************************************
*                          SCREEN PANT9998                      *
*****************************************************************
 01  SCREEN-PANT9998.
  03 FILLER PIC X(118) VALUE
     "PANT9998 scrn-&7X-' ENTERO-I.REAL-+.CADENA-~.ARRCAD-###J#G??
-    "'7((#11#N??17)(%11##??+7++#AN#,??-7++#AN#5??.7++#AN#>??/7+".
  03 FILLER PIC X(004) VALUE
     "+#AN".
  02 FILLER PIC X VALUE LOW-VALUES.
*****************************************************************
 01  VARIABLES-PANT9998.
     03 VAR-ALPHA-PANT9998.
        05 CADENA-P                          PIC X(01)  VALUE "3".
        05 CADENA-F                          PIC X(08)  VALUE SPACES.
        05 ARRCAD-F-INI
           07 ARRCAD-F-VEC OCCURS 03 TIMES.
              09 ARRCAD-P                    PIC X(01) VALUE "3".
              09 ARRCAD-F                    PIC X(08) VALUE SPACES.
     03 VAR-NUM-PANT9998 SIGN TRAILING SEPARATE.
        05 ENTERO-P                          PIC X(01)  VALUE "3".
        05 ENTERO-F                          PIC S9(05) VALUE ZEROES.
        05 REAL-P                            PIC X(01)  VALUE "3".
        05 REAL-F                            PIC S9(03)V9(02)
                                             VALUE ZEROES .
     03 FILLER PIC X VALUE LOW-VALUES.
*****************************************************************
```

Para el campo de la pantalla *CADENA* se tienen dos variables:
CADENA-P: Almacena el valor del atributo del campo
CADENA-F: Almacena el valor del campo

Antes de llamar al manejador, el programa debe mover a cada variable de entrada y/o salida un atributo válido para que ese dato sea protegido o modificable, parpadee o permanezca fijo, etc. ASEMAP cuenta con 18 atributos diferentes. También se deberá mover el valor que se desea que aparezca al presentar la pantalla. Por ejemplo:

```
MOVE "1"      TO CADENA-I      Atributo
MOVE SPACES TO CADENA-F      Valor
```

Para llamar al manejador de pantallas en un programa COBOL se debe poner la instrucción:

CALL "/usr/easemap/screen" USING PF-KEY *<pantalla>* *<variables>*

Donde:

screen es el nombre del programa objeto que "maneja" la pantalla.

PF-KEY es el nombre de la variable donde se devuelve el valor con el cual se salió de la pantalla. Se regresa el control al programa en RM/COBOL al oprimir la tecla <Esc> y dos números o al teclear el último campo de la pantalla. Los valores devueltos son: 0 para <Esc> 00, 1 para <Esc> 01, etc. Tiene la siguiente definición:

```
                    PF-KEY                  PIC ...  VALUE ZEROES.
```

pantalla es la primera parte del formato de pantalla, la cual contiene las descripciones de las variables y constantes.

variables son los valores de las variables dentro del programa COBOL.

La pantalla anterior se llamaría de la siguiente manera:

CALL "/usr/easemap/screen" USING PF-KEY SCREEN-PANT9998 VARIABLES-PANT9998

El manejador despliega la pantalla, con sus títulos o constantes que se especificaron en el diseño de la pantalla, y los datos de entrada/salida que corresponden con el conjunto de variables. El manejador se encarga de manejar de forma automática la terminal, despliegue, aceptación, edición, validación y ajuste de los datos. Se regresa el control al programa en RM/COBOL al oprimir la tecla <Esc> y dos números o al teclear el último campo de la pantalla

Como se puede ver, el manejo de pantallas en los programas interactivos es muy distinto en los dos ambientes.

1.5 Diferencia en la ejecución de comandos de sistema operativo entre COBOL NOS/VE y RM/COBOL

En COBOL NOS/VE se ejecutan comandos de sistema operativo desde el programa COBOL llamando a la rutina "$SCAN_COMMAND_LINE". En RM/COBOL se ejecuta un comando de sistema operativo llamando a la rutina "SYSTEM".

1.6 Diferencias entre SCL de NOS/VE y Korn shell de UNIX

NOS/VE cuenta con SCL (System Command Language), el cual es tanto el lenguaje de control de trabajos de NOS/VE como un lenguaje de programación de alto nivel. Todos los comandos son procesados por el intérprete SCL, el cual es el responsable de leer los comandos desde archivos o terminales interactivas y de procesarlos.

UNIX cuenta con varios intérpretes de comandos o shells, como C shell, Bourne shell y Korn shell. Se usará Korn shell para los sistemas en UNIX debido a que es el más rápido y ofrece un conjunto de comandos más poderoso que los otros, con capacidades como historia de comandos, uso de alias, completación de nombres de archivos, edición de comandos previos, etc.

Los comandos ejecutados en SCL son muy distintos de los manejados en Korn shell de UNIX, por lo tanto es necesario convertir los procedimientos realizados en SCL a scripts en Korn shell de UNIX.

1.7 Planteamiento de objetivos.

De lo considerado anteriormente, se puede ver que se necesitarán realizar las siguientes actividades. Conversión de:

a) Archivos secuenciales e indexados.
b) Programas en COBOL batch e interactivos.
c) Pantallas.
d) Procedimientos en SCL de NOS/VE a scripts en Korn shell de UNIX.
d) Procedimientos para la transferencia de archivos y ejecución de comandos remotos, usando rcp y remsh (servicios Berkeley).

Una vez definido ésto, se contemplará el diseño de procedimientos y utilerías que faciliten dicha conversión.

La metodología y los procedimientos desarrollados están destinados a los desarrolladores de sistemas que deseen convertir sistemas desarrollados en COBOL NOS/VE a RM/COBOL para UNIX.

2. Metodología para la conversión.

2.1 Introducción.

Para efectuar la conversión de los sistemas en equipos CYBER a equipos HP es necesario efectuar los siguientes pasos:

1. Realizar la conversión de archivos.

Para ésto, primeramente es necesario analizar la estructura física de los archivos en NOS/VE y en RM/COBOL; luego se convertirán los archivos secuenciales e indexados de NOS/VE a un formato común que puedan leer ambos sistemas (código ASCII) y se transferirán al equipo HP para realizar la conversión del archivo secuencial ASCII a un archivo indexado en RM/COBOL. Se debe seleccionar una forma de transferir los archivos entre ambos sistemas.

Para verificar que la conversión de archivos fue correcta, se calculará la suma de los campos *que sean importes* sobre todos los registros del archivo, lo que llamaremos cifra de control sobre un campo, y se comprobara que tales cifras sean las mismas en NOS/VE y en RM/COBOL.

2. Conversión de los programas batch.

Se iniciará por este tipo de programas porque no manejan pantallas. Hay que identificar los patrones más comunes a cambiar para hacer un programa que los realice, como: manejo del bloqueo de archivos y de registros, conversión de comandos SCL a su equivalente en Korn shell.

Para verificar que se comportan igual al sistema anterior, se correrán los mismos programas en NOS/VE con los mismos archivos y se hará lo mismo en el equipo HP; hay que comprobar que el sistema convertido entrega los mismos resultados en los reportes que el sistema actual.

3. Conversión de pantallas.

Se obtendrá una lista de todas las pantallas usadas por los programas interactivos.

Hay que revisar el formato en que se guardan en NOS/VE y la manera de extraer esa información. Después se diseñará un programa que efectúe esa tarea y, finalmente, se convertirán al nuevo formato de ASEMAP

4. Conversión de programas interactivos.

Se debe analizar el manejo de pantallas que se efectua en SCRF y cambiarlo por el nuevo manejo de pantallas en ASEMAP.

Al final se deben correr los programas interactivos en el equipo HP y verificar que entregan los mismos resultados que en NOS/VE. Se analizarán algunos cambios que se pueden hacer a ASEMAP para que se comporte el sistema de la misma

manera que en SCRF, como por ejemplo, el uso de teclas de función para regresar el control al programa COBOL.

Las metodologías del análisis de requerimientos combinan procedimientos sistemáticos con una notación única para analizar los dominios de información y funcional de un problema de software. El análisis de requerimientos define lo que el sistema deberá ser capaz de realizar, especifica la función y comportamiento de los programas, establece la precisión con la que deberán ser alcanzados los objetivos y describe el sistema desde el punto de vista del usuario. Se representa el dominio de la información que será tratada por el programa.

Una de las metodologías que se pueden usar son los métodos de análisis orientados al flujo de datos, como el análisis estructurado.

El análisis estructurado es el uso de herramientas como diagramas de flujo de datos, diccionario de datos y la descripción de miniespecificaciones con la finalidad de construir la especificación estructurada del sistema.

Para llevar a cabo esta especificación, la metodología se apoya en las siguientes herramientas:

a) Los diagramas de flujo de datos (DFD), una técnica gráfica que nos permite particionar los requerimientos en base a flujos de datos y a procesos.
b) El diccionario de datos, que nos permite definir los flujos de datos.
c) La descripcion de miniespecificaciones, que nos permite definir los procesos.

2.2 Diagramas de flujo de datos.

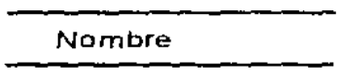
Un diagrama de flujo de datos (DFD), es una técnica gráfica que describe el flujo de la información y las transformaciones que se aplican a los datos, conforme se mueven de la entrada a la salida.

Se utiliza la notación Yourdon para describir el diagrama de flujo de datos.

La simbología de un DFD se ilustra a continuación:

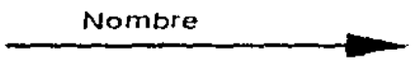
Entidad externa

Almacén de datos

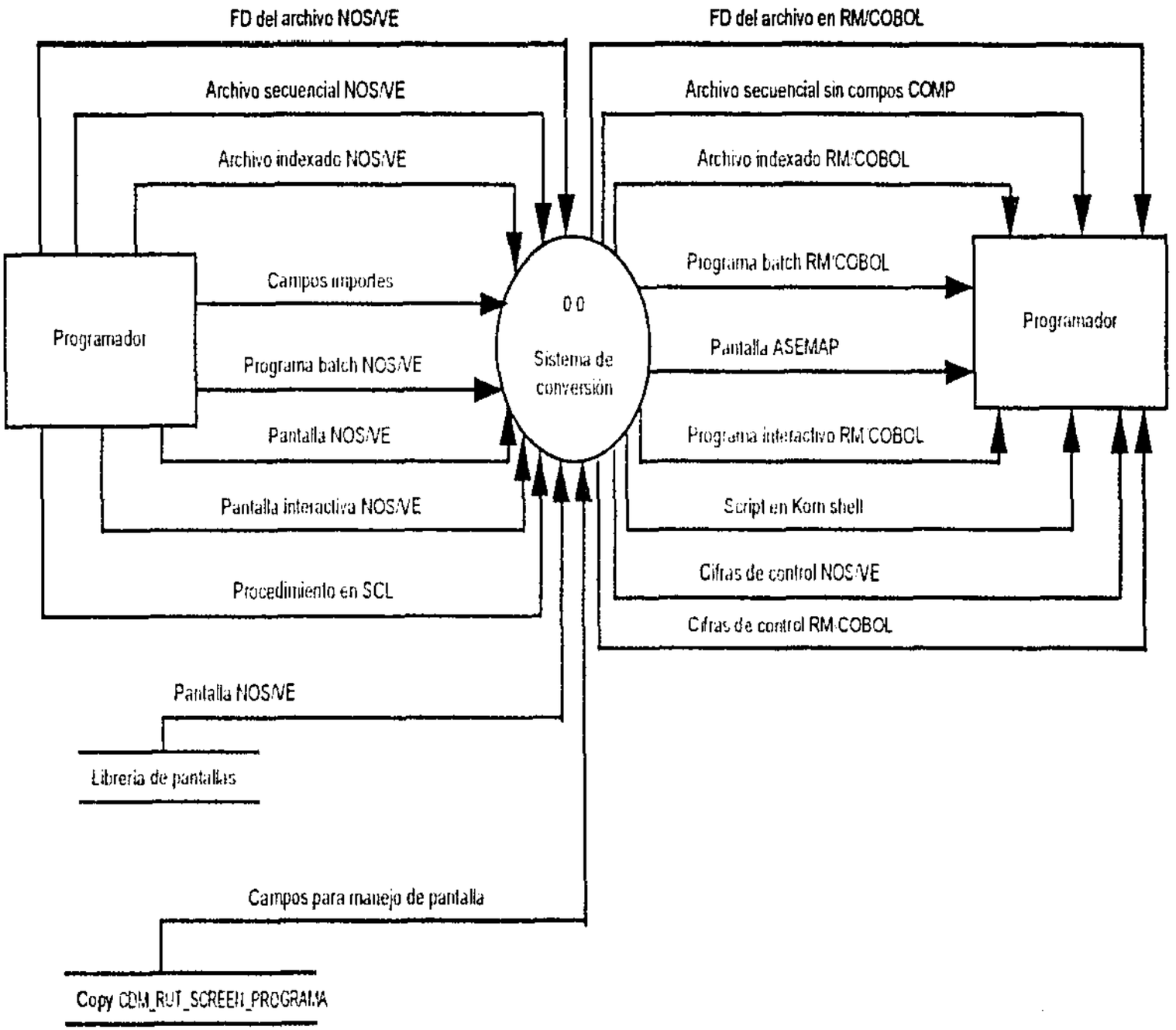


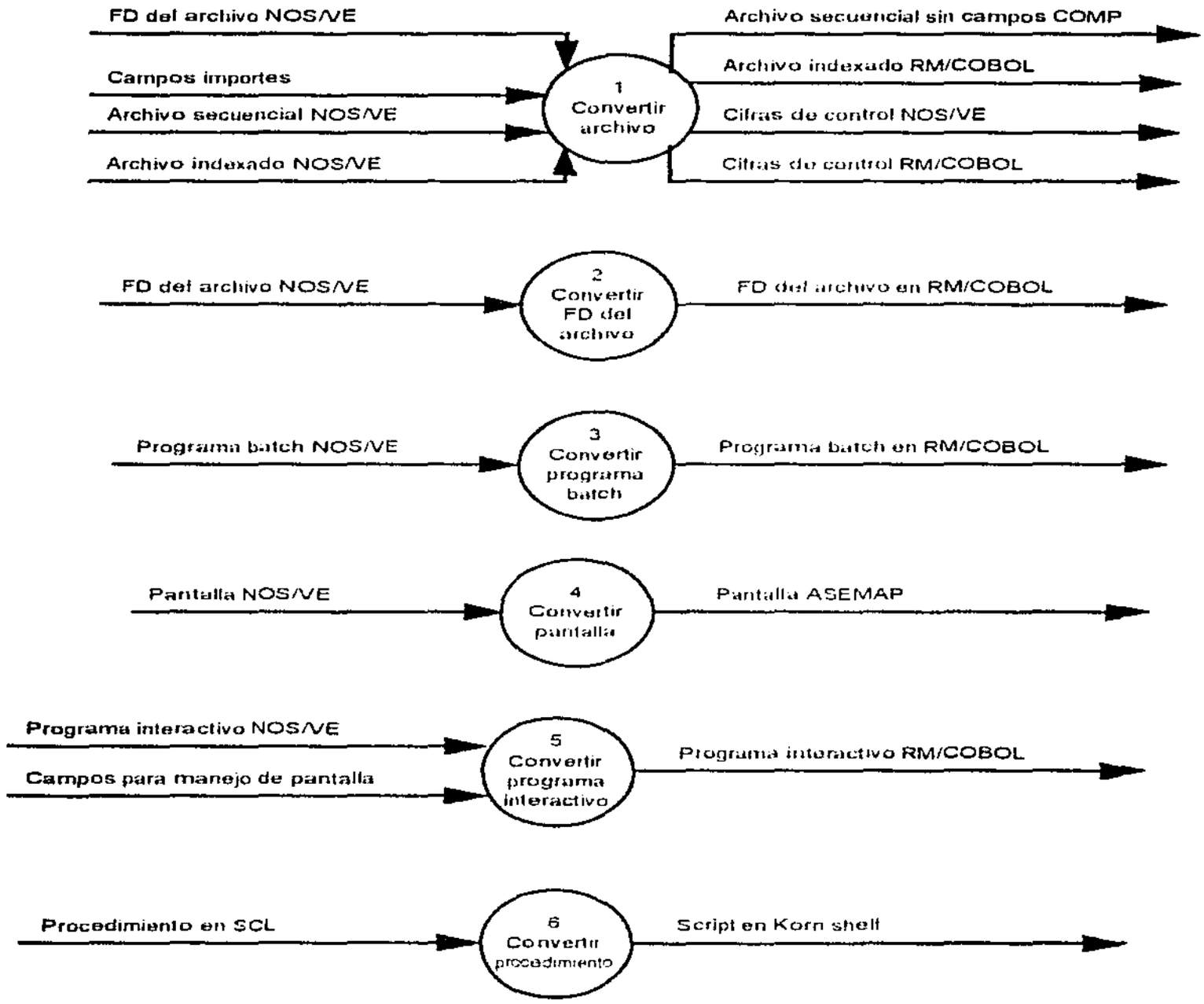
Proceso

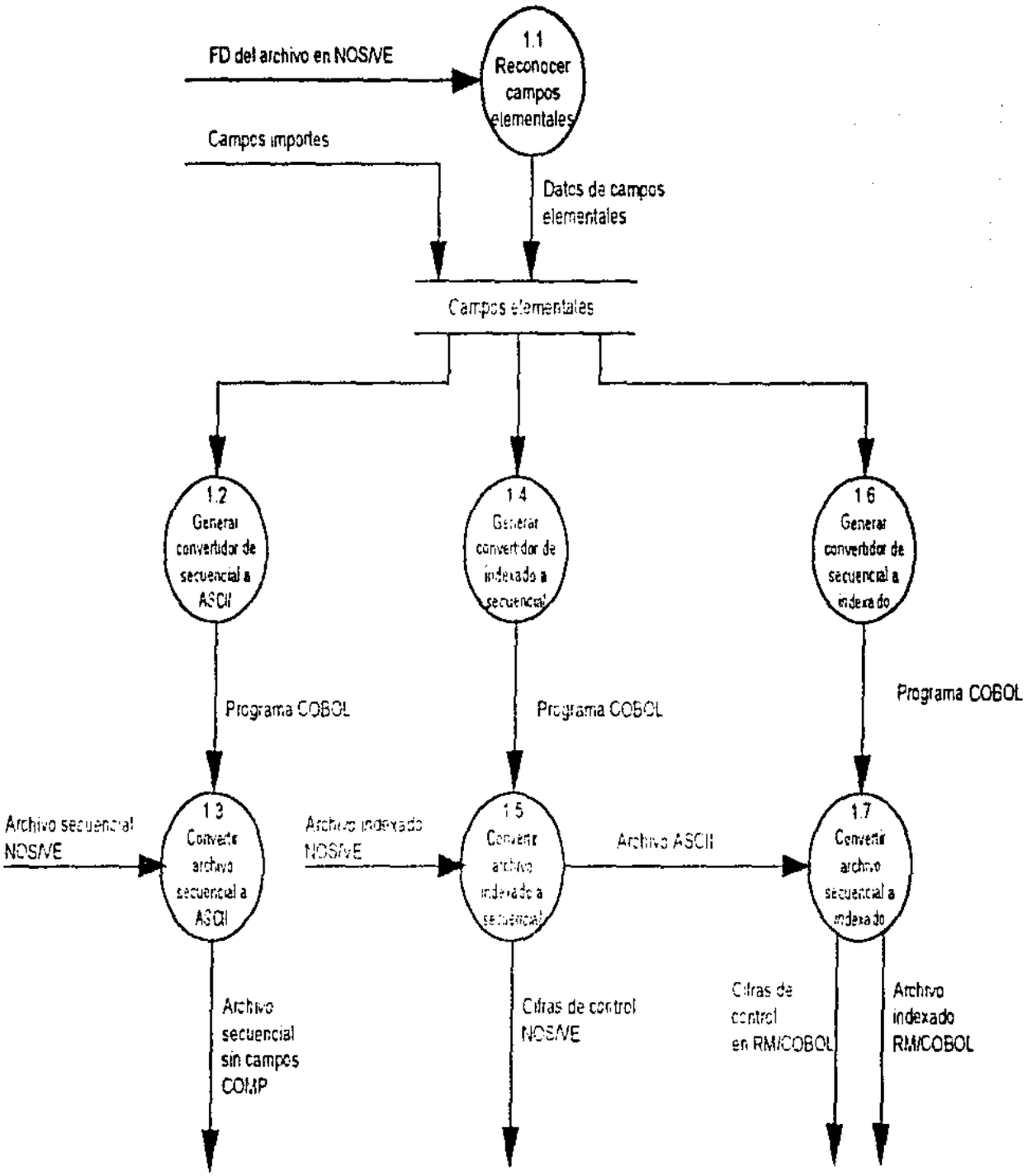
Flujo de datos

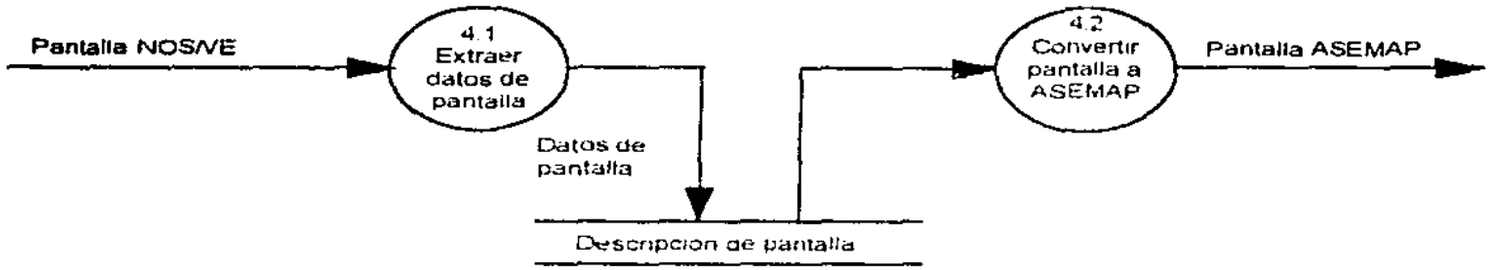


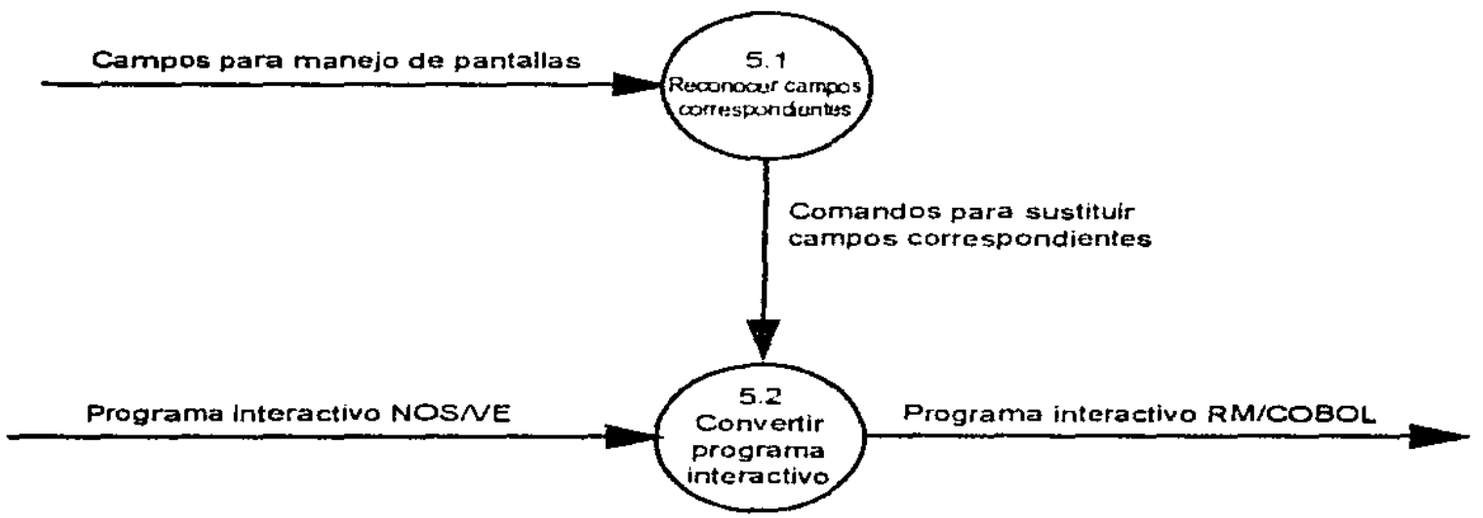
A continuación se presentan los diagramas de flujo de datos que son el resultado del análisis del sistema.


FD del archivo NOS/VE
FD del archivo en RM/COBOL
Archivo secuencial NOS/VE
Archivo indexado NOS/VE
Archivo secuencial sin compos COMP
Archivo indexado RM/COBOL
Programador
Campos importes
Programa batch NOS/VE
Pantalla NOS/VE
Pantalla interactiva NOS/VE
Procedimiento en SCL
0 0
Sistema de conversión
Programa batch RM/COBOL
Pantalla ASEMAP
Programa interactivo RM/COBOL
Script en Korn shell
Cifras de control NOS/VE
Cifras de control RM/COBOL
Programador
Pantalla NOS/VE
Libreria de pantallas
Campos para manejo de pantalla
Copy CDM_RUT_SCREEN_PROGRAMA


FD del archivo NOS/VE
Campos importes
Archivo secuencial NOS/VE
Archivo indexado NOS/VE
1
Convertir
archivo
Archivo secuencial sin campos COMP
Archivo indexado RM/COBOL
Cifras de control NOS/VE
Cifras de control RM/COBOL
FD del archivo NOS/VE
2
Convertir
FD del
archivo
FD del archivo en RM/COBOL
Programa batch NOS/VE
3
Convertir
programa
batch
Programa batch en RM/COBOL
Pantalla NOS/VE
4
Convertir
pantalla
Pantalla ASEMAP
Programa interactivo NOS/VE
Campos para manejo de pantalla
5
Convertir
programa
interactivo
Programa interactivo RM/COBOL
Procedimiento en SCL
6
Convertir
procedimiento
Script en Korn shell


FD del archivo en NOS/VE
1.1
Reconocer campos elementales
Campos importes
Datos de campos elementales
Campos elementales
1.2
Generar convertidor de secuencial a ASCII
1.4
Generar convertidor de indexado a secuencial
1.6
Generar convertidor de secuencial a indexado
Programa COBOL
Programa COBOL
Programa COBOL
Archivo secuencial NOS/VE
1.3
Convertir archivo secuencial a ASCII
Archivo indexado NOS/VE
1.5
Convertir archivo indexado a secuencial
Archivo ASCII
1.7
Convertir archivo secuencial a indexado
Archivo secuencial sin campos COMP
Cifras de control NOS/VE
Cifras de control en RM/COBOL
Archivo indexado RM/COBOL


Pantalla NOS/VE
4.1
Extraer
datos de
pantalla
Datos de
pantalla
Descripcion de pantalla
4.2
Convertir
pantalla a
ASEMAP
Pantalla ASEMAP


Campos para manejo de pantallas
5.1
Reconocer campos
correspondientes
Comandos para sustituir
campos correspondientes
Programa Interactivo NOS/VE
5.2
Convertir
programa
interactivo
Programa interactivo RM/COBOL

2.3 Diccionario de datos.

El diccionario de datos contiene las definiciones de todos los flujos de datos mencionados el DFD. Los datos compuestos se definen en términos de sus componentes; los datos elementales (datos que no pueden ser divididos) se definen en términos del significado de cada uno de los valores que puede asumir.

El diccionario de datos está compuesto de definiciones de flujo de datos, archivos (datos almacenados) y datos usados en los procesos.

La notación de un diccionario de datos se ilustra a continuación:

Construcción de datos	*Notación*	*Significado*
	=	Está compuesto de
Secuencia	+	y
Selección	[|]	Uno u otro
Repetición	$\{\}^n$	*n* repeticiones de
	( )	Datos opcionales

Los flujos de datos del sistema son los siguientes:

1. FD del archivo = Nombre del archivo + Nombre del registro + $\{$ campo de datos $\}^n$
2. Campo de datos = Número de nivel + Nombre + Tamaño + Tipo de datos
3. Archivo secuencial sin campos COMP = Campos alfanuméricos + Campos numéricos en modo DISPLAY
4. Archivo secuencial NOS/VE = Campos alfanuméricos + Campos numéricos en modo DISPLAY + Campos numéricos en modo COMPUTATIONAL
5. Programa batch NOS/VE = Instrucciones COBOL NOS/VE
6. Programa batch RM/COBOL = Instrucciones RM/COBOL
7. Pantalla NOS/VE = Pantalla objeto + FD de la pantalla
8. Pantalla ASEMAP = Mapa de control de la pantalla + Variables dentro del programa COBOL

9. Programa interactivo NOS/VE = Instrucciones de COBOL NOS/VE
   + Llamadas a SCRF

10. Programa interactivo RM/COBOL = Instrucciones RM/COBOL
    + Llamadas a ASEMAP

11. Procedimiento en SCL = Comandos de SCL

12. Script en Korn shell = Comandos de UNIX

13. Campos para manejo de pantallas = Variable que almacena el valor del atributo + Variable que almacena el valor del campo

14. Variable que almacena el valor del atributo = Nombre del campo + "-P"

15. Variable que almacena el valor del campo = Nombre del campo + "-F"

16. Datos de pantalla SCRF = Datos de constantes + Datos de variables

17. Datos de constante = Posición de inicio + Longitud + Atributo + Valor

18. Datos de variable = Nombre de variable + Posición de inicio + Longitud
    + Atributo + Tipo (alfanumérica | entera | real) + (ocurrencias)
    + (signado) + (número de decimales) + (máscara de edición)

19. Cifras de control NOS/VE = { Total de suma del valor del campo importe *n* }^{n}

20. Cifras de control RM/COBOL = {Total de suma del valor del campo importe *n* }^{n}

21. Datos de campos elementales = Nombre del campo elemental + (Ocurrencias)

22. Campos elementales = Nombre del campo + (ocurrencias)
    + (bandera para indicar cálculo de cifras de control)

2.4 Miniespecificaciones.

Una vez que se ha representado el dominio de la información (usando DFD y un diccionario de datos), se describe cada proceso (transformación) mediante miniespecificaciones.

Una miniespecificación es una descripción de los procesos primitivos de los DFD's, es decir de los procesos que ya no se descompusieron en más burbujas.

Las miniespecificaciones serán hechas usando un pseudo-lenguaje llamado español estructurado.

A continuación se dá una descripción funcional de todos los procesos primitivos del sistema. Cada miniespecificación tendrá un número que será el mismo utilizado en el proceso del DFD correspondiente.

1.1 Reconoce campos elementales y sus ocurrencias.

Descripción: Reconoce los campos elementales y su número de ocurencias tomando como entrada el FD de un archivo indexado.

```
Lee FD del archivo indexado
MIENTRAS no sea fin de archivo
    SI se está describiendo un campo elemental ENTONCES
       Escribe nombre del campo elemental
    OSI SI se está describiendo un arreglo ENTONCES
       Escribe el nombre del elemento del arreglo y
                el tamaño del arreglo (sus ocurrencias)
    FIN SI
 FIN MIENTRAS
```

1.2 Genera convertidor de secuencial a ASCII.

Descripción: Generador de programas en COBOL que convierte el archivo secuencial a ASCII (sin campos COMPUTATIONAL).

```
Abre el archivo de "campos elementales" del archivo a convertir
PARA CADA línea en el archivo
        SI es un campo elemental
          y no se calcularán cifras de control en el campo ENTONCES
        Genera código para copiar el campo en el archivo secuencial a su
                    campo correspondiente en el archivo de salida
    OSI   SI  es un campo elemental
          y se calcularán cifras de control en el campo ENTONCES
         Genera código para copiar el campo en el archivo secuencial a su
                    correspondiente en el archivo de salida
         Genera código para acumular el valor del campo
    OSI   SI el campo es un arreglo
          y no se calcularán cifras de control en el campo ENTONCES
            PARA CADA ocurrencia del campo
                Genera código para copiar el elemento del arreglo a su
                    campo correspondiente de salida
            FIN PARA
      OSI   SI el campo es un arreglo
             y se calcularán cifras de control en el campo ENTONCES
            PARA CADA ocurrencia del campo
                Genera código para copiar el elemento del arreglo a su
                    campo correspondiente de salida
                Genera código para acumular el valor del elemento del
                    arreglo en una variable
            FIN PARA
   FIN SI
FIN PARA
```

1.3 Convierte archivo secuencial a ASCII.

Descripción: Programa COBOL que convierte el archivo secuencial en secuencial sin campos COMPUTATIONAL y calcula cifras de control.

```
Inicia todos los acumuladores de cifras de control a 0
Imprime "convirtiendo archivo " archivo
Abre el archivo secuencial de entrada y el secuencial de salida
MIENTRAS no sea fin de archivo
    Lee el siguiente registro
    Incrementa el número de registros leídos
     SI es un campo elemental
        y no se calcularán cifras de control en el campo ENTONCES
       Copia el campo en el archivo indexado a su correspondiente en el
             archivo secuencial
     OSI SI es un campo elemental
             y se calcularán cifras de control en el campo ENTONCES
         Copia el campo en el archivo indexado a su correspondiente en el
                  archivo secuencial
          Acumula el valor del campo en el acumulador del campo
     OSI   SI   el campo es un arreglo
                y no se calcularán cifras de control en el campo ENTONCES
            PARA CADA ocurrencia del campo
                 Copia el elemento del arreglo a su campo correspondiente
                    de salida
            FIN PARA
     OSI   SI   el campo es un arreglo
                y se calcularán cifras de control en el campo ENTONCES
            PARA CADA ocurrencia del campo
                 Copia el elemento del arreglo a su campo correspondiente
                    de salida
                 Acumula el valor del elemento del arreglo en el acumulador
                    del campo
            FIN PARA
    FIN SI
FIN MIENTRAS
Cierra archivos
Imprime el total de registros leídos
Imprime el total de registros grabados
PARA CADA  campo en que se calcularon cifras de control
     Imprime "Total del importe del campo: " acumulador del campo
FIN PARA
```

1.4 Genera convertidor de indexado a secuencial.

Descripción: Generador de programas en COBOL que convierten el archivo indexado a secuencial.

```
Abre el archivo de "campos elementales" del archivo a convertir
PARA CADA línea en el archivo
        SI es un campo elemental
           y no se calcularán cifras de control en el campo ENTONCES
         Genera código para copiar el campo del archivo indexado a su
                  correspondiente en el archivo secuencial
    OSI   SI   es un campo elemental
           y se calcularán cifras de control en el campo ENTONCES
         Genera código para copiar el campo del archivo indexado a su
                  correspondiente en el archivo secuencial
         Genera código para acumular el valor del campo
    OSI   SI el campo es un arreglo
           y no se calcularán cifras de control en el campo ENTONCES
           PARA CADA ocurrencia del campo
                Genera código para copiar el elemento del arreglo a su
                    campo correspondiente del arreglo secuencial de salida
           FIN PARA
      OSI   SI el campo es un arreglo
              y se calcularán cifras de control en el campo ENTONCES
           PARA CADA ocurrencia del campo
                Genera código para copiar el elemento del arreglo a su
                    campo correspondiente del arreglo secuencial de salida
                Genera código para acumular el valor del elemento del
                   arreglo en una variable
           FIN PARA
   FIN SI
FIN PARA
```

1.5 Convierte archivo indexado a secuencial.

Descripción: Programa COBOL que convierte el archivo indexado en secuencial y calcula cifras de control.

```
Inicia todos los acumuladores de cifras de control a 0
Imprime "convirtiendo archivo " archivo
Abre el archivo indexado de entrada y el secuencial de salida
Posiciona el apuntador al inicio del archivo indexado
MIENTRAS no sea fin de archivo
    Lee el siguiente registro
    Incrementa el número de registros leídos
    SI es un campo elemental
       y no se calcularán cifras de control en el campo ENTONCES
        Copia el campo del archivo indexado a su correspondiente en el
              archivo secuencial
    OSI SI es un campo elemental
                  y se calcularán cifras de control en el campo ENTONCES
        Copia el campo del archivo indexado a su correspondiente en el
                    archivo secuencial
        Acumula el valor del campo en el acumulador del campo
    OSI   SI   el campo es un arreglo
                  y no se calcularán cifras de control en el campo ENTONCES
          PARA CADA ocurrencia del arreglo
               Copia el elemento del arreglo a su campo correspondiente
                  del arreglo secuencial de salida
          FIN PARA
    OSI   SI   el campo es un arreglo
                   y se calcularán cifras de control en el campo ENTONCES
          PARA CADA ocurrencia del arreglo
               Copia el elemento del arreglo a su campo correspondiente
                  del arreglo secuencial de salida
              Acumula el valor del elemento del arreglo en el acumulador
                  del campo
          FIN PARA
   FIN SI
FIN MIENTRAS
Cierra archivos
Imprime el total de registros leídos
Imprime el total de registros grabados
PARA CADA  campo en que se calcularon cifras de control
    Imprime "Total del importe del campo: " acumulador del campo
FIN PARA
```

1.6 Genera convertidor de secuencial a indexado.

Descripción: Generador de programas en RM/COBOL que convierte el archivo secuencial a indexado

```
Genera código para abrir el archivo de campos elementales
POR cada línea en el archivo
    SI es un campo elemental
      y no se calcularán cifras de control en el campo ENTONCES
       Genera código para copiar el campo en el archivo secuencial a su
              correspondiente en el archivo indexado
    OSI  SI  es un campo elemental
        y se calcularán cifras de control en el campo ENTONCES
       Genera código para copiar el campo en el archivo secuencial a su
              correspondiente en el archivo indexado
        Genera código para acumular el valor del campo
    OSI SI el campo es un arreglo
          y no se calcularán cifras de control en el campo ENTONCES
           PARA CADA ocurrencia del arreglo
               Genera código para copiar el elemento del arreglo
                 secuencial a su campo correspondiente del arreglo
                 indexado de salida
           FIN PARA
      OSI  SI el campo es un arreglo
             y se calcularán cifras de control en el campo ENTONCES
           PARA CADA ocurrencia del arreglo
               Genera código para copiar el elemento del arreglo
                 secuencial a su campo correspondiente del arreglo
                 indexado de salida
               Genera código para acumular el valor del elemento del
                  arreglo en una variable
           FIN PARA
    FIN SI
 FIN PARA
```

1.6 Genera convertidor de secuencial a indexado.

Descripción: Generador de programas en RM/COBOL que convierte el archivo secuencial a indexado

```
Genera código para abrir el archivo de campos elementales
POR cada línea en el archivo
    SI es un campo elemental
       y no se calcularán cifras de control en el campo ENTONCES
        Genera código para copiar el campo en el archivo secuencial a su
                correspondiente en el archivo indexado
    OSI  SI  es un campo elemental
        y se calcularán cifras de control en el campo ENTONCES
        Genera código para copiar el campo en el archivo secuencial a su
                correspondiente en el archivo indexado
         Genera código para acumular el valor del campo
    OSI SI el campo es un arreglo
          y no se calcularán cifras de control en el campo ENTONCES
           PARA CADA ocurrencia del arreglo
                Genera código para copiar el elemento del arreglo
                  secuencial a su campo correspondiente del arreglo
                  indexado de salida
           FIN PARA
      OSI  SI el campo es un arreglo
              y se calcularán cifras de control en el campo ENTONCES
           PARA CADA ocurrencia del arreglo
                Genera código para copiar el elemento del arreglo
                  secuencial a su campo correspondiente del arreglo
                  indexado de salida
                Genera código para acumular el valor del elemento del
                   arreglo en una variable
           FIN PARA
   FIN SI
FIN PARA
```

1.7 Convierte archivo secuencial a indexado.

Descripción: Programa RM/COBOL que convierte el archivo secuencial a un archivo indexado y calcula cifras de control.

```
Inicia todos los acumuladores de cifras de control a 0
Imprime "Convirtiendo archivo " archivo
Abre el archivo secuencial de entrada y el indexado de salida
MIENTRAS no sea fin de archivo
   Lee el siguiente registro
   Incrementa el número de registros leídos
   SI es un campo elemental
      y no se calcularán cifras de control en el campo ENTONCES
       Copia el campo del archivo secuencial a su correspondiente en el
            archivo indexado
   OSI SI es un campo elemental
                y se calcularán cifras de control en el campo ENTONCES
        Copia el campo del archivo secuencial a su correspondiente en el
                archivo indexado
        Acumula el valor del campo en el acumulador del campo
   OSI  SI  el campo es un arreglo
              y no se calcularán cifras de control en el campo ENTONCES
          PARA CADA ocurrencia del arreglo
              Copia el elemento del arreglo secuencial a su campo
                correspondiente del arreglo indexado de salida
          FIN PARA
   OSI  SI  el campo es un arreglo
                y se calcularán cifras de control en el campo ENTONCES
          PARA CADA ocurrencia del arreglo
              Copia el elemento del arreglo secuencial a su campo
                correspondiente del arreglo indexado de salida
             Acumula el valor del elemento del arreglo en el acumulador
                 del campo
          FIN PARA
   FIN SI
FIN MIENTRAS
Cierra archivos
Imprime el total de registros leídos
Imprime el total de registros grabados
PARA CADA  campo en que se calcularon cifras de control
    Imprime "Total del importe del campo: " acumulador del campo
FIN PARA
```

2. Convierte FD del archivo

Descripción: Lee el FD del archivo indexado en NOS/VE y genera el FD del archivo indexado en RM/COBOL, sin campos COMPUTATIONAL.

```
Abre FD del archivo NOS/VE para lectura
PARA CADA campo del FD
     Borre la palabra "COMP" de la descripción del campo
FIN PARA
PARA CADA llave del archivo
      Calcule el nuevo tamaño de la llave que sea un campo redefinido
FIN PARA
```

3. Convierte programa batch

Descripción: Convierte un programa batch NOS/VE a un programa batch RM/COBOL.

```
Sustituye las instrucciones de manejo de bloqueo de registros NOS/VE
       por su equivalente en RM/COBOL
Convierte comandos de SCL por su equivalente en Korn shell
Sustituye llamadas a la rutina "$SCAN_COMMAND_LINE" por llamadas a la
      rutina "SYSTEM" para ejecutar comandos de sistema operativo y
      convierte comandos de SCL por su equivalente en Korn shell
Sustituye la frase "ORGANIZATION IS SEQUENTIAL" por
    "ORGANIZATION IS LINE SEQUENTIAL"
Mueve la cláusula "SELECT" de la zona A a la zona B
Pon entre comillas el nombre del archivo en la instrucción ASSIGN
Cambia la prueba de condición de clase "NOT NUMERIC" por
    "IS NOT NUMERIC"
Convierte los datos numéricos en modo COMPUTATIONAL a modo DISPLAY
```

4.1 Extrae datos de pantalla

Descripción: Lee los datos de cada pantalla indicada en el archivo LIST_SCR y graba esta información en el archivo DECODE_SCR.

NOTA: Para designar una subcadena se usa la notación:
nombre_cadena(posición_inicio, longitud)
Si como longitud de la subcadena se usa el carácter "-", se está indicando que la subcadena abarca desde la posición de inicio hasta el final de la cadena.

```
Abre el archivo LIST_SCR de entrada
Abre el archivo DECODE_SCR de salida
REPITE
   Lee el nombre de la siguiente forma a procesar (nombre_forma)
   Escribe (1,9) 'FORM      ' (8,31) nombre_forma a  DECODE_SCR
   Escribe (1,40) 'CONSTANTS' a  DECODE_SCR
   Abre la forma nombre_forma
   Obtén los nombres de las tablas definidas para la forma actual
   PARA CADA tabla definida en la forma actual
      Obtén los atributos de la tabla (nombre y atributos de la tabla)
      Guarda en el arreglo de tablas el nombre de la tabla
      Guarda en el arreglo de occurrencias el número de ocurrencias de la
            tabla
   FIN PARA
   Obtén los objetos definidos en la forma actual y número de objetos
   PARA CADA objeto definido en la forma actual
      Guarda en el arreglo objetos de la forma los atributos renglón y
            columna de inicio del objeto
      Guarda en línea_salida(33,2) el atributo renglón  y en
            línea_salida(35,2) el atributo columna de inicio
      EN CASO DE  tipo del objeto
        CAJA:
           línea_salida(37,2) = anchura de caja
           línea_salida(39,2) = altura de caja
           Guarda en el arreglo de cajas el contenido de la cadena
                 línea_salida(1,-)
        TEXTO_CONSTANTE:
            línea_salida(45,-) = texto
            línea_salida(37,2) = longitud del texto constante
            línea_salida(39,2) = atributo de despliegue
            Escribe línea_salida al archivo DECODE_SCR
```

```
        TEXTO_VARIABLE:
             SI es un arreglo ENTONCES
                  línea_salida(50,2) = número de ocurrencias
             FIN SI
            línea_salida(1,31) = nombre de la variable
            Obtén atributos de la variable
            línea_salida(39,2) = atributo de despliegue
            línea_salida(45,1) = formato de salida (carácter, moneda,
                   entero, real en notación de punto fijo o real
                   en notación de punto flotante)
            Guarda en el arreglo de variables el contenido de la cadena
                   línea_salida
     FIN CASO
FIN DESDE
SI el número de objetos de texto variables fue mayor a  0 ENTONCES
     línea_salida(1,9) = 'VARIABLES'
     escribe línea_salida al archivo DECODE_SCR
     PARA CADA objeto de texto variable
            Escribe el contenido del arreglo de variables al archivo
                   DECODE_SCR
     FIN PARA
FIN SI
SI el número de cajas fue mayor a  0 ENTONCES
     línea_salida(1,9) = 'BOXES    '
     escribe línea_salida al archivo DECODE_SCR
     PARA CADA caja
            Escribe el contenido del arreglo de cajas al archivo
                   DECODE_SCR
     FIN PARA
FIN SI
línea_salida(1,9) = 'END      '
escribe línea_salida al archivo DECODE_SCR
Cierra forma
FIN SI
HASTA  no sea fin de archivo
Cierra archivos LIST_SCR y DECODE_SCR
```

4.2 Convierte pantalla a ASEMAP

Descripción: Lee la información dejada en el archivo DECODE_SCR y genera los copys de las pantallas para ASEMAP.

Escribir en el archivo de pantalla el encabezado, que consiste en el nombre de la pantalla entre asteriscos
PARA CADA cadena constante en la matriz de pantalla
    Escribir en el archivo de formato de pantalla dentro de FILLER's de COBOL una tilde seguida de dos caracteres que corresponden a la posición renglon columna del primer carácter de la cadena en la pantalla y luego el texto de la cadena constante
FIN PARA
Leer las variables de la pantalla en la tabla de variables
Ordenar la tabla de variables comenzando por las alfanuméricas y éstas a su vez de acuerdo a su posición en la pantalla, ésto es, de izquierda a derecha y de arriba a abajo
PARA CADA variable en la tabla de variables
    Escribir en conjuntos de once caracteres en el archivo de formato de pantalla los caracteres correspondientes a su posición en el arreglo de las variables, dos signos de interrogación, posición renglón columna en la pantalla, longitud larga y corta, número de decimales, máscara de edición y tipo de signado
FIN PARA
Escribir un FILLER con LOW-VALUES
Escribir en el archivo de formato de pantalla las variables de la pantalla con el formato de COBOL
Escribir un FILLER con LOW-VALUES

5.1 Reconoce campos correspondientes entre SCRF y ASEMAP

Descripción: Lee el archivo CDM_RUT_SCREEN_<PROGRAMA> y reconoce la variable correspondiente de SCRF que almacena el valor del atributo en ASEMAP y genera comandos de sed para sustituirla por la de ASEMAP.

```
PARA CADA variable que almacena el valor del atributo en SCRF
     de la forma FACOF-<CAMPO>
    Encuentra el nombre de la variable correspondiente en ASEMAP que
         almacena el valor del atributo y genera un comando que la
         sustituya
FIN PARA
PARA CADA variable que Almacena el valor del campo en SCRF
     de la forma <CAMPO>-P
    Encuentra el nombre de la variable correspondiente en ASEMAP que
         almacena el valor del campo y genera un comando que la
         sustituya, cuando el nombre no sea el mismo
FIN PARA
```

5.2 Convierte programa interactivo

Descripción: Convierte un programa interactivo en COBOL NOS/VE en un programa interactivo en RM/COBOL, con instrucciones para el manejo de pantallas por ASEMAP.

```
    Sustituye las instrucciones de manejo de bloqueo de registros NOS/VE
              por su equivalente en RM/COBOL
    Convierte comandos de SCL por su equivalente en Korn shell
    Sustituye llamadas a la rutina "$SCAN_COMMAND_LINE" por llamadas a la
         rutina "SYSTEM" para ejecutar comandos de sistema operativo y
         convierte comandos de SCL por su equivalente en Korn shell
    Sustituye la frase "ORGANIZATION IS SEQUENTIAL" por
        "ORGANIZATION IS LINE SEQUENTIAL"
    Mueve la cláusula "SELECT" de la zona A a la zona B
    Pon entre comillas el nombre del archivo en la instrucción ASSIGN
    Cambia la prueba de condición de clase "NOT NUMERIC" por
        "IS NOT NUMERIC"
    Convierte los datos numéricos en modo COMPUTATIONAL a modo DISPLAY
    Convierte el valor del atributo en SCRF por su atributo
        correspondiente en ASEMAP
```

Convierte el nombre de la variable que almacena el valor del atributo en SCRF (FACOF-*VARIABLE*) por su correspondiente en ASEMAP (*VARIABLE*-P).

6. Convierte procedimiento.

Descripción: Convierte un procedimiento en SCL de NOS/VE a un script en Korn shell de UNIX.

```
PARA CADA comando en SCL
      Convierte a su equivalente en Korn shell
FIN PARA
```

3. Diseño.

3.1 Introducción.

El diseño puede ser definido como "el proceso de aplicar distintas técnicas y principios con el propósito de definir un proceso o sistema con los suficientes detalles como para permitir su realizacion física"

El diseño estructurado es una metologia orientada al flujo de datos. Parte del diagrama de flujo de datos (DFD) y convierte las especificaciones en cartas estructuradas y la descripción de cada uno de los modulos.

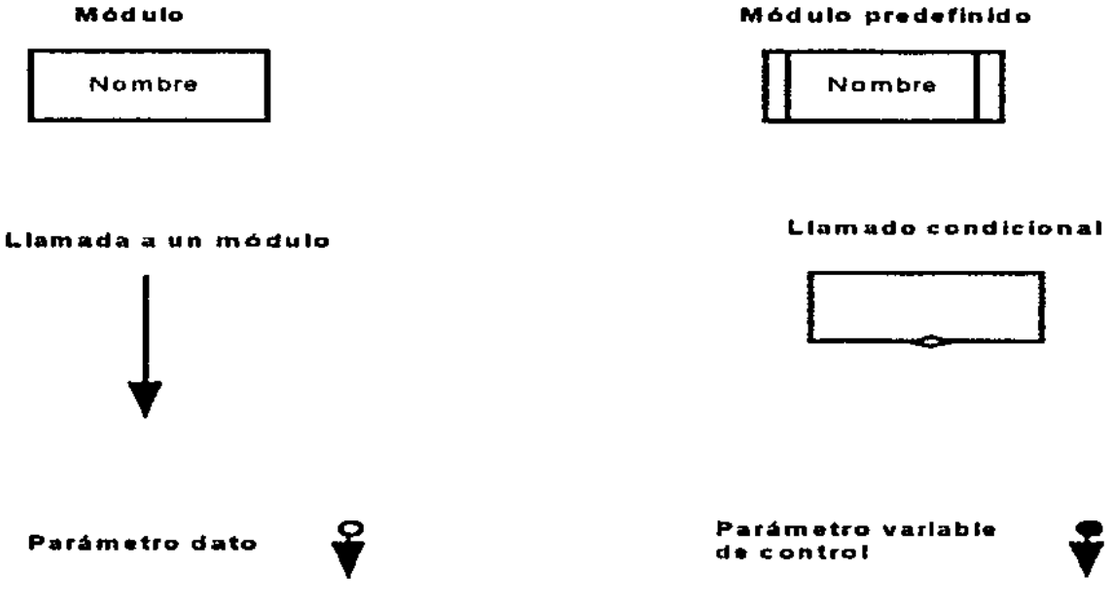
Se establece el tipo del flujo de información en el DFD. Un DFD presenta un flujo de transformación cuando la información entra como un flujo de llegada, sufre una transformación y se presenta un flujo de salida. Un flujo de transacción se presenta cuando los datos que se mueven a lo largo de un camino de llegada se convierten en una transacción, la cual es evaluada y, basandose en su valor, el flujo se inicia por uno de los muchos *caminos de acción*

El DFD se convierte a la estructura de transformación o de transacción. Las transformaciones (burbujas del DFD) se convierten en estructura de programa como módulos. La estructura de un programa representa una distribución descendente del control. La *factorización* da como resultado una estructura de programa en la que los módulos de nivel superior toman las decisiones de ejecución y los módulos de nivel inferior ejecutan la mayoría del trabajo de entrada, transformación y de salida.

Después se refina la estructura del programa aplicando los conceptos de independencia funcional de módulos y varias heurísticas de diseño.

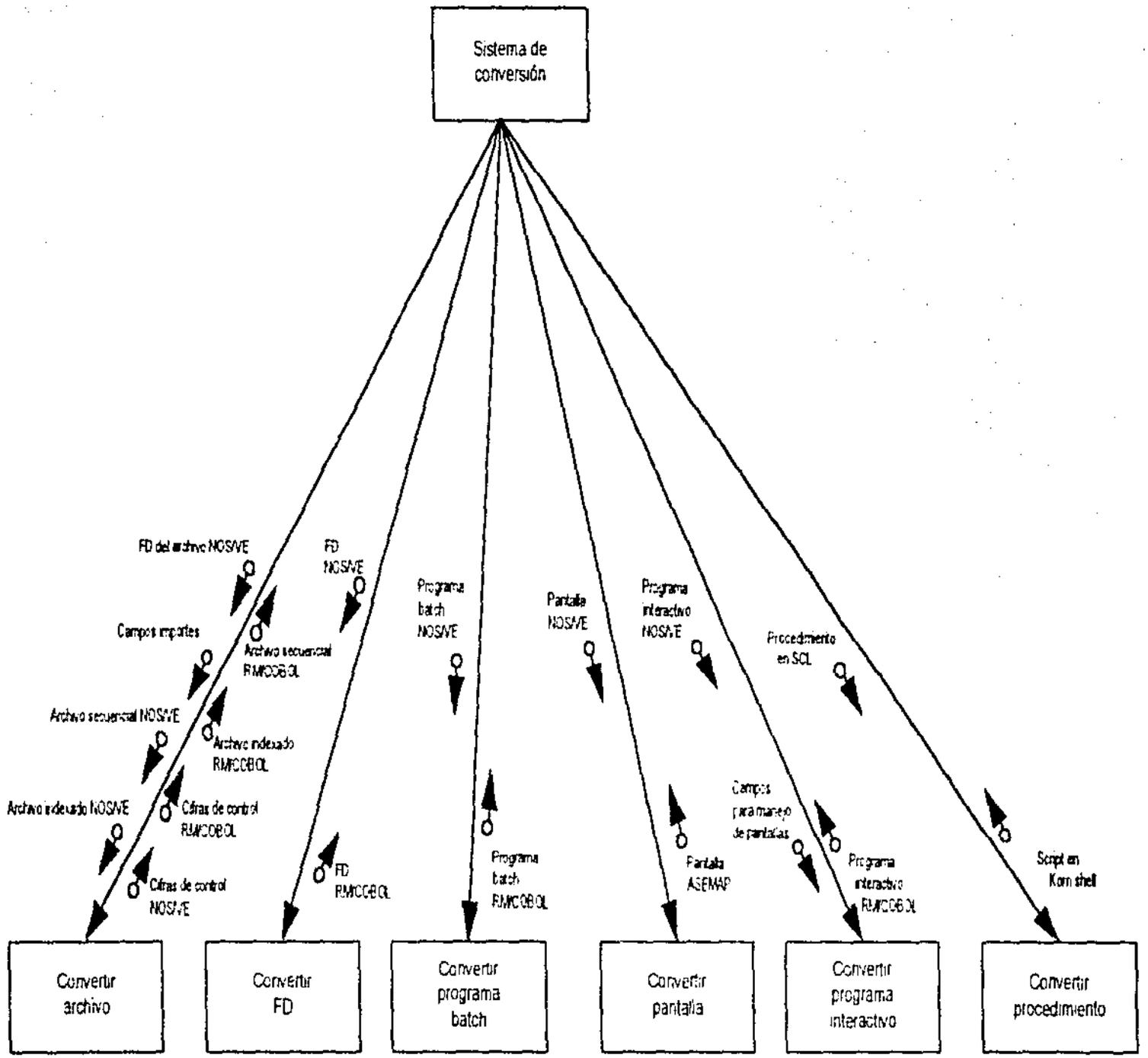
La independencia funcional se adquiere desarrollando módulos con una función clara y con una interfaz sencilla con otros módulos, teniendo "aversión" a la excesiva interacción entre módulos. Se mide usando los criterios de cohesión y acoplamiento. Un módulo coherente ejecuta una tarea sencilla y requiere poca interacción con procedimientos que se ejecutan en otras partes de un programa. El acoplamiento es una medida de la interconexión entre módulos en una estructura de programa.

A continuación se presenta la simbología utilizada en las cartas estructuradas:

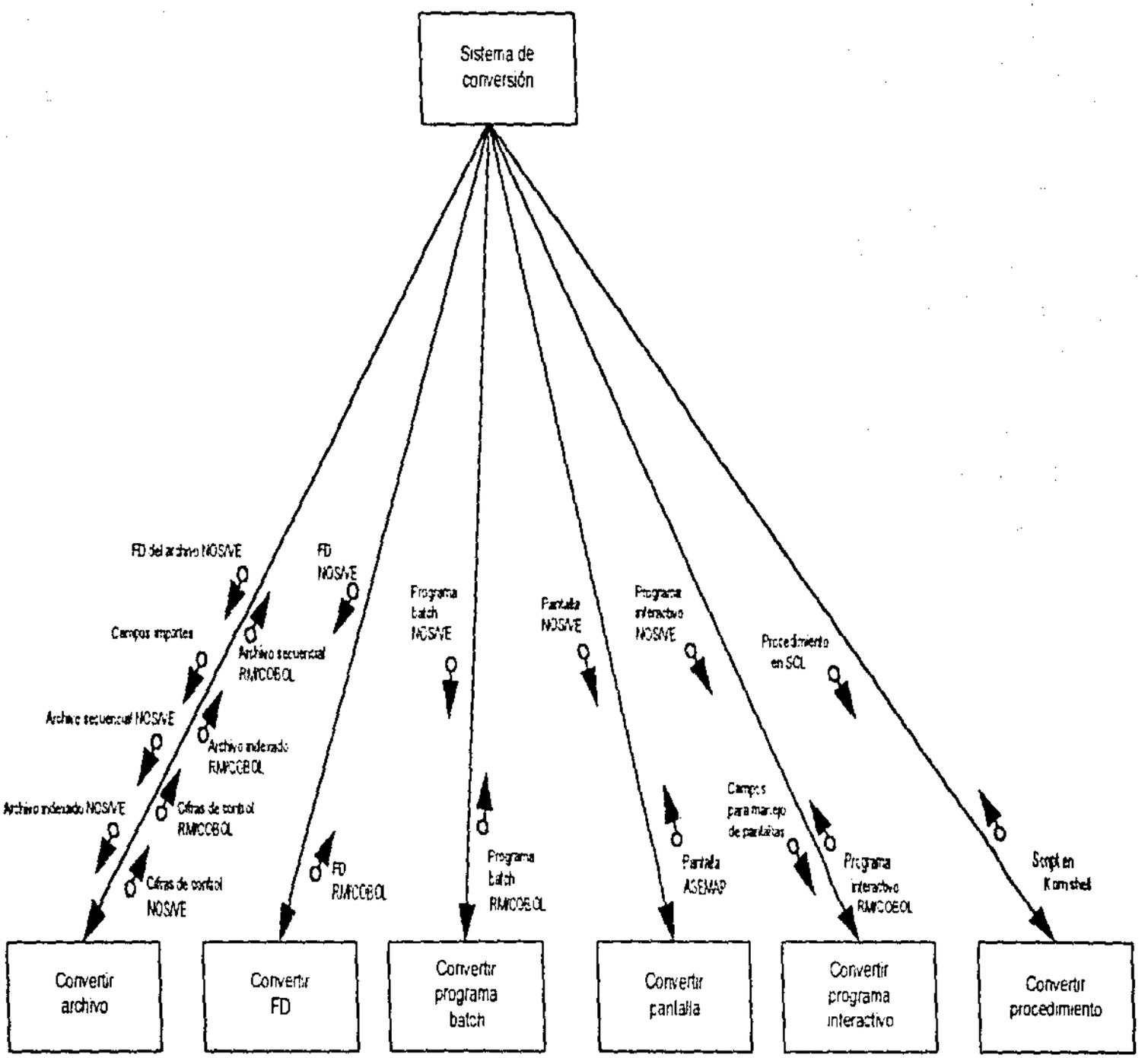


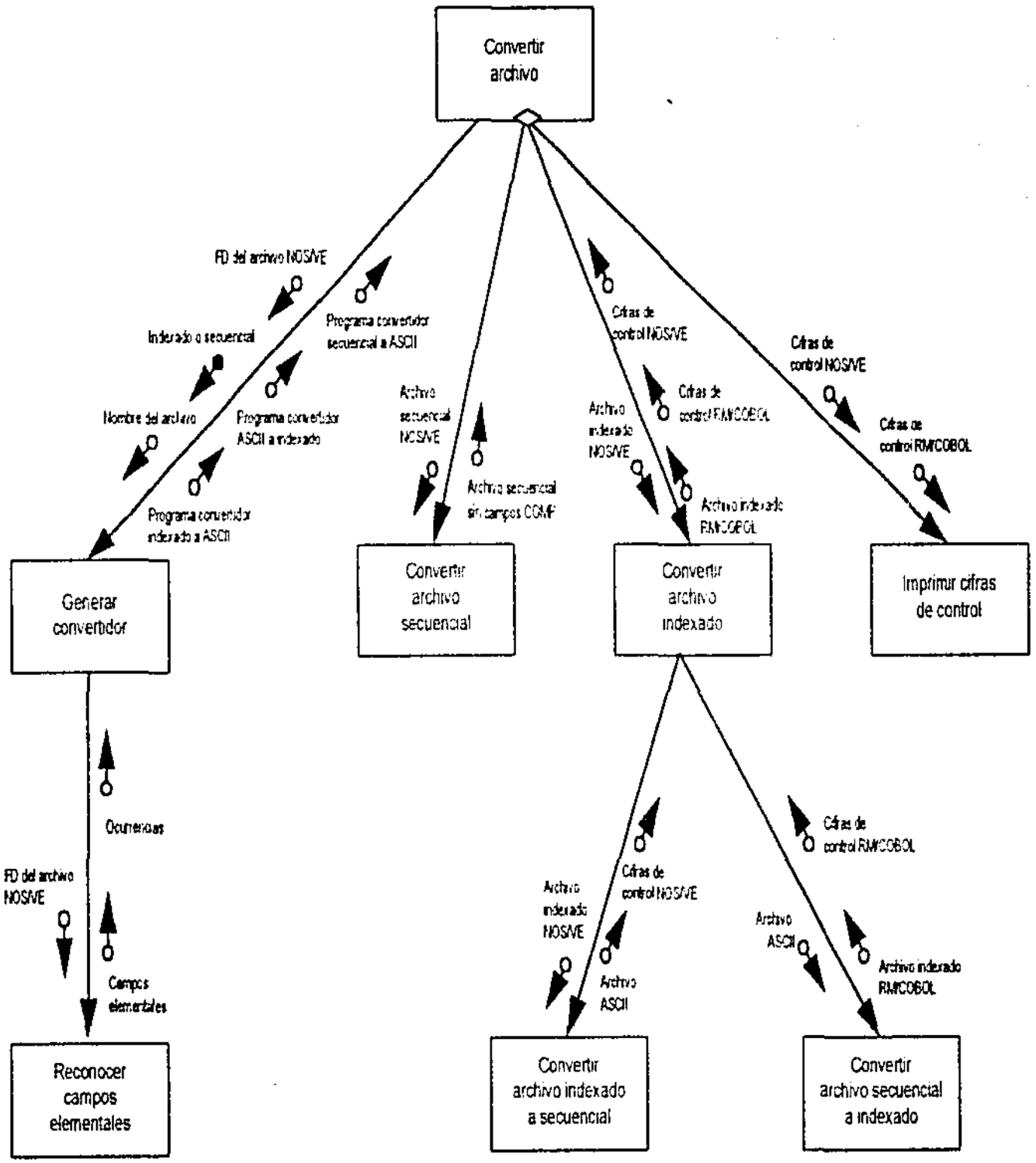
Se muestran a continuación las cartas estructuradas que resultaron de este análisis.

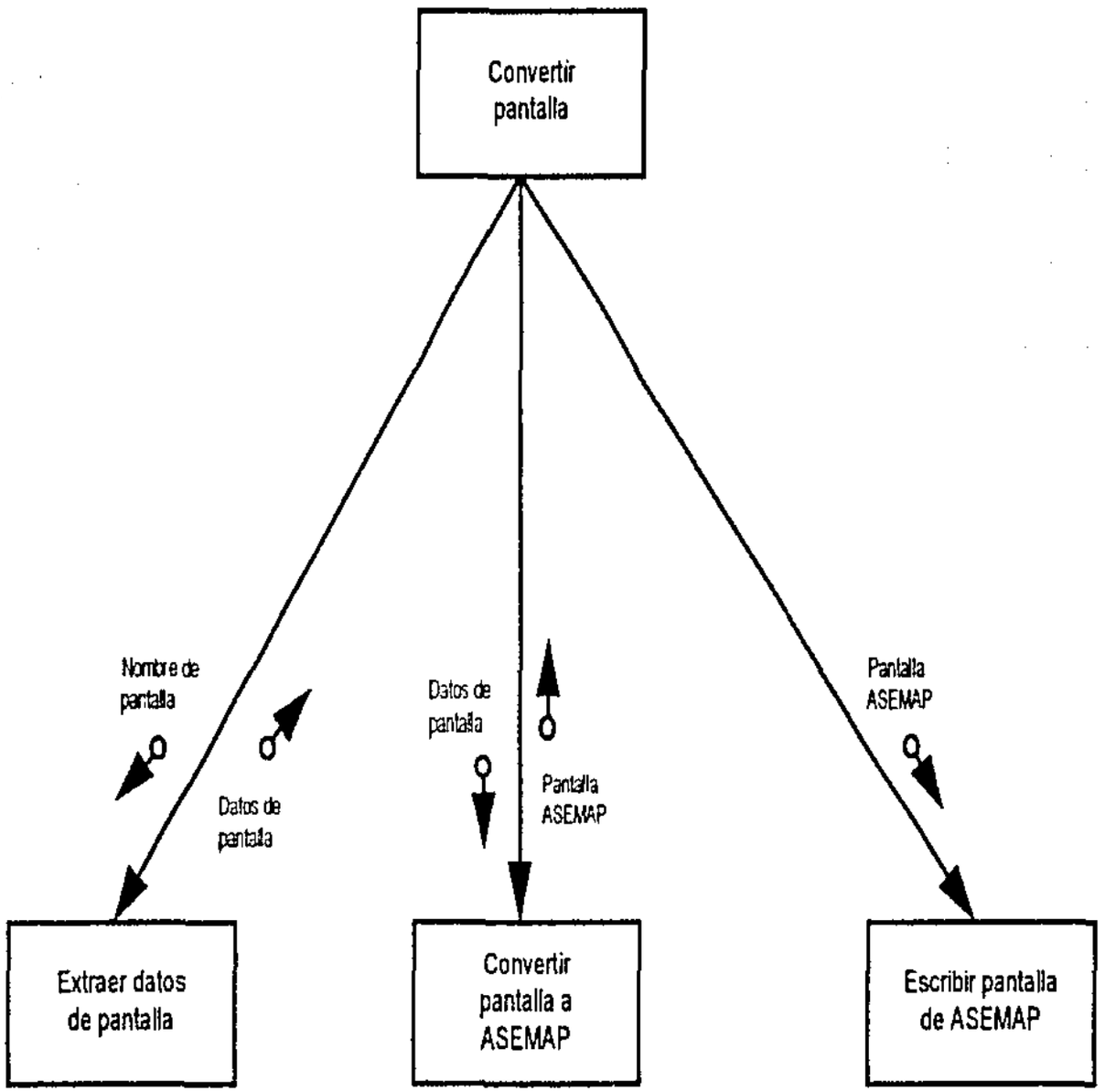
3.2 Cartas estructuradas del sistema.

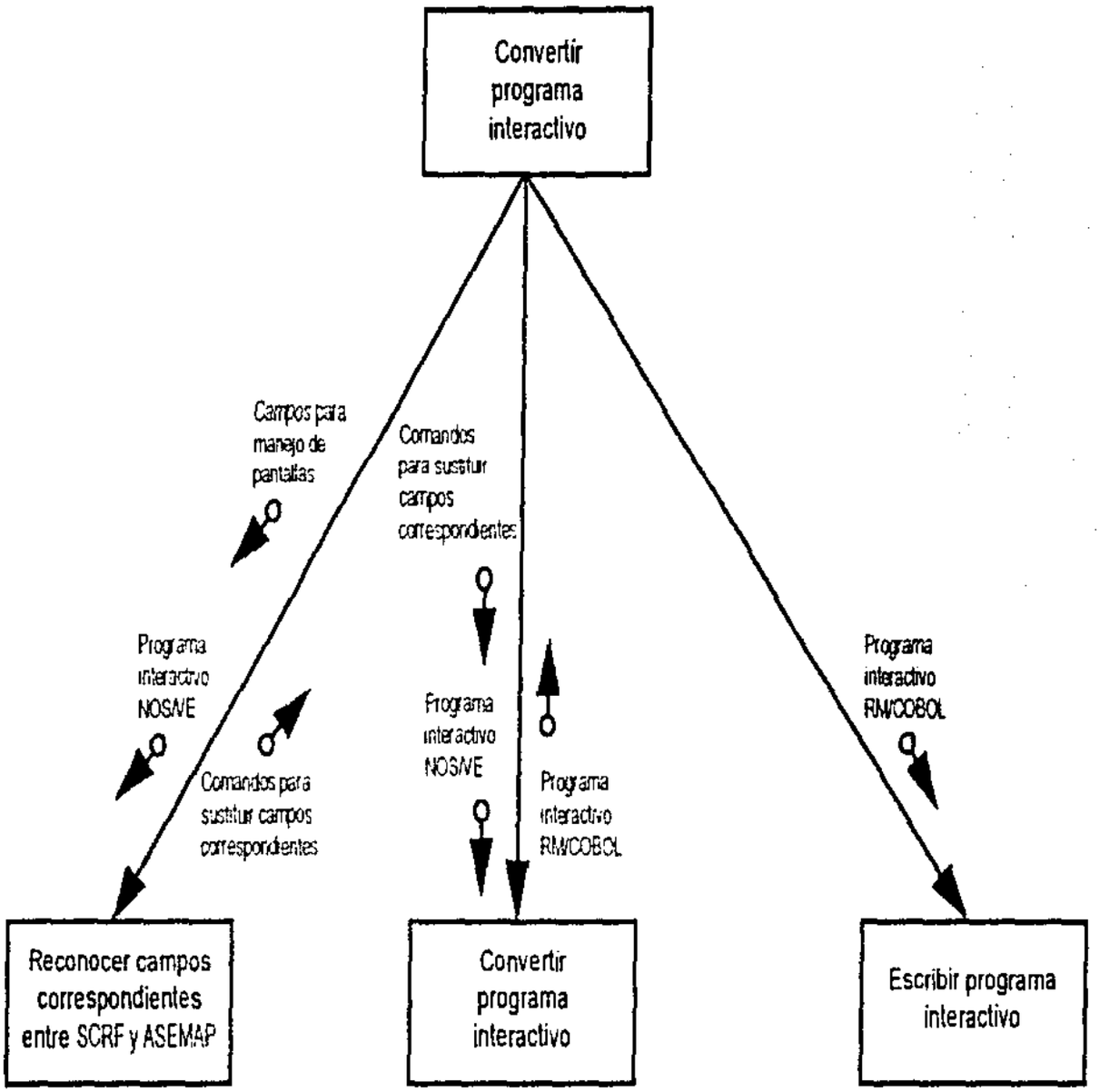


3.2 Cartas estructuradas del sistema.




Convertir archivo
FD del archivo NOS/VE
Indexado o secuencial
Nombre del archivo
Programa convertidor secuencial a ASCII
Programa convertidor ASCII a indexado
Programa convertidor indexado a ASCII
Archivo secuencial NOS/VE
Archivo secuencial sin campos COMP
Archivo indexado NOS/VE
Archivo indexado RM/COBOL
Cifras de control NOS/VE
Cifras de control RM/COBOL
Cifras de control NOS/VE
Cifras de control RM/COBOL
Generar convertidor
Convertir archivo secuencial
Convertir archivo indexado
Imprimir cifras de control
Ocurrencias
FD del archivo NOS/VE
Campos elementales
Archivo indexado NOS/VE
Archivo ASCII
Cifras de control NOS/VE
Cifras de control RM/COBOL
Archivo ASCII
Archivo indexado RM/COBOL
Reconocer campos elementales
Convertir archivo indexado a secuencial
Convertir archivo secuencial a indexado


Convertir pantalla
Nombre de pantalla
Datos de pantalla
Datos de pantalla
Pantalla ASEMAP
Pantalla ASEMAP
Extraer datos de pantalla
Convertir pantalla a ASEMAP
Escribir pantalla de ASEMAP


Convertir programa interactivo
Campos para manejo de pantallas
Comandos para sustituir campos correspondientes
Programa interactivo NOS/VE
Comandos para sustituir campos correspondientes
Programa interactivo NOS/VE
Programa interactivo RM/COBOL
Programa interactivo RM/COBOL
Reconocer campos correspondientes entre SCRF y ASEMAP
Convertir programa interactivo
Escribir programa interactivo

4. Diseño de procedimientos y utilerías para facilitar la conversión.

4.1 Introducción.

El realizar manualmente los programas que realizaran la conversión de archivos y las modificaciones muy frecuentes en los programas batch e interactivos suponía un gran esfuerzo para los programadores, así como invertir una gran cantidad de tiempo. Por lo tanto, se vió que se podían diseñar las siguientes herramientas que facilitarían mucho la conversión:

a) Generadores de programas en COBOL que conviertan los archivos indexados en secuenciales y obtengan cifras de control.

b) Un programa convertidor de programas batch e interactivos de COBOL NOS/VE a RM/COBOL.

c) Programas para la conversión de pantallas de programas interactivos en CYBER a la utilería ASEMAP en UNIX.

Se analiza a continuación la forma de realizar la conversión de archivos, programas y pantallas y el diseño de cada una de las herramientas para facilitar la conversión.

4.2 Conversión de archivos secuenciales e indexados.

Debido a que la forma de representación de los datos numéricos COMPUTATIONAL en COBOL NOS/VE es distinta a la de RM/COBOL, es necesario convertir los datos numéricos en modo COMPUTATIONAL a modo DISPLAY (ASCII) en NOS/VE antes de transferirlos al equipo HP. En el equipo HP se decidió dejar los datos numéricos en modo DISPLAY, sin convertirlos a modo COMPUTATIONAL, por políticas de la empresa, debido a que los sistemas ya existentes en RM/COBOL no manejan datos numéricos en modo COMPUTATIONAL.

Un problema que se presentó en los campos numéricos signados al dejarlos en modo DISPLAY en COBOL NOS/VE y RM/COBOL, fue que al realizar pruebas con los programas se encontraron *problemas de acceso a registros cuya llave estaba compuesta de campos numéricos signados*. Esto se debe a la forma en que se almacenan tanto en COBOL NOS/VE como en RM/COBOL los campos numéricos signados en modo DISPLAY.

Para los elementos de datos numéricos signados en modo DISPLAY se usa un byte entero por dígito decimal, pero el signo se representa, por omisión, en combinación con el último dígito en el último byte, según la siguiente tabla:

Signo y dígito	*Represen-tación*
+9	I
+8	H
+7	G
+6	F
+5	E
+4	D
+3	C
+2	B
+1	A
+0	{
-0	}
-1	J
-2	K
-3	L
-4	M
-5	N
-6	O
-7	P
-8	Q
-9	R

En muchos casos, los campos numéricos signados en modo DISPLAY formaban parte de alguna de las llaves del archivo, y como las llaves se redefinen como alfanuméricas, no quedan en el orden pensado. Esto provoca que instrucciones como:

START *ARCHIVO* KEY NOT LESS THAN

o SORT no funcionen como se espera en los programas.

Para solucionar ésto, se tomó como norma el quitar el signo a todos los campos numéricos signados que no sean importes. Como el problema se presenta en los campos numéricos signados que forman parte de alguna llave del archivo y estos campos no necesitan ser signados, pues no son importes, el problema quedó resuelto.

Después de hacer pruebas y consultar los manuales de COBOL, se encontró que tanto el conjunto de caracteres como la secuencia de ordenación *(collating sequence)* que usan COBOL NOS/VE y RM/COBOL es ASCII.

Para que hubiera una manera de verificar que los archivos fueron correctamente convertidos, se usaron cifras de control en los campos que eran importes. Esto consiste en sumar el mismo campo para todos los registros de un archivo en NOS/VE y hacer lo mismo en RM/COBOL. Si no coincidía esta suma, había un problema en la conversión de archivos.

4.2.1 Conversión de archivos indexados.

Los archivos indexados tienen una estructura física distinta en los dos sistemas operativos. Para realizar la conversión de archivos indexados es necesario primero convertirlos a un formato que ambos sistemas entiendan, los archivos ASCII.

Los archivos de texto ASCII, también llamados archivos ASCII o archivos de texto, son un formato universal para archivos de datos. Mientras muchos programas tienen su propios formatos de datos especiales, el archivo de texto ASCII es el formato común para pasar datos de una aplicación a otra, o entre distintos sistemas operativos. Sólo los primeros 128 caracteres (0 - 127) dentro de las 256 combinaciones de un byte conforman el estándar ASCII. El resto son utilizados en forma diferente dependiendo del equipo.

Por lo tanto, se convertirán los datos numéricos en modo COMPUTATIONAL a modo DISPLAY antes de transferirlos en forma ASCII del equipo CYBER al equipo HP.

Por lo tanto es necesario efectuar los siguientes pasos para realizar la conversión de archivos indexados:

1. *Convertir los archivos indexados a archivos secuenciales ASCII en NOS/VE*; es necesario convertir los datos numéricos en modo COMPUTATIONAL a modo DISPLAY y quitar el signo a todos los campos numéricos signados que no sean importes.

Para realizar la conversión de archivos indexados a secuenciales en CYBER se cuenta con la utilería FMU para formatear archivos y otra forma sería mediante un programa en COBOL. Sin embargo, debido a que con la utilería FMU no se pueden calcular cifras de control sobre los campos de importes, se decidió usar un solo programa en COBOL por cada archivo para realizar la conversión de indexado a secuencial y el cálculo de las cifras de control sobre los campos de importes. Esto permite hacer la conversión más rápidamente y no añadir más carga al equipo CYBER.

2. *Transferir los archivos del equipo CYBER al equipo HP.* Esto se puede realizar mediante el comando `ftp`. Se pueden transmitir en formato ASCII o binario mediante `ftp`.

Cuando se transfería en formato ASCII, el comando `ftp` agregaba un carácter '\n' (salto de línea) al final de cada registro del archivo secuencial. En este caso, el programa en RM/COBOL necesita la declaración "ORGANIZATION IS LINE SEQUENTIAL" para leer el archivo como secuencial lineal.

Un archivo secuencial de texto ASCII puede transmitirse en forma binaria mediante `ftp` desde NOS/VE al equipo HP. Este archivo ASCII puede ser leído sin problema por un programa en RM/COBOL declarando la organización del archivo

como secuencial binario ("ORGANIZATION IS BINARY SEQUENTIAL"). Esto se debe a que NOS/VE, al igual que RM/COBOL, almacenan un archivo secuencial de longitud fija sin ninguna estructura adicional.

3. *Convertir, en el equipo HP, los archivos secuenciales ASCII a archivos indexados en RM/COBOL.*

Se realiza mediante programas en COBOL que leen el archivo secuencial, lo convierten a un archivo indexado y calculan cifras de control.

Se debe verificar que las cifras de control calculadas en COBOL NOS/VE sean iguales a las calculadas en RM/COBOL.

4.2.2 Conversión de archivos secuenciales.

Como todos los archivos secuenciales que se usan en NOS/VE son de longitud fija y se almacenan sus registros uno tras otro sin alguna estructura adicional, se pueden transferir directamente con el comando `ftp` de CYBER a HP. Sin embargo, si el archivo secuencial tiene datos numéricos en modo COMPUTATIONAL, es necesario convertirlos a modo DISPLAY antes de transferirlos al equipo HP, debido a la razón mencionada anteriormente. Una vez transferidos, se acordó dejar los datos numéricos en modo DISPLAY.

4.2.3 Desarrollo de un generador de programas en COBOL NOS/VE que conviertan los archivos indexados en secuenciales y obtengan cifras de control.

Primeramente se desarrolló un programa modelo para la conversión de archivos indexados a secuenciales en NOS/VE y que calculara las cifras de control. El programa modelo efectúa las siguientes funciones:

1. Los datos numéricos en modo COMPUTATIONAL en el archivo indexado se convierten a modo DISPLAY en el archivo secuencial.
2. Calcular cifras de control sobre los campos que son importes.
3. Se lee todo el archivo indexado, copiando el contenido de cada uno de los campos del archivo indexado (que tienen la terminación "-FD") al campo correspondiente en el archivo secuencial de salida (que tendrá la terminación "-SQ"). Por ejemplo:

```
MOVE OFICINA-CLAVESAC-FD TO OFICINA-CLAVESAC-SQ
```

Hay que hacer el FD del archivo secuencial de salida, copiando el FD del archivo indexado y cambiando la cadena "FD <ARCHIVO>" por "FD <ARCHIVO>SEQ", y por cada campo, cambiar "-FD" por "-SQ"; hay que borrar también la definición de las llaves. Se quitó el signo a todos los campos numéricos signados que no sean importes. Hay que borrar la palabra "COMP" de la descripción del archivo.

Por ejemplo, si se tiene la siguiente descripción del archivo:

```
FD   DESCONT
     LABEL RECORD IS OMITTED.
01   REG-DESCONT-FD.
     02  DESCONT-FD.
         05   CONSEC1-DESCONT-FD        PIC 9(09).
         05   NIVEL3-DESCONT-FD-VEC.
              06   NIVEL3-DESCONT-FD    PIC 9(04)
                   OCCURS 10 TIMES.
         05   TIPODATO-DESCONT-FD-VEC.
              06   TIPODATO-DESCONT-FD PIC X(01)
                   OCCURS 10 TIMES.
         05   NUMDECIM-DESCONT-FD       PIC 9(04) COMP.
         05   MOVNEGAT-DESCONT-FD       PIC  X(01).
     02 DESCONT-FD-00 REDEFINES DESCONT-FD.
         05   DESCONTK-00               PIC X(09).
         05   FILLER                    PIC X(55).
```

La descripción del archivo secuencial de salida sería:

```
FD   DESCONTSEQ
     LABEL RECORD IS OMITTED.
01   REG-DESCONT-SQ.
     02  DESCONT-SQ.
         05   CONSEC1-DESCONT-SQ        PIC 9(09).
         05   NIVEL3-DESCONT-SQ-VEC.
              06   NIVEL3-DESCONT-SQ    PIC 9(04)
                   OCCURS 10 TIMES.
         05   TIPODATO-DESCONT-SQ-VEC.
              06   TIPODATO-DESCONT-SQ PIC X(01)
                   OCCURS 10 TIMES.
         05   NUMDECIM-DESCONT-SQ       PIC 9(04).
         05   MOVNEGAT-DESCONT-SQ       PIC X(01).
```

Para generar el FD del archivo secuencial de salida, se usa un script en base a comandos de `sed`.

`sed` es un editor de flujo orientado a líneas, no interactivo. Un editor de flujo lee la entrada estándar (o archivos), posiblemente aplica uno o más comandos de edición a esa entrada, y entonces escribe esta salida a la salida estándar. `sed` no altera los contenidos de sus archivos de entrada, escribe en la salida estándar.

`sed` suministra muchas funciones de edición que permiten: cambiar partes de una línea, añadir algún texto a la entrada, borrar algunas partes de la entrada, añadir nuevas líneas, borrar una línea, cambiar una línea, desplegar una línea, añadir una línea al archivo, salvar y recuperar la entrada para uso posterior, saltar a otras funciones de edición. `sed` puede buscar las líneas que concuerden con un patrón o expresión regular, y realizar la acción correspondiente cuando lo encuentra.

Una expresión regular es una cadena de caracteres que puede ser usada para "igualar" un *conjunto* de cadenas de caracteres.

La sintaxis de sed es:

```
sed    'lista de comandos'   archivos ...
```

sed lee un renglón a la vez a partir de los archivos de entrada; aplica posiblemente los comandos de edición de la lista, por orden, a cada renglón y escribe la línea resultante en la salida estándar.

También es posible poner los comandos de sed en un archivo (cmdfile) y ejecutarlos desde allí por medio de:

```
sed    -f   cmdfile   archivos ...
```

Por ejemplo, los siguientes comandos de sed se usan para generar el FD del archivo secuencial de salida en base al FD del archivo indexado en NOS/VE:

```
s/   *COMP   */ /
s/   *COMP *\./\./
s/   *COMP *$//
s/-FD /-SQ /
s/-FD-/-SQ-/
s/-FD\./-SQ./
```

Estos comandos borran la palabra "COMP" de la descripción del archivo, y en cada campo sustituyen la cadena "-FD" por "-SQ".

Comandos similares se usan para generar el FD del archivo indexado de RM/COBOL

El generador de programas COBOL que convierten los archivos indexados en secuenciales necesita como base saber los nombres de los campos que se deben copiar del archivo indexado al secuencial y el número de ocurrencias de los campos que son arreglos. Para efectuar ésto, es necesario elaborar un programa que reconozca los nombres de los campos tomando como entrada la descripción del archivo (FD) de COBOL NOS/VE; se usó YACC y LEX para generar dicho programa.

LEX es un generador de analizadores léxicos y puede ser utilizado por analizadores de sintaxis generados por YACC. LEX traduce expresiones regulares a un autómata DFA escrito en lenguaje C, que reconoce un lenguaje.

A continuación se muestra el programa en LEX para reconocer los componentes léxicos:

```
D          [0-9]+
ID         [a-zA-Z][a-zA-Z0-9-]*
%%
[ \t\n]                    ;
COMP                       ;
TIMES                      ;
```

```
OCCURS                  { return OCCURS;      }
REDEFINES               { return REDEFINES;   }
{D}+                    { sscanf(yytext, "%d", &yylval);
                          return NUMBER; }
PIC" "+[0-9()A-Z]+      { return PIC;   }
{ID}                    { strcpy(campo, yytext);
                          return CAMPO;   }
\.                      { return '.';     }
[^\.]                   ;
%%
```

YACC es un generador de analizadores de sintaxis LALR(1). YACC recibe como entrada la especificación de una gramática de contexto libre que describe un lenguaje de programación. Como salida se obtiene un programa en C que analiza o verifica el lenguaje. Se añade código de acción a la gramática para crear un traductor dirigido por sintaxis.

La gramática que se usa para reconocer los campos elementales y su número de ocurrencias leyendo como entrada el FD del archivo indexado es la siguiente:

lista : ε
    | *lista expr* '.'
    ;
expr : **número campo picture ocurrencias número**
    | **número campo ocurrencias número picture**
    | **número campo redefines campo**
    | **número campo ocurrencias número**
    | **número campo picture**
    | '.'

Esta gramática se pasa a una especificación en YACC del traductor, la cual se hace pasar por el compilador de YACC para generar un programa en lenguaje C, que es compilado con el compilador de C para obtener el programa objeto deseado. A continuación se muestra el listado de la especificación en YACC del programa campo.y :

```
/* CAMPO.Y
   Reconoce los campos elementales y su numero de ocurrencias
   tomando como entrada el FD del archivo indexado */
%{
#include <string.h>
#include <stdio.h>
char campo[50];
int  ocurr, nivel, flag = 0;
%}
%token OCCURS
%token NUMBER
```

```
%token CAMPO
%token PIC
%token REDEFINES
%%
list :   /* nada */
       |   list expr '.'
       |   list error '.'                { yyerrok; }
       ;
expr :   NUMBER CAMPO PIC OCCURS NUMBER
                { printf("%s %d\n", campo, $5);   }
       |  NUMBER CAMPO OCCURS NUMBER PIC
                { printf("%s %d\n", campo, $4);   }
       |  NUMBER CAMPO REDEFINES CAMPO   { exit(0); }
       |  NUMBER CAMPO OCCURS NUMBER
                { ocurr = $4; nivel = $1; flag = 1;   }
       |  NUMBER CAMPO PIC              {   imprcampo( $1 );   }
       |  '.'
       ;
%%


main()
{
  yyparse();
}

imprcampo( numero )
int numero;
{
    if (flag)
       if ( numero > nivel)
          printf("%s %d\n", campo, ocurr);
       else {
          flag = 0;
          printf("%s\n", campo);
       }
    else
       printf("%s\n", campo);
}

#include "lex.yy.c"
```

El programa leerá la descripción del archivo (FD) y dejará un archivo de salida, con el nombre de cada uno de los campos del archivo, así como el número de ocurrencias en los campos que son arreglos. Por ejemplo:

```
CONSEC1-DESCONT-FD
NIVEL3-DESCONT-FD 10
TIPODATO-DESCONT-FD 10
NUMDECIM-DESCONT-FD
MOVNEGAT-DESCONT-FD
```

Cada campo va en una línea, en mayúsculas. Si el campo es un arreglo, a continuación va el número de ocurrencias del campo, separados por un espacio.

El programador puede entonces indicar si desea usar una cifra de control sobre cierto campo, colocando al final de la línea una letra F o f, separada por un espacio en blanco Por ejemplo

```
CONSEC1-DESCONT-FD F
NIVEL3-DESCONT-FD 10 F
TIPODATO-DESCONT-FD 10
NUMDECIM-DESCONT-FD F
MOVNEGAT-DESCONT-FD
```

Se diseñó un script (genindsq) en base a awk que lee esta información y genera el programa COBOL que convierte el archivo indexado a secuencial en NOS/VE y calcula cifras de control. Otro script (gensqind) genera el programa en COBOL que convierte el archivo secuencial en indexado en RM/COBOL y calcula cifras de control.

4.3 Conversión de programas batch de COBOL NOS/VE a RM/COBOL.

En general, los programas batch fueron los más fáciles de convertir debido a que no manejan pantallas. Para facilitar la tarea del programador, se seleccionaron sobre todo los cambios más repetitivos que tenían que efectuarse en los programas en COBOL NOS/VE para convertirlos a RM/COBOL y que por lo tanto le llevarían más tiempo al programador, para que un programa los efectuara automáticamente. Otros cambios, sobre todo en programas fuera de los estándares mencionados anteriormente, se dejaron a cargo del programador.

Entre los cambios que se decidió hacerlos automáticamente están:

1. Borrar las instrucciones de manejo de bloqueo de registros de CYBER y convertir a su equivalente en RM/COBOL. El programador debe revisar que todos los archivos que se usarán solo para consulta, se abran en el modo de apertura de entrada (INPUT).

2. Se deben convertir los comandos de SCL que son llamados desde el programa a sus equivalentes en Korn shell en UNIX.

Como los comandos ejecutados en SCL son muy distintos de los manejados en Korn shell, se sugirió revisar la función que efectuaban en las llamadas a comandos

de SCL mediante la rutina "$SCAN_COMMAND_LINE" y hacer un script en UNIX que hiciera lo mismo.

Una diferencia que se encontró entre COBOL NOS/VE y RM/COBOL es que mientras que en NOS/VE se puede modificar una variable de SCL desde un programa en COBOL mediante la rutina "$SCAN_COMMAND_LINE", en RM/COBOL no es posible hacer lo mismo mediante la rutina "SYSTEM", pues cada llamada a esta rutina se ejecuta mediante un comando: "sh *comando*", el cual no puede modificar una variable del ambiente permanentemente, sino solo lo modifica localmente en el ambiente del comando ejecutado. Una forma en la cual un programa en RM/COBOL puede comunicar un valor de una variable a otros programas o al ambiente es escribiendo el valor de la variable a un archivo, para que después otro programa pueda leerlo y usarlo, o se pueda leer desde Korn shell.

3. Cambiar: `ORGANIZATION IS SEQUENTIAL`
   a: `ORGANIZATION IS LINE SEQUENTIAL`

4. Si la cláusula SELECT está en la columna 7 (área indicadora) o en la zona A (columnas 8 a la 11), hay que pasarla a la zona B (a partir de columna 12).

5. Poner entre comillas el nombre del archivo en la instrucción ASSIGN; la manera de designar los archivos varía según el sistema operativo.
   Cambiar: `SELECT ARCHIVO ASSIGN TO ARCHIVO-FISICO`
   a: `SELECT ARCHIVO ASSIGN TO "ARCHIVO-FISICO"`

6. Cambiar la prueba de condición de clase:
   `NOT NUMERIC`
   a: `IS NOT NUMERIC`

7. Cambiar la llamada a la rutina "$SCAN_COMMAND_LINE" por la llamada a la rutina "SYSTEM".

Se tomó como norma para la conversión convertir los datos numéricos en modo COMPUTATIONAL a modo DISPLAY, por lo que hay que borrar la palabra "COMP".

Aunque en NOS/VE se cuenta con la utilería EDIT_FILE, el cual es un editor programable, este no puede reconocer expresiones regulares, las cuales son muy útiles para realizar los cambios anteriores. En cambio, en UNIX se cuenta con varias herramientas útiles que reconocen expresiones regulares e incluso gramáticas, como `sed` y YACC. Además, el equipo CYBER se encuentra normalmente muy cargado. Por lo tanto, se decidió realizar la conversión de los programas batch e interactivos en el equipo HP. Para esto, se requiere transferir los programas batch al equipo HP.

4.3.1 Desarrollo de un programa convertidor de programas batch de COBOL NOS/VE a RM/COBOL.

Se estudiaron los patrones a buscar en COBOL NOS/VE y se determinó su equivalente en RM/COBOL.

Una herramienta poderosa mediante la cual se pueden hacer los cambios es sed.

Mediante sed se pueden buscar cadenas fijas y expresiones regulares y traducir por su equivalente en RM/COBOL.

A continuación se muestra cómo se realizaron algunos de estos cambios mediante comandos de sed:

1. Borrar las instrucciones de manejo de bloqueo de registros de CYBER y convertir a su equivalente en RM/COBOL. Para borrar las instrucciones de manejo de bloqueo de registros en NOS/VE, se usaron comandos como:

```
/CBP$SET_KEY_LOCKING_OPTIONS/d
/"NO_LOCK"/d
/"WAIT_FOR_LOCK"/d
/"EXCLUSIVE_ACCESS"/d
/ CDM-STATUS-LOCK/d
```

Estos comandos "borran" las líneas que contengan esas cadenas, al no escribirlas en la salida estándar.

Como el bloqueo de registros se realiza mediante la instrucción READ de COBOL y hay un copy diferente que se tiene para cada archivo, solamente se modificaron estos copys sin afectar el programa.

2. Se deben convertir los comandos de SCL que son llamados desde el programa a sus equivalentes en Korn shell de UNIX. Por ejemplo:

```
s/ COPF / cp /
s/ PRIF / lp /
```

En otros casos, cuando se mandaba llamar un procedimiento en SCL, era necesario hacer un script en shell que hiciera lo mismo.

3. Cambiar: `ORGANIZATION IS SEQUENTIAL`
   a: `ORGANIZATION IS LINE SEQUENTIAL,`
   se logra mediante el comando:

```
s/ *SEQUENTIAL / LINE SEQUENTIAL /
```

4. Si la cláusula SELECT está en la columna 7 (área indicadora) o en la zona A, hay que pasarla a la zona B.

```
s/^      *SELECT /          SELECT /
```

5. Poner entre comillas el nombre del archivo en la instrucción ASSIGN:

```
s/ ASSIGN  *TO  *\([A-Z0-9-][A-Z0-9-]*\)/ ASSIGN TO RANDOM "\1"/
```

6. Cambiar la prueba de condición de clase:

```
     NOT NUMERIC
a:   IS  NOT NUMERIC
     s/\([^I][^S]\)   *NOT   *NUMERIC/\1 IS NOT NUMERIC/
```

7. Cambiar la llamada a la rutina "$SCAN_COMMAND_LINE" por la llamada a la rutina "SYSTEM":

```
     s/CLP$SCAN_COMMAND_LINE/SYSTEM/g
```

Se diseñó un script en base a sed para realizar la conversión de programas batch.

4.4 Conversión de pantallas de Screen Formatting a ASEMAP.

Es necesario convertir todas las pantallas desarrolladas en el Design Screen de NOS/VE al formato leído por ASEMAP.

Primeramente, se obtendrá una lista de todas las pantallas usadas por los programas interactivos.

Hay que revisar la manera en que se guarda la información acerca de las pantallas en SCRF y extraer esa información. Después se diseñará un programa que lea esa información y convierta la pantalla al formato de ASEMAP.

4.4.1 Desarrollo de programas para la conversión de pantallas de programas interactivos en Screen Formatting a la utilería ASEMAP en UNIX.

Se desarrolló un programa en CYBIL (el lenguaje de implementación de NOS/VE) para leer toda la descripción de una pantalla, incluyendo constantes, variables, sus atributos, posición de inicio y longitud en la pantalla, etc. Este llama a diferentes funciones y procedimientos de CYBIL para obtener los objetos que contiene la forma y luego para encontrar las propiedades de los objetos de texto (letreros fijos) y de las variables de entrada y/o salida, y escribe la salida a un archivo. Por ejemplo, la llamada al procedimiento:

```
FDP$GET_VARIABLE_ATTRIBUTES(FORM_ID, FORM_OBJECTS[I].NAME,
                            GET_VAR_ATT,STATUS);
```

obtiene información seleccionada accerca de la variable cuyo nombre es FORM_OBJECTS[*i*] y los deja en el registro de atributos de variable GET_VAR_ATT, cuyos campos almacenan la longitud de la variable en caracteres, el formato de entrada de datos (carácter, entero, etc.), el tipo de transferencia de entrada y salida (entrada, salida o entrada/salida), el formato de salida y la longitud de la salida formateada (carácter, entero, real en notación de punto fijo o real en notación de punto flotante) y otra información. Las siguientes líneas:

```
IF FDC$INVERSE_VIDEO IN GET_OBJ_ATT[3].DISPLAY_ATTRIBUTE THEN
  OUTPUT_LINE(39,1) := '2'
ELSE
  OUTPUT_LINE(39,1) := '1'
IFEND;
```

escriben en la columna 39 del archivo DECODE_SCR un carácter "2" si está habilitado el atributo de video inverso en la variable, o un carácter "1" si no lo está.

El programa fuente se llama EXTSCR y el programa ejecutable se dejó con el nombre BIN_EXTSCR.

El programa lee del archivo LIST_SCR el nombre de todas las pantallas que se van a convertir, una por línea. Por ejemplo:

```
PANTA_00880
PANTA_00881
```

En base a la librería objeto que contiene las pantallas:

```
:asemex9.control.pantallas.screen_library
```

obtiene toda la información necesaria y deja la descripción de las pantallas en un archivo de salida (DECODE_SCR).

Este archivo se transmite al equipo HP al directorio de pantallas y entonces se corre un programa en lenguaje "C" que lee la descripción de las pantallas y genera los archivos con la descripción de la pantalla que usa ASEMAP: PANT0880.scrn, PANT0881.scrn, etc.

Por ejemplo, si se tiene la pantalla PANTA_99998, cuya descripción en SCRF es la siguiente:

```
01  PANTA-99998.
    03  CDM-TAB-ARRCAD-F OCCURS 3.
        05  CDM-ARRCAD-F PIC X(8).
    03  CDM-ENTERO-F PIC S9(18) COMP SYNC LEFT.
    03  CDM-REAL-F COMP-1.
    03  CDM-CADENA-F PIC X(8).
```

Al correr el programa BIN_EXTSCR, se generó como salida en el archivo decode_scr:

```
FORM    PANTA_99998
CONSTANTS
                                  320 114     X
                                  411 614     ENTERO
                                  611 414     REAL
                                  811 614     CADENA
                                 1011 614     ARRCAD
VARIABLES
CDM-ENTERO-F                      420 513     I
CDM-REAL-F                        620 613     R F2
CDM-CADENA-F                      820 813 C   A
CDM-ARRCAD-F                     1020 813 C   A       3
END
```

Al inicio de la descripción de cada pantalla viene la palabra FORM, seguida del nombre de la pantalla. Luego, viene la sección de descripción de las constantes (letreros constantes de la pantalla) y la sección de descripción de las variables, describiendo la posición del letrero o variable en la pantalla (renglón, columna), su longitud, número de atributo, tipo de variable, número de ocurrencias, etc.

A continuación se indican los datos que guarda cada columna:

CONSTANTES Y VARIABLES

Inicio	***Longitud***	***Contenido***
33	2	Renglón en que se encuentra la constante o variable
35	2	Columna en que se encuentra la constante o variable
37	2	Longitud de la constante o variable
39	2	Atributo de despliegue de la constante o variable

CONSTANTES

Inicio	***Longitud***	***Contenido***
45		Texto del letrero constante

VARIABLES

Inicio	***Longitud***	***Contenido***
45	1	Tipo de variable: A (alfanumérica), I (entero), R (real), M (moneda)
51	1	Número de ocurrencias

Al transmitir este archivo de salida al equipo HP y correr el programa:

```
cvasemap decode_scr
```

se generó el copy para ASEMAP `PANT99998.scrn:`

```
*****************************************************************
*                          SCREEN PANT9998                      *
*****************************************************************
 01   SCREEN-PANT9998.
  02  FILLER PIC X(118) VALUE
      "PANT9998.scrn-47X-'.ENTERO-).REAL-+ CADENA-~.ARRCAD-###J#G??
-     "'7((#11#N??)7)(%11##??+7++#AN#.??-7++#AN#5??.7++#AN#>??/7+".
  02  FILLER PIC X(004) VALUE
      "+#AN".
  02  FILLER PIC X VALUE LOW-VALUES.
*----------------------------------------------------------------
 01   VARIABLES-PANT9998.
      03 VAR-ALPHA-PANT9998.
         05 CADENA-P                          PIC X(01)  VALUE "3".
         05 CADENA-F                          PIC X(08)  VALUE SPACES.
         05 ARRCAD-F-INI.
            07 ARRCAD-F-VEC OCCURS 03 TIMES.
               09 ARRCAD-P                    PIC X(01) VALUE "3".
               09 ARRCAD-F                    PIC X(08) VALUE SPACES.
      03 VAR-NUM-PANT9998 SIGN TRAILING SEPARATE.
         05 ENTERO-P                          PIC X(01)  VALUE "3".
         05 ENTERO-F                          PIC S9(05) VALUE ZEROES.
         05 REAL-P                            PIC X(01)  VALUE "3".
         05 REAL-F                            PIC S9(03)V9(02)
                                              VALUE ZEROES .
      03 FILLER PIC X VALUE LOW-VALUES.
*****************************************************************
```

El convertidor quita el prefijo "CDM-" del nombre de los campos, y crea para cada campo <*campo*> de la pantalla, las variables <*campo*>-P y <*campo*>-F, que guardan su atributo y su valor, respectivamente.

4.5 Conversión de programas interactivos de COBOL NOS/VE a RM/COBOL.

El manejo de las pantallas en los programas interactivos de COBOL NOS/VE es muy distinto que al usar la utilería ASEMAP en RM/COBOL. Esto implica el realizar el cambio de los valores de atributos adecuados y el cambiar los nombres de los campos de las pantallas en NOS/VE a sus correspondientes en ASEMAP.

Se seleccionaron sobre todo los cambios más frecuentes para que un programa convertidor los efectuara automáticamente. Otros cambios se dejaron al programador el revisarlos y convertirlos.

Como se pudo observar de la comparación del manejo de pantallas entre SCRF y ASEMAP realizado anteriormente, los valores de los atributos son distintos. Para saber cuáles atributos son equivalentes entre SCRF y ASEMAP, se hizo un programa de prueba en el que utilizaron todos los atributos disponibles en SCRF; se convirtió el programa a RM/COBOL y se diseño una pantalla similar, buscándose los atributos correspondientes de ASEMAP. Se encontró la siguiente equivalencia de atributos entre ASEMAP y SCRF y se le asignó un nombre al valor del atributo en ASEMAP:

SCRF	*ASEMAP*	*Nombre en RM/COBOL*	*Descripción*
01	1	BRI-MOD	Alta intensidad, modificable
02	B	BRI-PRO	Alta intensidad, protegido
03	5	OPA-MOD	Baja intensidad, modificable
04	F	OPA-PRO	Baja intensidad, protegido
05	6	DES-MOD	Baja intensidad parpadeante, modificable
06	G	DES-PRO	Baja intensidad parpadeante, protegido
07	0	OCU-MOD	Oculto, modificable
08	A	OCU-PRO	Oculto, protegido
09	3	BRI-MODI	Alta intensidad con video inverso, modificable
10	D	BRI-PROI	Alta intensidad con video inverso, protegido
11	7	OPA-MODI	Baja intensidad con video inverso, modificable
12	H	OPA-PROI	Baja intensidad con video inverso, protegido
13	4	DES-MODI	Baja intensidad con video inverso y parpadeante, modificable
14	E	DES-PROI	Baja intensidad con video inverso y parpadeante, protegido
15	0	OCU-MODI	Oculto, con video inverso, modificable
16	A	OCU-PROI	Oculto, con video inverso, protegido

Es necesario traducir el valor en SCRF por su nombre correspondiente en ASEMAP.

Ya no es necesario usar el verbo COPY CDM_RUT_SCREEN_<*PROGRAMA*> dentro del programa, para el manejo de las pantallas, puesto que los valores del campo y del atributo se pasan directamente a las variables correspondientes en ASEMAP.

Tampoco es necesario usar una variable que contenga los valores de los atributos que se pasarán a la pantalla (FACOF-*CAMPO*) puesto que se puede mover directamente el valor del atributo en la variable que lo almacena en ASEMAP. Por lo tanto, es necesario convertir el nombre de la variable que almacena el valor del atributo en SCRF (FACOF-*CAMPO*) por el nombre de la variable correspondiente en ASEMAP que almacena el valor del atributo (*CAMPO*-P). Una forma de encontrar el nombre correspondiente, es analizar el copy CD_RUT_SCREEN_<*PROGRAMA*> . Se encontraron los siguientes casos:

1. Normalmente, si se tiene el campo de la pantalla CDM-*CAMPO*-F, el valor del campo lo almacena la variable *CAMPO*-F y el valor del atributo lo almacena la variable FACOF-*CAMPO*-P. Por ejemplo:

Campo de la pantalla: CDM-ENTERO-F
Valor del campo: ENTERO-F
Valor del atributo: FACOF-ENTERO-P

Su manejo en el copy CDM_RUT_SCREEN_PRUEBA es el siguiente:

```
IF FACOF-ENTERO-P = 7 OR 8
    MOVE ZEROES TO CDM-ENTERO-F
ELSE
    MOVE ENTERO-F TO CDM-ENTERO-F
END-IF
...
MOVE "CDM-ENTERO-F" TO CDM-NOMBRE-CAMPO
MOVE FACOF-ENTERO-P TO CDM-NUMERO-FACOF
ADD 8 TO CDM-NUMERO-FACOF
MOVE 1 TO CDM-OCURRENCIA
PERFORM CDM-SET-OBJECT-ATTRIBUTE
...
MOVE CDM-ENTERO-F TO ENTERO-F
```

2. El nombre del campo de la pantalla no corresponde con los nombres de las variables que almacenan el valor del atributo y el valor del campo dentro del programa. Un ejemplo lo hallamos en el copy CDM_RUT_SCREEN_CAPGASTO, donde:

Campo de la pantalla: CDM-CCTAN11-03385-F
Valor del campo: CCTAN11-MOVSDIA3-F
Valor del atributo: FACOF-CCTAN11-MOVSDIA3-P

Su manejo en el copy CDM_RUT_SCREEN_CAPGASTO es el siguiente:

```
IF CDM-PANTALLA-A-DESPLEGAR = 5
IF FACOF-CCTAN11-MOVSDIA3-P =  7 OR 8
   MOVE  SPACES  TO CDM-CCTAN11-03385-F
ELSE
   MOVE CCTAN11-MOVSDIA3-F TO CDM-CCTAN11-03385-F
END-IF
...
MOVE "CDM-CCTAN11-03385-F" TO CDM-NOMBRE-CAMPO
MOVE FACOF-CCTAN11-MOVSDIA3-P TO CDM-NUMERO-FACOF
ADD 8 TO CDM-NUMERO-FACOF
MOVE 1 TO CDM-OCURRENCIA
PERFORM CDM-SET-OBJECT-ATTRIBUTE
...
MOVE CDM-CCTAN11-03385-F TO CCTAN11-MOVSDIA3-F
```

3. La misma variable en el programa en COBOL se usa para almacenar el valor del campo o el valor del atributo de campos de diferentes pantallas. Un ejemplo lo hallamos en el copy CDM_RUT_SCREEN_CAPGASTO, donde la variable CORRECTO almacena el valor de los campos CDM-CORRECTO-01490 y CDM-CORRECTO-03385. Así, para la pantalla PANTA_01490:

Campo de la pantalla: CDM-CORRECTO-01490
Valor del campo: CORRECTO
Valor del atributo: FACOF-CORRECTO-MOVSDIA

Para la pantalla PANTA_03385 se tiene:

Campo de la pantalla: CDM-CORRECTO-03385
Valor del campo: CORRECTO
Valor del atributo: FACOF-CORRECTO-MOVSDIA3

Su manejo en el copy CDM_RUT_SCREEN_CAPGASTO es el siguiente:

Para la pantalla PANTA_01490:

```
IF CDM-PANTALLA-A-DESPLEGAR = 1
IF FACOF-CORRECTO-MOVSDIA =  7 OR 8
    MOVE  SPACES  TO CDM-CORRECTO-01490
ELSE
    MOVE CORRECTO TO CDM-CORRECTO-01490
END-IF

MOVE "CDM-CORRECTO-01490" TO CDM-NOMBRE-CAMPO
```

```
MOVE FACOF-CORRECTO-MOVSDIA TO CDM-NUMERO-FACOF
MOVE 1 TO CDM-OCURRENCIA
PERFORM CDM-SET-OBJECT-ATTRIBUTE
...
MOVE CDM-CORRECTO-01490 TO CORRECTO
```

Para la pantalla PANTA_03385:

```
IF CDM-PANTALLA-A-DESPLEGAR = 5
IF FACOF-CORRECTO-MOVSDIA3 =  7 OR 8
    MOVE  SPACES  TO CDM-CORRECTO-03385
ELSE
    MOVE CORRECTO TO CDM-CORRECTO-03385
END-IF
```

Por lo tanto, además de los cambios ya analizados en la conversión de los programas batch de COBOL NOS/VE a RM/COBOL, es necesario hacer automáticamente los siguientes cambios, debido al manejo de pantallas de ASEMAP:

1. Convertir el valor del atributo en SCRF por su nombre correspondiente en ASEMAP y convertir el nombre de la variable que almacena el valor del atributo en SCRF (FACOF-*CAMPO*) por el nombre de la variable correspondiente en ASEMAP que almacena el valor del atributo (*CAMPO*-P). Por ejemplo:

```
MOVE 01       TO FACOF-OFICINA-03431-P     por
MOVE BRI-MOD  TO OFICINA-03431-P
```

2. Borrar el verbo:

```
COPY CDM_RUT_SCREEN_<PROGRAMA>     OF CDMLIB_DECKS.
```

3. Agregar la línea : `COPY ATRIBUTOS OF COPYLIB.`

con lo que se expanderá el contenido del copy ATRIBUTOS, que contiene la definición de los nombres de los atributos usados para la conversión (BRI-MOD, BRI-PRO, etc.).

4. Borrar el procedimiento CONTROL-SERVER que asigna un número a cada pantalla. Por ejemplo, se borrarían las siguientes líneas:

```
CONTROL-SERVER.
    MOVE 1 TO NUM-PANT
    MOVE 3811 TO PANT(1)
    MOVE 2 TO NUM-PANT
    MOVE 3812 TO PANT(2)
```

5. Agregar abajo de la línea: `COPY EFECOM OF COPYLIB.`
   la siguiente línea: `COPY CLEAR3811 OF CLEAR.`

El copy "CLEAR3811" contiene la rutinas CLEAR-ALPHA-PANT3811 y CLEAR-NUM-PANT3811, las cuales copian el valor SPACES a todas las variables alfanuméricas de la pantalla y copian el valor ZEROS a todas las variables numéricas de la pantalla, respectivamente. Estas dos rutinas limpian la pantalla.

6. Cambiar:
```
MOVE ZEROS  TO NUM-PANT3811
MOVE SPACES TO ALPHA-PANT3811
```
   por:
```
PERFORM CLEAR-NUM-PANT3811
PERFORM CLEAR-ALPHA-PANT3811
```

En SCRF todos los campos numéricos de una pantalla eran abarcados por el campo NUM-<*PANTALLA*> y todos los campos alfanuméricos de una pantalla eran abarcados por el campo ALPHA-<*PANTALLA*>, por lo cual podían ser limpiados con una sola instrucción MOVE. Ahora, en ASEMAP, es necesario tener una rutina que los limpie pues ya no están en forma contigua.

4.5.1 Desarrollo de un programa convertidor de programas interactivos de COBOL NOS/VE a RM/COBOL.

Se diseñó un script en base a sed para realizar la conversión de programas interactivos. Además de los cambios ya analizados en la conversión de los programas batch de COBOL NOS/VE a RM/COBOL, es necesario efectuar varios cambios, debido al manejo de pantallas de ASEMAP. Entre los cambios adicionales están:

1. Sustituir los valores de los atributos en SCRF por su nombre correspondiente en ASEMAP. Esto se logra mediante los siguientes comandos de sed:

```
/MOVE  *[0-9][0-9]  *TO  *FACOF-/{
s/MOVE *01  */MOVE BRI-MOD /
s/MOVE *02  */MOVE BRI-PRO /
s/MOVE *03  */MOVE OPA-MOD /
s/MOVE *04  */MOVE OPA-PRO /
s/MOVE *05  */MOVE DES-MOD /
s/MOVE *06  */MOVE DES-PRO /
s/MOVE *07  */MOVE OCU-MOD /
s/MOVE *08  */MOVE OCU-PRO /
s/MOVE *09  */MOVE BRI-MODI /
s/MOVE *10  */MOVE BRI-PROI /
```

```
s/MOVE *11   */MOVE OPA-MODI /
s/MOVE *12   */MOVE OPA-PROI /
s/MOVE *13   */MOVE DES-MODI /
s/MOVE *14   */MOVE DES-PROI /
s/MOVE *15   */MOVE OCU-MODI /
s/MOVE *16   */MOVE OCU-PROI /
)
```

2. Convertir el nombre de la variable que almacena el valor del atributo en SCRF (FACOF-*CAMPO*) por el nombre de la variable correspondiente en ASEMAP que almacena el valor del atributo (*CAMPO*-P).

3. Convertir, donde sea necesario, el nombre de la variable que almacena el valor del campo (*CAMPO*-F) por el nombre de la variable correspondiente en ASEMAP, cuando no sean los mismos.

Una forma de encontrar el nombre correspondiente a los puntos 2 y 3, es analizar el copy CDM_RUT_SCREEN_<*PROGRAMA*> .

Se diseñó el script `facof` en base a `awk` que analiza el copy CDM_RUT_SCREEN_<*PROGRAMA*> y por cada párrafo de la forma:

```
IF FACOF-ENTERO-P = 7 OR 8
    MOVE ZEROES TO CDM-ENTERO-F
ELSE
    MOVE ENTERO-F TO CDM-ENTERO-F
END-IF
```

en este copy generará en un archivo llamado ***<programa>.fac*** con los comandos de `sed` necesarios para sustituir la variable FACOF-*CAMPO* por la variable correspondiente en ASEMAP (*CAMPO*-P).

A continuación se muestra parte del programa en `awk` que realiza ésto:

```
#  facof
/IF +FACOF[A-Z0-9-]+ += +[0-9]+/ {
    facof=$2
    getline
    getline
    getline
    if ($1 == "MOVE") {
      cdm = $4
      valor = $2
      sub(/^CDM-/, "", cdm)
      cdmv = cdm
```

```
        if (cdm ~ /(-F$|-F-I$|-FK$)/ ) {
            sub(/^CDM-/, "", cdm)
            cdmv = cdm
            sub(/-F$/, "-P", cdm)
            sub(/-F-I$/, "-P-I", cdm)
            sub(/-FK$/, "-PK", cdm)
            cdmf = cdm
            }
        else {
            cdmv = cdm "-F"
            cdmf = cdm "-P"
        }
        printf "s/ %s\([, \.]\)/ %s\\1/g\n", facof, cdmf > "sd_facof"
        printf "s/ %s$/ %s/g\n",       facof, cdmf > "sd_facof"
        if (valor != cdmv) {
            printf "s/ %s / %s /g\n",       valor, cdmv > "sd_cdm"
        }
    }
}
```

Al analizar el copy CDM_RUT_SCREEN_PRUEBA se produce la siguiente salida en el archivo PRUEBA.fac:

```
s/ FACOF-ENTERO-P\([, \.]\)/ ENTERO-P\1/g
s/ FACOF-ENTERO-P$/ ENTERO-P/g
s/ FACOF-REAL-P\([, \.]\)/ REAL-P\1/g
s/ FACOF-REAL-P$/ REAL-P/g
s/ FACOF-CADENA-P\([, \.]\)/ CADENA-P\1/g
s/ FACOF-CADENA-P$/ CADENA-P/g
```

Estos comandos se aplican al programa para sustituir el campo que almacena el atributo del campo por su correspondiente en ASEMAP.

Si hay campos en los que no coincida el campo que guarda el valor en COBOL NOS/VE con el que debería ser al analizar el copy CDM_RUT_SCREEN_*<programa>*, entonces se producen los archivos ***<programa>*.cdmi** y ***<programa>*.cdmd** con tales campos.

El archivo ***<programa>*.cdmi** contiene los comandos de sed para sustituir los nombres de campos de la pantalla que no corresponden con los nombres de las variables que almacenan el valor del atributo y el valor del campo dentro del programa (caso 2 mencionado en la sección 4.5). En el caso del copy CDM_RUT_SCREEN_CAPGASTO:

```
#  capgasto.cdmi
s/ CCTAN11-MOVSDIA3-F / CCTAN11-03385-F /g
```

En el caso de la variable CCTAN11-MOVSDIA3-F se usa para guardar el valor del campo CDM-CCTAN11-03385-F, que no coincide con el nombre CCTAN11-03385-F que se deduce del nombre del campo.

El archivo ***<programa>.cdmd*** nos informa los casos en que la misma variable en el programa en COBOL se use para almacenar el valor del campo o el valor del atributo de campos de diferentes pantallas (caso 3 mencionado en la sección 4.5). En el caso del copy CDM_RUT_SCREEN_CAPGASTO, se tiene:

```
#   capgasto.cdmd
s/ CORRECTO / CORRECTO-01490-F /g
s/ CORRECTO / CORRECTO-03385-F /g
```

En este caso, la variable CORRECTO se usa para guardar el valor de los campos CORRECTO-01490-F y CORRECTO-03385-F. Se deja al programador este archivo para que analice estos casos y realice los cambios necesarios.

4.6 Conversión de procedimientos en SCL de NOS/VE a scripts en Korn shell de UNIX.

Todos los procedimientos diseñados en NOS/VE se escribieron en SCL, el intérprete de comandos de NOS/VE. Se decidió convertir los procedimientos a Korn shell por su rapidez y la variedad de ventajas que dá.

El Korn shell suministra tanto un intérprete de comandos como un lenguaje de programación como interfaz a HP-UX. Usaremos los términos *programas shell* y *scripts shell* como sinónimos, para referirnos a los programas escritos usando el lenguaje de programación shell. El shell suministra un ambiente de programación poderoso y simple de usar y posee un lenguaje estructurado. Los programas shell pueden llamar a cualquier comando de HP-UX e incluso a programas escritos por el propio usuario.

Muchas de los comandos en SCL no tienen equivalente en Korn shell o no se necesitan, por lo que se desarrolló un modelo de cómo se podían convertir los procedimientos de SCL a Korn shell y se dejó al programador la responsabilidad de convertirlos.

4.7 Conversión de procedimientos para la transferencia de archivos y ejecución de comandos remotos desde PCs.

Control Data proporciona el emulador Connect View para poder ejecutar comandos remotos y realizar la transferencia de archivos entre una PC y un equipo con sistema operativo NOS/VE.

Por otro lado, HP suministra los comandos **rcp** y **remsh** (servicios Berkeley) para la transferencia de archivos y ejecución de comandos remotos entre una PC en red y un equipo con sistema operativo UNIX.

Se investigó la función realizada por los comandos que proporciona Connect View y se investigó como lograr lo mismo mediante los comandos rcp y remsh.

5. Implantación y realización de la conversión.

La implantación de utilerías para facilitar la conversión se dividió en las siguientes partes:

a) Desarrollo de generadores de programas en COBOL que conviertan los archivos indexados en secuenciales y obtengan cifras de control.

b) Desarrollo de un programa convertidor de programas batch e interactivos de COBOL NOS/VE a RM/COBOL.

c) Desarrollo de programas para la conversión de pantallas de programas interactivos en CYBER a la utilería ASEMAP en UNIX.

Se diseñaron los scripts y programas en lenguaje C para efectuar estas tareas, como se indicó en el capítulo anterior, usando herramientas como sed.

Para probar las herramientas desarrollada, se usaron algunos archivos y programas batch e interactivos complejos que abarcaran la mayoría de los problemas que se iban a encontrar en todo el sistema.

5.1 Manual del usuario.

La conversión de CYBER a HP consta de los siguientes pasos:

1. Creación de un medio ambiente para realizar la conversión.
2. Conversión de archivos.
3. Conversión de programas batch de CYBER a HP.
4. Conversión de pantallas en Screen Formatting a la utilería ASEMAP en UNIX.
5. Conversión de programas interactivos de CYBER a HP.
6. Conversión de procedimientos de SCL a Korn shell.

NOTA: Se usan las siguientes abreviaturas a lo largo de la documentación:

HOME: Directorio hogar de la cuenta. Ej.: /mnt2/conta

HCOPY: Directorio de copys de la cuenta. Ej.: /mnt2/conta/cop

HUTIL: Directorio con los programas fuentes generados para la conversión. Ej.: /mnt2/conta/util

RUNPATH: Directorio donde se encuentran los programas ejecutables de RM/COBOL. Ej.: /mnt2/conta/obj

Antes de realizar la conversión del sistema es necesario realizar un análisis del sistema actual que nos permita realizar con mayor facilidad la conversión. Se necesita realizar las siguientes actividades:

1. Identificar los archivos que usa el sistema, tanto propios como de otros sistemas.

2. Obtener una lista de los programas batch que forman realmente parte del conjunto de programas de producción (ciclo diario, ciclo mensual y procesos mensuales).
3. Obtener una lista de los programas interactivos que están en producción, así como de los subprogramas y subrutinas relacionadas.
4. Generar una matriz de programas/archivos que nos permita saber qué archivos accesa cada programa y la forma de acceso (exclusivo, sólo lectura o total).
5. Imprimir un listado de los procedimientos en SCL de los ciclos diarios, mensuales y procesos especiales, para conocer el funcionamiento del sistema.
6. Imprimir la descripción (FD) de todos los archivos, así como por cada archivo obtener la siguiente información:
a) Tamaño del registro.
b) Número de llaves (contando la llave primaria) y por cada llave: posición de inicio de la llave, longitud de la llave y, si es una llave alterna, si acepta duplicados o no.

Es conveniente guardar toda esta documentación en una carpeta, pues se usará frecuentemente durante la conversión.

1. Creación de un medio ambiente para realizar la conversión.

Se creó para el sistema una cuenta en el equipo CYBER (CONVCONT) y en el equipo HP (dconta) para allí llevar a cabo la conversión.

En el equipo HP, en lugar de bibliotecas de fuentes se usan directorios en los cuales se ponen los copys de cada archivo correspondientes a cada una de las librerías.

/mnt2/conta/cop/SGFD/POLCAPT
/mnt2/conta/cop/SGSELECT/POLCAPT
...

Esto se realizará por cada librería y por cada archivo que use el sistema.

2. Conversión de archivos.

Consta de los siguientes pasos:

a) Conversión de archivos indexados a secuenciales en CYBER.
b) Transferencia de los archivos mediante ftp en modo binario al equipo HP.
c) Conversión de archivos secuenciales a indexados en RMCOBOL.

a) Conversión de archivos indexados a secuenciales en CYBER.

Para generar el programa que convierte los archivos indexados a secuenciales, siga los siguientes pasos:

1. Genere los FDs de los archivos secuenciales en el directorio $HCOPY/SGFDSEQ en el equipo HP.

2. Genere los nuevos FDs de los archivos indexados en RM/COBOL, recalculando la longitud de las llaves si estaban formadas por campos numéricos en modo COMPUTATIONAL; quite el signo a todos los campos numéricos que no sean importes.

3. Extracción de los campos de los archivos a convertir.

Es necesario extraer, del FD de cada uno de los archivos, los campos del archivo para poder crear programas que copien cada uno de los campos de los archivos indexados a los archivos secuenciales en CYBER y obtengan cifras de control.

Se creó el directorio $HCOPY/campo, este contendrá por cada archivo un archivo del mismo nombre con los nombres de los campos a convertir y una indicación de si se calcularán cifras de control sobre cierto campo.

Para obtener este archivo, desde el directorio $HCOPY teclee:

campo < fd_cyber/*ARCHIVO* > campo/*ARCHIVO*

El archivo **$HCOPY/campo/*ARCHIVO*** contiene todos los campos del archivo, así como el número de ocurrencias en los campos que son arreglos.

Cada campo va en una línea, en mayúsculas. Para indicar que desea calcular cifras de control sobre cierto campo en particular, coloque al final de la línea, separado por un espacio en blanco una **F** o **f**. Por ejemplo:

```
NOMBRE
IMPORTE1   14   F
IMPORTE2   F
```

4. Para generar el programa COBOL que convierte el archivo indexado en secuencial y calcula cifras de control en CYBER, ejecute el comando:

genindsq *<archivo>*

Por ejemplo: genindsq polcapt

Esto genera el programa **$HUTIL/indsq*<archivo>***.

Copie con ftp este programa al equipo CYBER al catálogo **.<sistema>.conv.fte** y compílelo.

5. Para generar el programa COBOL que convierte los archivos secuenciales a indexados en RM/COBOL y calcula cifras de control, teclee:

gensqind <archivo>

Por ejemplo: gensqind polcapt

Este programa genera el programa objeto **$RUNPATH /sqind<*archivo*>.cob**

b) Transferencia de los archivos mediante ftp en modo binario al equipo HP.

Entre al equipo HP y posiciónese en su directorio de datos y teclee
transf <*archivo*>_seq <equipo>

donde <*equipo*> es el nombre del equipo desde donde se traerá la información (asemex1 o asemex2). Ejemplo:
transf movsdia_seq asemex2

Esto transfiere el archivo MOVSDIA_SEQ del directorio .<*sistema*>.conv.arch en el equipo ASEMEX2 al equipo HP y borra el archivo secuencial en la CYBER si el archivo fue transferido correctamente. Se deja una bitácora de los archivos transferidos y los errores que hubo en el archivo **log_transf**

c) Conversión de archivos secuenciales a indexados en RMCOBOL.

Para correr este programa, necesitar cambiarse al directorio donde se encuentran los archivos y ejecutar el comando:
runcobol sqind<*archivo*>
Por ejemplo: runcobol sqindmovsdia

Las cifras de control para el archivo se van agregando al archivo **log**.

3. Conversión de programas batch de CYBER a HP.

Es necesario transferir el programa batch al directorio $HOME/fte/batch con el nombre del programa en mayúsculas.
Para convertirlo, teclee:
convb PROCCAR
El programa original queda con el nombre PROCCAR.ori

4. Conversión de pantallas en Screen Formatting a la utileria ASEMAP en UNIX.

Es necesario hacer una lista de todas las pantallas usadas por cada programa del sistema para poder convertirlas.
Por ejemplo, si se tiene la pantalla PANTA_09998, cuya descripción en SCRF es la siguiente:

```
01   PANTA-99908.
     03   CDM-TAB-ARRCAD-F OCCURS 3.
          05   CDM-ARRCAD-F PIC X(8).
     03   CDM-ENTERO-F PIC S9(18) COMP SYNC LEFT.
     03   CDM-REAL-F COMP-1.
     03   CDM-CADENA-F PIC X(8).
```

Para convertir la pantalla efectúe los siguientes pasos:

a) Posicionarse en el catálogo $USER.SALIDAS en CYBER
Edite el archivo LIST_SCR y escriba el nombre de todas las pantallas que se van a convertir, una por línea. Por ejemplo:
PANTA_09998

b) Ejecutar los comandos:
crecle :asemex9.control.pantallas.screen_library
exet .soporte.procedimientos.bin_extscr

El programa lee de la libreria objeto que contiene las pantallas:
:asemex9.control.pantallas.screen_library
toda la información necesaria y deja la descripción de las pantallas en un archivo de salida (DECODE_SCR).

Al correr el programa BIN_EXTSCR, se generó como salida en el archivo decode_scr:

```
FORM     PANTA_99098
CONSTANTS
                                  320 114      X
                                  411 614      ENTERO
                                  611 414      REAL
                                  811 614      CADENA
                                 1011 614      ARRCAD
VARIABLES
CDM-ENTERO-F                      420 513      I
CDM-REAL-F                        620 613      R F2
CDM-CADENA-F                      820 813 C    A
CDM-ARRCAD-F                     1020 813 C    A       3
END
```

Este archivo se debe transmitir al equipo HP al directorio $HOME/pant

c) Este archivo se debe transmitir al equipo HP al directorio $HOME/pant y entonces se corre un programa en lenguaje "C" que lee la descripción de la pantalla y genera el archivo con la descripción de la pantalla que usa ASEMAP: PANT9998.scrn.

Al transmitir el archivo decode_scr al equipo HP y correr el programa:

```
cd $HOME/pant
cvasemap  decode_scr
```

se generó el copy para ASEMAP PANT9998.scrn:

```
*****************************************************************
*                          SCREEN PANT9998                      *
*****************************************************************
 01  SCREEN-PANT9998.
  02 FILLER PIC X(118) VALUE
     "PANT9998.scrn~&7X~'.ENTERO~).REAL~+.CADENA~~.ARRCAD~###J#G??
-    "'7((#11#N??)7)(%11##??+7++#AN#,??-7++#AN#5??.7++#AN#>??/7+".
  02 FILLER PIC X(004) VALUE
     "+#AN".
  02 FILLER PIC X VALUE LOW-VALUES.
*----------------------------------------------------------------
 01  VARIABLES-PANT9998.
     03 VAR-ALPHA-PANT9998.
        05 CADENA-P                          PIC X(01)   VALUE "3".
        05 CADENA-F                          PIC X(08)   VALUE SPACES.
        05 ARRCAD-F-INI.
           07 ARRCAD-F-VEC OCCURS 03 TIMES.
              09 ARRCAD-P                    PIC X(01) VALUE "3".
              09 ARRCAD-F                    PIC X(08) VALUE SPACES.
     03 VAR-NUM-PANT9998 SIGN TRAILING SEPARATE.
        05 ENTERO-P                          PIC X(01)   VALUE "3".
        05 ENTERO-F                          PIC S9(05) VALUE ZEROES.
        05 REAL-P                            PIC X(01)   VALUE "3".
        05 REAL-F                            PIC S9(03)V9(02)
                                             VALUE ZEROES .
     03 FILLER PIC X VALUE LOW-VALUES.
*****************************************************************
```

5. Conversión de programas interactivos de CYBER a HP.

Esto abarca la extracción de los programas interactivos y el copy:

CDM_RUT_SCREEN_<PROGRAMA>

de cada programa. Después se corre el programa que realiza la conversión del programa interactivo.

Para convertir los programas interactivos es necesario efectuar los siguientes pasos:

1. Transferir el programa interactivo al directorio **/mnt2/conta/fte/inter** con el nombre del programa en mayúsculas.

2. Extraer el copy CDM_RUT_SCREEN_<programa> de **:asemex9.*<sistema>*.fuentes.cdmlib_decks** y transferirlo con ftp al directorio **/mnt2/conta/fte/inter/cdmrs** con el nombre del programa en mayúsculas:

$HOME/fte/inter/cdmrs/CAPGASTO

3. Para convertir un programa interactivo, posiciónese en el directorio **$HOME/fte/inter** y teclee:

convi CAPGASTO

El programa original queda con el nombre CAPGASTO.ori

Es conveniente compilar con debug el programa original en CYBER con las librerías fuente de producción para que en el listado queden expandidos los copys y se tenga el listado completo del programa, transferirlo a HP con el nombre **l_<programa>**. De esa manera se puede saber cómo se estaba usando cualquier variable o si existen variables que ya no se usan.

Si hay campos en los que no coincida el campo que guarda el valor en COBOL NOS/VE con el que debería ser al analizar el copy CDM_RUT_SCREEN_*<programa>*, entonces se producen los archivos ***<programa>*.cdmi** y ***<programa>*.cdmd** con tales campos.

El archivo ***<programa>*.cdmd** nos informa los casos en que la misma variable en el programa en COBOL se use para almacenar el valor del campo o el valor del atributo de campos de diferentes pantallas. En el caso del copy CDM_RUT_SCREEN_CAPGASTO, se tiene:

```
#   capgasto.cdmd
s/ CORRECTO / CORRECTO-01490-F /g
s/ CORRECTO / CORRECTO-03385-F /g
```

En este caso, la variable CORRECTO se usa para guardar el valor de los campos CORRECTO-01490-F y CORRECTO-03385-F. Se deja al programador este archivo para que analice estos casos y realice los cambios necesarios.

5. Conclusiones.

La metodología desarrollada permitió realizar con éxito la conversión del sistema de contabilidad en ASEMEX. Al correr pruebas en paralelo, se comprobó que el sistema convertido entregaba los mismos resultados que el sistema en CYBER. Las medidas para comprobar la conversión apropiada de archivos y de programas, permitieron que la conversión fuera exitosa.

Las utilerías diseñadas para facilitar la conversión facilitaron bastante a los programadores el realizar la conversión de archivos, pantallas y programas, y les permitieron dedicarse a los detalles que estaban fuera de estándares.

Esto nos ayuda a ver que el seguir una metodología nos permitirá ir en el camino correcto cuando es necesario realizar conversiones de sistemas.

GLOSARIO DE TERMINOS

Tiempo de respuesta

Es el tiempo de regreso de un sistema interactivo, y a menudo se define como el tiempo transcurrido desde que el usuario presiona la tecla de ENTER hasta que el sistema comienza a imprimir una respuesta.

Bloqueo de archivos y registros (file and record locking)

Técnicas para el manejo de datos en un ambiente multiusuario. La protección de un archivo impide a los usuarios tener acceso al archivo. La protección de un registro prohibe el acceso a un único registro dentro de un archivo de datos.

La protección de archivos y registros se inicia sobre la base de que el primero en llegar es el primero en ser servirdo. El primer usuario que accede al archivo o registro impide, o cierra, el acceso a otros usuarios. Después de que el archivo o registro quede actualizado, es abierto para que otros puedan acceder a él.

Secuencia de ordenación o intercalación (collating sequence)

Secuencia en la cual los caracteres son ordenados para propósitos de ordenación y comparación.

Archivo ASCII (ASCII file)

Archivo de datos o de texto que contiene caracteres codificados en ASCII. Sólo los primeros 128 caracteres (0 - 127) dentro de las 256 combinaciones de un byte conforman el estándar ASCII. El resto son utilizados en forma diferente dependiendo del computador.

Programa interactivo (interactive program)

Un programa es interactivo cuando se desarrolla a través de un diálogo constante entre el programa y el usuario. El instrumento para este diálogo suele ser la pantalla y el teclado.

Programa por lotes o grupos (batch program)

Programa por lote o trabajo por lote se refiere a un programa que procesa un conjunto entero de datos o un grupo de transacciones de una sola vez, tal como un programa de informes o de clasificación. Las operaciones por lote son también llamadas operaciones fuera de línea (offline).

Las transacciones se reúnen y se actualizan los archivos maestros al final del día o en algún otro periodo de tiempo.

Los sistemas de información usualmente combinan el procesamiento por lotes con el procesamiento por transacciones. Por ejemplo, en un sistema de procesamiento de pedidos, el procesamiento por transacciones es la actualización

continua de los archivos de clientes y de inventario a medida que se introduzcan los pedidos.

Al final del día, los programas de procesamiento por lote generan listas de despacho para el almacén. Al final de la semana o de algún otro período, los programas de lotes imprimen facturas e informes generales.

BIBLIOGRAFIA

1) Aho, A. V.; Hopcroft, J. E. y Ullman, J. D. *Estructuras de datos y algoritmos*, Addison-Wesley Iberoamericana, U.S.A., 1988.

2) *COBOL for NOS/VE: Usage*. Control Data Corporation.

3) Deitel, Harvey M. *Introducción a los sistemas operativos.* Addison-Wesley Iberoamericana. México, 1987.

4) Pressman, R. S. *Ingeniería del software: un enfoque práctico.* McGraw-Hill, España, 1989, 2a. edición.

5) Philippakis, Kazmier. *COBOL avanzado.* 2a. ed. McGraw-Hill.

6) *RM/COBOL-85 Language Reference Manual, versión 5.* Liant Software.

7) *RM/COBOL-85 User's guide, versión 5.1 for UNIX.* Liant Software.