CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL IPN

APARTADO POSTAL 14-740 AV. IPN 2508, MÉXICO, D.F., C.P. 07000

DEPARTAMENTO DE INGENIERÍA ELÉCTRICA
SECCIÓN COMPUTACIÓN

"Síntesis de Arquitecturas mediante Cartas de Máquinas de Estado Algorítmica"

Tesis que presenta el Ing. Efrén López Martínez para obtener el grado de Maestro en Ciencias en la especialidad de Ingeniería Eléctrica con opción en Computación

TRABAJO DIRIGIDO POR:
Dr. Sergio Victor Chapa Vergara
Investigador titular y jefe de la Sección de Computación del CINVESTAV

México, D.F. 1998

El árbol que llena los brazos de un hombre, crece de un pequeño tallo.

La torre de nueve pisos se hizo uniendo piedra tras piedra.

El viaje de mil leguas comienza por un solo paso.

Lao Tse

Agradecimientos

A MIS ESCUELAS:

- Primaria "Adrián Zamora López", mis primeros conocimientos.
- "Escuela Secundaria Técnica No. 8", todo un reto haber estado ahí.
- "Escuela Secundaria y de Bachilleres Oficial Minatitlán", el pilar para mis posteriores estudios.
- "Universidad Nacional Autónoma de México", mi Alma Mater.
- "Centro de Investigaciones y de Estudios Avanzados del IPN", mi consagración.

A MIS PROFESORES, por los conocimientos brindados, con dedicatoria especial a:

- Profesora Reyna Cárdenas Torres, por su cariño y los recuerdos.
- Ing. Odilón Huerta Rico por la amistad y las bases sólidas de matemáticas, "Con la comprensión del problema, lo demás es lo de menos".
- M. en C. David Jaimes González Maxinez, modelo a seguir.

A MIS AMIGOS Y COMPAÑEROS, por su amistad desinteresada y porque siempre estuvieron conmigo en los buenos y malos momentos.

A todo el personal de la Sección de Computación, por todo el apoyo brindado durante mi estancia en el CINVESTAV.

Al CONACYT por los recursos y ayuda otorgados durante el posgrado

A MIS HERMANOS: Angelina, Martín, Carmina, Graciela, Efraín, Reyna y Araceli, por creer siempre en mi.

A MIS PADRES: Cesáreo López Martínez y Raymunda Martínez de López, por su amor, por su gran fe en mi y porque han sido, son y serán por siempre, la fuente de energía que me impulsa a continuar superándome hoy y siempre, ¡¡¡los amo!!!.

A mi ESPOSA: María Concepción Estrada Vazquez, por aceptarme como su pareja de por vida, ¡¡¡te amo!!!.

A DIOS, por la vida, la salud, por haber puesto en mi camino a todas esas maravillosas personas sin yo merecerlo y por los grandes momentos de felicidad que he recibido.

A todos mil gracias, espero no haber defraudado a nadie, con cariño:

Efrén López Martínez.

ÍNDICE

"SÍNTESIS DE ARQUITECTURAS MEDIANTE CARTAS DE MÁQUINAS DE ESTADO ALGORÍTMICA"

OBJETIVO PRINCIPAL:

- ⇨ Crear una herramienta de software para el auxilio del diseño de controladores digitales.
  - ☛ Subobjetivo 1: Crear los fundamentos para la síntesis computarizada de controladores digitales con distintas arquitecturas, a partir de Cartas ASM.
  - ☛ Subobjetivo 2: Complementar el desarrollado del editor gráfico para satisfacer el objetivo principal.
  - ☛ Subobjetivo 3: Aprovechar los resultados del editor gráfico para llevar a cabo la síntesis de controladores digitales utilizando la arquitectura MICA I.
  - ☛ Subobjetivo 4: Obtención de los resultados de la síntesis en pantalla y/o papel.
  - ☛ Subobjetivo 5: Aplicación de los conceptos de orientación a objetos en el desarrollo de la aplicación.

INTRODUCCIÓN I.i

CAPÍTULO I: CARTAS ASM: (ALGORITHM STATE MACHINE) 1
I.1 Descripción de los elementos de una Cata ASM 5
I.1.1 Caja de estado 5
I.1.2 Diamante de decisión 5
I.1.3 Caja de salidas condicionales 6
Nodo conector 6
I.1.4 Bloque ASM 6
I.2 Reglas de diseño 9
I.3 Ejemplo de aplicación 14

CAPÍTULO II: ARQUITECTURAS DE DISEÑO 21
II.1 Introducción 22
II.2 Opciones de diseño (generalidades) 23
II.2.1 Diseño con registros de corrimiento 23
II.2.2 Diseño con contadores 25
II.2.3 Controladores microprogramados básicos 26
II.2.4 Controladores con número fijo de instrucciones (arquitectura MICA I) 28
II.2.5 Controladores que manejan subrutinas (arquitectura MICA II) 31
II.3 Arquitectura MICA I 36
II.3.1 Descripción de los elementos de la arquitectura 36
II.3.2 Interacción entre elementos 37
II.3.3 Definición de las instrucciones de la arquitectura 38
II.3.4 Alambrado interno de la arquitectura 40
II.3.5 Procedimiento de diseño 41

CAPÍTULO III:ORIENTACIÓN A OBJETOS COMO METODOLOGÍA DE DISEÑO 45
III.1 Contexto histórico y enfoque paradigmático 46
III.2 El paradigma de la orientación a objetos 49
III.3 Historia de los sistemas orientados a objetos 51
III.4 Elementos de los sistemas orientados a objetos 53

III.4.1 Objeto, interface y métodos 54
III.4.2 Encapsulamiento y ocultamiento de información 55
III.4.3 Clasificación, clase y abstracción 56
III.4.4 Instancia, clase base, clase derivada y tipo abstracto de datos 57
III.4.5 Herencia 58
III.4.6 Polimorfismo 59
III.5 Ejemplo de modelado orientado a objetos 60
III.6 Elementos de beneficio en la OO 63
III.7 Análisis orientado al objeto (AOO) 63
III.7.1 Especificación de requerimientos orientado a objetos de Bailin 65
III.7.2 Análisis orientado al objeto de Coad-Yourdon 66
III.7.3 Análisis orientado al objeto de Shaler-Mellor 67
III.8 Diseño orientado al objeto (DOO) 68
III.8.1 Diseño estructurado orientado al objeto (DEOO) de Wasserman 68
III.8.2 Diseño orientado a objetos de Booch 69
III.8.3 Diseño de manejo de responsabilidades (DMR) de Wirfs-Brock 70
III.9 Comentarios a los métodos orientados a objetos 71

CAPÍTULO IV: DESARROLLO DE LA APLICACIÓN 73
IV.1 Definición del problema 75
IV.2 Una estrategia informal 76
IV. 3 Formalización de la estrategia 77
IV.3.1 Identificación de objetos 77
IV.3.2 Operaciones y atributos aplicados a los objetos 79
IV.4 Componentes e interfaces del programa 80
IV.5 Representaciones gráficas para el DOO 82
IV.6 Detalle de implementación 83
IV.7 Desarrollo simplificado del método 83
IV.8 Comentarios al desarrollo del LDP 108

CONCLUSIONES C.i

APÉNDICE A: DIAGRAMA DE ALGUNOS CIRCUITOS TTL A.1

APÉNDICE B: MANUAL DE USUARIO B.1
Bienvenida B.2
Instalación del sistema B.2
Pasos de la instalación: B.2
¿Cómo entrar al sistema? B.3
¿Cómo salir del sistema? B.4
Convenciones: B.4
Descripción de la interface gráfica del sistema B.4
Elementos permanentes de la interface gráfica del sistema B.4
Menú de elementos de Cartas ASM B.5

Descripción de los iconos del área de iconos B.5
Descripción de las Barras de Desplazamiento B.6
Descripción de los botones del menú de elementos B.7
Operación arrastrar elementos B.7
Operación imprimir B.7
Operación edición del nombre y/o código de estado y/o variables de prueba de los elementos de decisión B.8
Operación edición de variables de salidas y ubicación de ellas en los estados correspondientes B.8
Operación cargar carta de disco B.9
Operación borrado de elementos del sistema B.10
Operación unión de dos elementos de la Carta ASM B.10
Ejemplo de edición de una Carta ASM B.11

APÉNDICE C: BIBLIOGRAFÍA C.1

APÉNDICE D: ARCHIVO DE ESTRUCTURA DE LA CARTA Y ARCHIVO DE SALIDAS D.1

APÉNDICE E: DEFINICIÓN DE CLASES E.1

INTRODUCCIÓN

Introducción

Dentro de todos los acontecimientos de que somos testigos, en la actualidad existe uno que es especialmente distintivo de ella, el auge de la tecnología. Las sociedades tecnificadas se han convertido en las potencias industriales a nivel mundial, rivalizando entre ellas dentro de una carrera tecnológica sin par en la historia humana, sobre todo en áreas como el diseño de dispositivos digitales, la fabricación de computadoras, software y la robótica; que a su vez se han convertido en industrias estratégicas, alrededor de las cuales giran multitud de otras industrias no menos importantes.

Es importante notar que el avance técnico ha ido muy de la mano con el desarrollo de la electrónica digital, al punto de que un anuncio acerca de una mejora en los dispositivos digitales trae consigo la promoción de los artículos basados en ellos que estarán por venir. Es en este aspecto que la industria de fabricación de computadoras ha demostrado ser la más pujante de entre todas. Inicialmente usando componentes dedicados al control industrial, las microcomputadoras siguen su evolución a ritmos ya no de años, sino de meses en los que las generaciones del hardware se suceden y quedan obsoletas.

Como parte muy importante dentro de este acelerado ritmo de evolución, tenemos el hardware, que como ya se dijo, desde sus inicios en la aplicación experimental no ha dejado de mejorar, como tampoco lo han hecho los métodos aplicados a su diseño, desde los puramente algebraicos practicados sobre simples representaciones booleanas de un problema, hasta los complejos algoritmos manejados por computadora, que resuelven problemas tanto de lógica combinacional como secuencial. En el caso especial de las computadoras, las tendencias en el software influyen decididamente en el desarrollo de nuevos dispositivos digitales, cosa que beneficia colateralmente al ámbito industrial que ve actualizada su tecnología más velozmente que lo que podría esperar de sólo depender de la demanda requerida por los procesos industriales.

Actualmente, tanto en el diseño de hardware para computadoras, como en el área de aplicaciones industriales, se cuenta con una variedad de metodologías para el diseño de sistemas digitales. Dependiendo de la escala de integración de la electrónica digital, tenemos:

- ✓ El diseño con lógica combinatoria, que se aplica a la solución de sistemas sencillos usando una representación matemática del problema. Sus componentes más usados son: compuertas lógicas, multiplexores, demultiplexores, decodificadores y comparadores.
- ✓ El siguiente método -un poco más avanzado-, es el diseño con lógica secuencial, éste emplea el concepto de retroalimentación y una representación diagramática para su comprensión y síntesis. Algunos de los componentes electrónicos a los que recurre son: biestables, contadores, registros de corrimiento y multiplexores.
- ✓ Continuando con el grado de complejidad, el diseño por medio de la máquina de estados algorítmica (Cartas ASM por sus siglas en ingles: Algorithm State Machine) impone un mayor potencial en el análisis y solución a problemas reales. Esta herramienta emplea fluxogramas para la representación lógica del problema, siendo su flexibilidad tal que puede ser aplicada fácilmente tanto a la lógica combinacional como a la secuencial, sin que por ello se aleje de problemas mayores como lo serían los controladores digitales microprogramados, usando diferentes arquitecturas como lo son: MICA I, MICA II, bit-slide, etc.

Existen, claro está, otros métodos de diseño aplicables a dispositivos PLD´s (Programmable Logic Devices, tales como PAL, EPLD, GAL´s, etc.), o PLC´s; los cuales recurren a otros métodos de representación del

problema; algunos mediante suma de productos derivados de tablas de verdad, diagramas de estados o mapas de Karnaugh, otros mediante diagramas de escalera, etc.

La elección de una de las metodologías de diseño para llevarla a término en el presente trabajo, trató de balancear lo actual del método con la didáctica. Con ese fin el equipo de desarrollo se decidió por la metodología denominada MICA I para incorporarla al proyecto propuesto, por considerarla representativa de la evolución de los controladores digitales. Para tal efecto, se contemplan los siguientes objetivos:

Objetivo principal: Crear una herramienta de software para el auxilio del diseño de controladores digitales.

Subobjetivo 1: Crear los fundamentos para la síntesis computarizada de controladores digitales con distintas arquitecturas, a partir de Cartas ASM.

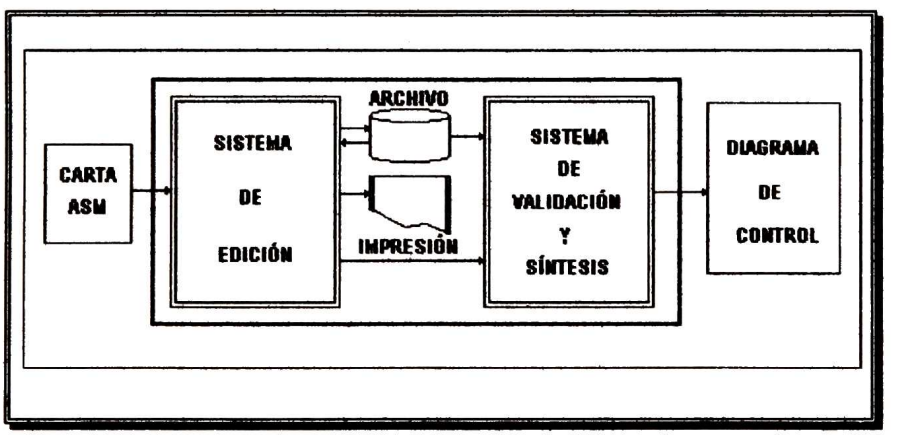
Subobjetivo 2: Complementar el desarrollo del editor gráfico para satisfacer el objetivo principal.

Subobjetivo 3: Aprovechar los resultados del editor gráfico para llevar a cabo la síntesis de controladores digitales utilizando la arquitectura MICA I.

Subobjetivo 4: Obtención de los resultados de la síntesis en pantalla y/o papel.

Subobjetivo 5: Aplicación de los conceptos de orientación a objetos en el desarrollo de la aplicación.

El sistema proyectado para cumplir con los objetivos podría tener una estructura como se muestra a continuación:



La función primordial del sistema a implementar, consiste en realizar el procesamiento de la información derivada de una Carta ASM, a fin de obtener la solución al problema por medio de una técnica aplicable a los controladores digitales.

Si tomamos en cuenta la existencia de un proyecto dedicado a conseguir un ambiente editor de Cartas ASM, podríamos utilizar toda su estructura funcional y acomodar un sistema complementario que determine la resolución de un diseño basado en ellas. De lo contrario tendríamos que construir una interface de usuario que permitiera la captura de la información relacionada con el diseño, que bien podría ser una especie de compilador que reconozca ciertas proposiciones que en su conjunto describan la carta.

Si consideramos también que el editor nos proporcionará la información ya capturada y lista para ser accesada y procesada, deberemos tomar en cuenta el análisis previo del sistema que nos indicará dónde, de que manera y cómo podemos insertar nuestro desarrollo para poder proporcionar los resultados finales del sistema propuesto; que incluyen tablas particulares de la arquitectura, tablas de resultados de la carta a

sintetizar, tablas donde se muestra el microprograma en formato binario y en formato hexadecimal y, por supuesto, el diagrama del circuito controlador.

Atendiendo al subobjetivo número uno, donde se pretende la automatización de la abstracción propia de los métodos de diseño de controladores digitales, se expone el proceso completo de síntesis de Cartas ASM utilizando una de las posibles arquitecturas de diseño (cumpliendo así con el subobjetivo tres), y dejando abierta la posibilidad de realizar posteriores trabajos para la implementación con otras diferentes (como sería el diseño con contadores, con MICA II, etc.) utilizando como base los resultados del proyecto preliminar que generó la interface gráfica y el modelo presentado en este trabajo.

El capitulado propuesto para el desarrollo de esta segunda parte es el siguiente.

Capítulo I.- Cartas ASM (Algorithm State Machine). Donde se proporcionan conceptos que describen las características de las Cartas ASM y se dan las reglas que deben cumplirse para considerarla una carta válida.

En este capítulo se describen las características de los elementos que componen a una carta y la forma en que interactúan entre si.

Capítulo II.- Sintesis de Arquitecturas de diseño. En este capítulo se muestran algunas de las opciones de diseño de controladores digitales, en forma general. Se muestra en forma detallada la arquitectura con la que se realizará la síntesis de la carta (arquitectura MICA I), explicando los distintos módulos que la componen y explicando la interrelación entre ellos.

Capítulo III.- Orientación a objetos como metodolgía de diseño. Donde se proporcionan los conceptos que sirven de base para la programación orientada a objetos.

Capítulo IV.- Desarrollo de la aplicación. Donde se diseña y desarrolla la aplicación bajo un lenguaje de alto nivel con características apropiadas para ello.

Conclusiones. Aquí se hace una evaluación de los resultados y un análisis de las posibles opciones de crecimiento del sistema.

CAPÍTULO I

CARTAS ASM: (ALGORITHM STATE MACHINE)

CAPÍTULO I

CARTAS ASM: (ALGORITHM STATE MACHINE)

En 1973 fue propuesto por Clare[1], un método auxiliar para el diseño de sistemas digitales. Llamó a los sistemas digitales como "Máquina de Estados Algorítmica" (ASM: Algorithm State Machine) y formalizó la relación entre los estados y el funcionamiento del sistema, en diagramas semejantes a los diagramas de flujo. A estos diagramas se les denomina "Cartas ASM".

Las Cartas ASM son entonces, representaciones de algoritmos, y las podemos relacionar con los diagramas de flujo que se elaboran antes de escribir un programa de computación. La Carta ASM representa al "programa" que va a gobernar el funcionamiento del sistema, siendo posible "escribir" este programa en memoria o de forma alambrada por hardware.

Una Carta ASM se define como una representación formal, -mediante un diagrama de flujo- del algoritmo que da solución a un problema de diseño de un sistema digital planteado.

Una descripción fundamental del comportamiento de cualquier sistema de procesamiento de información, ya sea de hardware o software, está provista de un algoritmo. Éste, es una secuencia de eventos cuidadosamente diseñada, la cual es necesaria para obtener un conjunto de resultados o acciones a partir de un conjunto de datos (así, un algoritmo está inherentemente ligado a los datos y al control del flujo). Un algoritmo tiene un punto de inicio y termina después de un número finito de pasos, los cuales están definidos con precisión y sin ambigüedades.

En el diseño de circuitos secuenciales, es imposible desarrollar un solo procedimiento formal que exprese correctamente cualquier tabla de transición; de cualquier forma, a partir de una descripción verbal del sistema a diseñar, se puede utilizar el Algoritmo de la Máquina de Estados para implementarlo, y esto es posible de lograr, gracias a la semejanza de una Carta ASM con un diagrama de flujo común y a la conversión directa de un diagrama de flujo a una Carta ASM.

La definición básica de un sistema que realiza un algoritmo está provista por la estructura de la MÁQUINA DE ESTADOS GENERAL. Esta estructura es mostrada en la Fig. I.1 y comprende 3 submódulos:

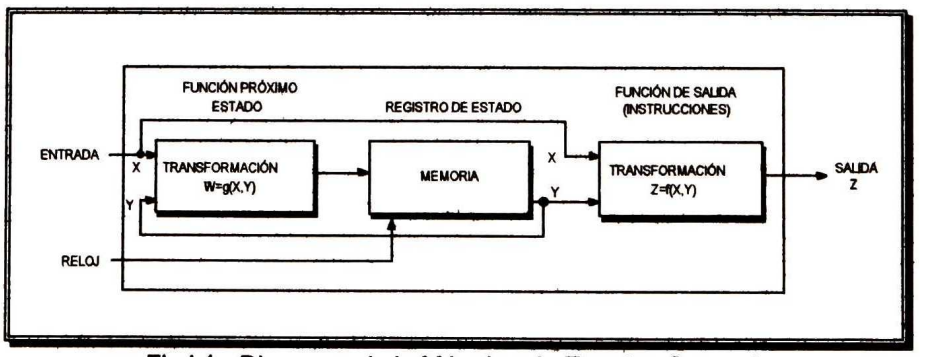


Fig.I.1.- Diagrama de la Máquina de Estados General.

[1] Clare, Designing Logic System Using State Machines.

1. Módulo de Función Próximo Estado. Es un módulo de transformación, implementado con un circuito combinacional, y que se encarga de definir a que estado debe pasar la máquina, después de permanecer en el actual, durante un tiempo determinado por una entrada periódica de reloj; el tiempo durante el cual la máquina permanece en un estado, se conoce como: "tiempo de estado".

2. Módulo Registro de Estado. Es un módulo que contiene elementos de memoria que almacenan el código del estado actual, desde un intervalo de tiempo a otro. Un estado está definido por las variables de estado, que son dígitos binarios, y que definen su código. Al estado definido por el código presente en un instante dado le llamaremos "Estado Activo". Este módulo puede ser implementado de varias formas, desde la utilización de elementos biestables (flip-flops) o registros para almacenar palabras (agrupación de bits) o bloques de palabras, hasta la utilización de circuitos integrados de memorias donde podemos depositar registros con información más completa.

3. Módulo de Función de Salida. Al igual que el módulo de Función Próximo Estado, es un módulo de transformación implementado con un circuito combinacional, pero éste se encarga de definir las salidas que están activas mientras la máquina permanece en el estado presente.

Estos tres módulos operan sobre un conjunto de entradas X para producir un conjunto de salidas Z. Las entradas, representadas por cualquier dato en la trayectoria de entrada, son usadas para obtener los datos de salida, la trayectoria de salida y las señales de control para el funcionamiento interno de la máquina. La trayectoria de salida, puede tomarse como trayectoria de control o puede ser usada para dar inicio a acciones en otros módulos de la máquina. De esta forma, pueden ser referenciadas como "instrucciones".

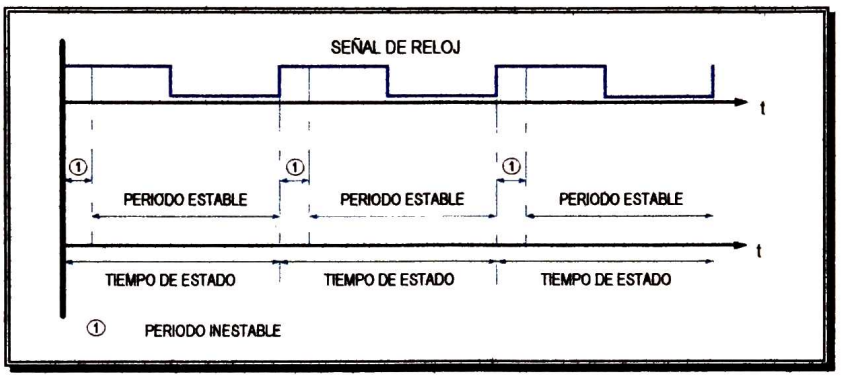
Esta estructura general es suficientemente flexible para cubrir todas la clases de circuitos combinacionales y secuenciales, así como algunos aspectos de unidades de procesamiento de información más complejos. En cada caso los submódulos son interpretados de la manera más apropiada. En el contexto general podemos referirnos a un sistema de este tipo, como una "MÁQUINA DE ESTADOS ALGORÍTMICA" o "ASM".

Dentro del modelo general ASM, los módulos de memoria almacenan información -durante un intervalo de tiempo-, y pueden ser implementados de varias formas, desde simples almacenamientos de bits en elementos biestables, hasta registros de bytes o palabras, o en bloques de varias palabras. Los módulos de memoria son obviamente dependientes del tiempo en cuanto a que los valores de salida de éstos, dependen del estado en que se encuentre la máquina en un instante dado.

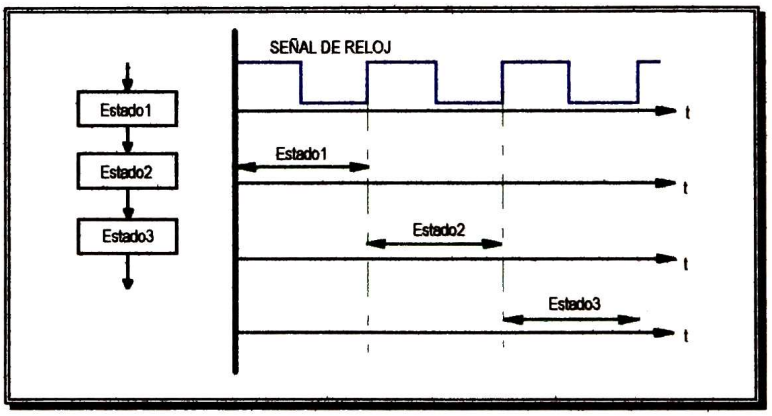
Los módulos de transformación producen un conjunto de salidas por cada conjunto de entradas de acuerdo a relaciones lógicas bien definidas. En términos de hardware, esto puede ser representado por un circuito combinacional o algunos otros dispositivos para cambio de información de una forma a otra; en este contexto, esto es mejor conocido como INTERFACES. El módulo de transformación no es dependiente del tiempo, excepto por el retardo natural de la operación, presente en cualquier realización física. El módulo de transformación puede ser también un componente de software.

Cada estado de la máquina tiene un estado siguiente, determinado por la función próximo-estado; la máquina permanece en el estado actual durante un "tiempo de estado", que es determinado por una entrada periódica o "reloj" al registro de estados. Estando en el nuevo estado, la "Función Próximo Estado" calcula el siguiente nuevo estado, y la "Función de Salidas" calcula el nuevo conjunto de valores de salida, usando los valores de entrada actuales en el nuevo estado, este proceso tiene un retardo antes de ser completado, así, la primera parte del "tiempo de estado" es ocupado por este retardo, dando un período de inestabilidad, el resto

del "tiempo de estado", es un período estable donde las salidas están disponibles y el valor del siguiente estado está listo para la siguiente transición.



Aunque el diseño con Cartas ASM tiene mucho parecido con la técnica de Diagramas de Flujo usada en el diseño de programas de computadora, existe una diferencia fundamental entre ambos, y es con respecto a la relación que existe entre los estados (o procesos) y el tiempo. Los Diagramas de Flujo, usualmente representan un flujo continuo en el tiempo desde el inicio hasta el final, reflejado en la secuencia de la realización de las operaciones del programa; éstas se ejecutan una inmediatamente después de terminada la anterior; las Cartas ASM son conceptualmente diferentes, ya que un estado dura al menos el tiempo determinado por la duración de un "ciclo de reloj", el próximo estado no puede iniciar si no hasta el siguiente "ciclo de reloj", por lo menos. En el siguiente esquema se muestra lo anterior.



En el esquema anterior se observa como el Estado1 permanece activo durante un tiempo mínimo (a ésta duración se le llama Tiempo de Estado) determinado por un ciclo de la señal de reloj; el Estado2 no se activará sino hasta el inicio del siguiente ciclo.

Otra diferencia, entre los diagramas de flujo y las Cartas ASM, se encuentra en la forma de desarrollar las "decisiones"; en los diagramas de flujo, éstas son instrucciones condicionales, mientras que en las Cartas ASM son funciones alambradas físicamente, en las que se evalúan las condiciones de una señal para determinar el flujo a seguir.

En el diseño tradicional de sistemas digitales, se divide al sistema en una parte de datos y otra de control de flujo. Afortunadamente el método ASM puede ser empleado para describir ambas partes, porque en él se pueden especificar los datos y el control del flujo, simultáneamente. Es posible configurar la estructura de las Cartas ASM para reflejar esta dicotomía. Esencialmente, esto requiere la partición de los datos del sistema en: "entradas de control" (variables de prueba), "salidas de estado" y "salidas de control" (instrucciones). Cada una de las partes del sistema (datos y control de flujo) o su totalidad pueden ser descritas por un conjunto de funciones booleanas (llamadas así en honor a George Boole, 1815-1864) derivadas de la estructura ASM.

I.1 Descripción de los elementos de una Carta ASM

Como las Cartas ASM proveen una representación diagramática del comportamiento de cualquier sistema de procesamiento y forman parte de la documentación del diseño, se utilizan en su construcción tres símbolos básicos; éstos son: la Caja de Estado, el Diamante de Decisión y la Caja de Salidas Condicionales.

I.1.1 Caja de estado

La "Caja de Estado" representa un estado o conjunto de condiciones de la Carta ASM; al tiempo que la máquina permanece con este estado como activo se le llama "tiempo de estado". El símbolo de la "Caja de Estado", se muestra en la Fig. I.2, que indica que cada estado tiene un "Nombre" (usualmente un mnemónico o un número) y un "Código de Estado"; este último, está representado por una combinación única de las variables de estado (que son las variables mediante las cuales se definen los estados), y que por lo regular son definidas durante el proceso de "asignación de estados". Así, cuando se dibuja por primera vez la carta, el código de estado es desconocido. Las salidas generadas durante el estado, igualmente representadas en forma de mnemónicos, están listadas dentro de la caja. Una "salida de estado" está activa, solamente cuando la máquina está en un estado que la incluye dentro de su lista de "salidas de estado". Cada Caja de Estado tiene una "Trayectoria de entrada" y una "Trayectoria de salida". La trayectoria de salida puede llevar a otra caja de estado, en este caso, se tiene una transición incondicional o directa; o puede llevar a un "Diamante de decisión", en cuyo caso se tendrá una transición condicional.

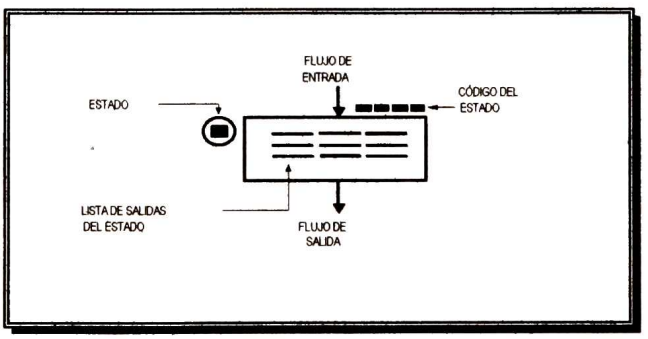


Fig.I.2.- Caja de Estado.

I.1.2 Diamante de decisión

El diamante de decisión se muestra en la Fig. I.3, e involucra las entradas al sistema. Representa un punto en el que el estado siguiente depende del valor que adquiera la variable de entrada o variable de prueba. La variable de prueba se anota dentro del diamante, y puede ser una expresión booleana. Las "Trayectorias de salida" del

diamante son siempre dos y se señalan como 0 y 1 por fines de seguridad, evitando así la incertidumbre generada si se marcan como "verdadero" y "falso" cuando la presencia de una variable se detecta en valor negado. El flujo de la trayectoria es desviado de acuerdo al valor de la variable de prueba. Así, el diamante tendrá una sola entrada y como se dijo anteriormente, sólo dos salidas.

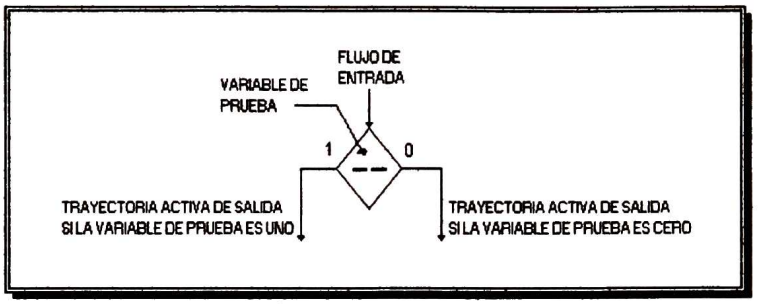


Fig.I.3.- Diamante de Decisión.

I.1.3 Caja de salidas condicionales

La Caja de salidas condicionales está **siempre** asociada a un diamante de decisión y a una caja de estado (formando un bloque ASM, que describiremos más adelante); describe aquellas salidas que están activas, sólo si ciertas condiciones -definidas en términos de las entradas al sistema- son verdaderas y, siempre y cuando la máquina se encuentre en el estado al que está asociada. Así, la caja de salidas condicionales, contiene una lista de salidas condicionadas y como se dijo antes, siempre está asociada a un diamante de decisión. La trayectoria de entrada siempre proviene de un diamante de decisión, pero la trayectoria de salida puede dirigirse a otro diamante de decisión o a una caja de estado. Su símbolo se muestra en la Fig. I.4.

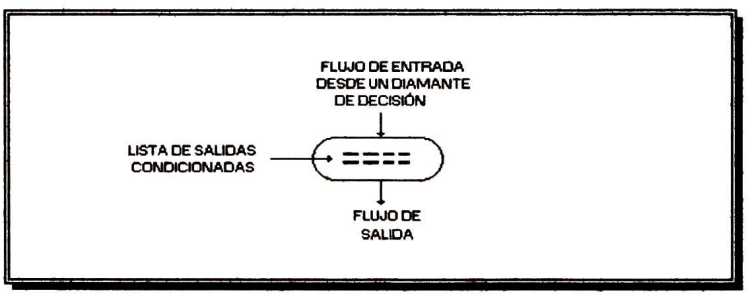


Fig.I.4.- Caja de Salida Condicional.

Nodo conector

Aunque no forma parte de los elementos básicos de una Carta ASM, se puede llegar a utilizar para hacer más claras las representaciones de entradas múltiples hacia un símbolo ASM. Como una convención particular podemos fijar en un nodo hasta 3 entradas múltiples pero sólo una salida.

I.1.4 Bloque ASM

Los elementos básicos (antes descritos), pueden ser combinados para formar un bloque ASM. Ésta estructura consiste de una caja de estado y la red de diamantes de decisión y cajas de salidas condicionales que puedan

existir hasta antes del o de los estados siguientes. Según lo anterior, un bloque ASM puede estar formado con las siguientes combinaciones de elementos: un solo estado, un estado y una red de diamantes de decisión, o un estado y una red de diamantes de decisión combinada con salidas condicionales. Lo anterior es mostrado en la Fig. I.5.

Un bloque ASM tiene sólo una trayectoria de entrada y cualquier número de trayectorias de salida. Cada trayectoria de salida, debe dirigirse a otro estado, y así se convierte en la trayectoria de entrada de otro bloque o de sí mismo. Cada posible trayectoria de un estado a otro, determina una "Trayectoria de Enlace". Cada trayectoria de enlace corresponde a una parte de una expresión booleana, que en su totalidad define las salidas del bloque o la función de siguiente estado.

Dentro de un bloque ASM, la caja de estado es el único elemento que afecta al factor tiempo, todos los demás elementos son asumidos como actividades concurrentes. Así, todos los diamantes de decisión dentro del bloque, son evaluados simultáneamente sin importar su posición dentro de éste; las salidas listadas en las cajas de salidas condicionales que pertenecen al bloque, se activan simultáneamente, aunque no estén todas en la misma caja. Desde este punto de vista, las Cartas ASM difieren de un programa o de un diagrama de flujo. Una Carta ASM consiste de uno o más bloques ASM interconectados de una manera consistente, que describe totalmente el comportamiento de la máquina de estados.

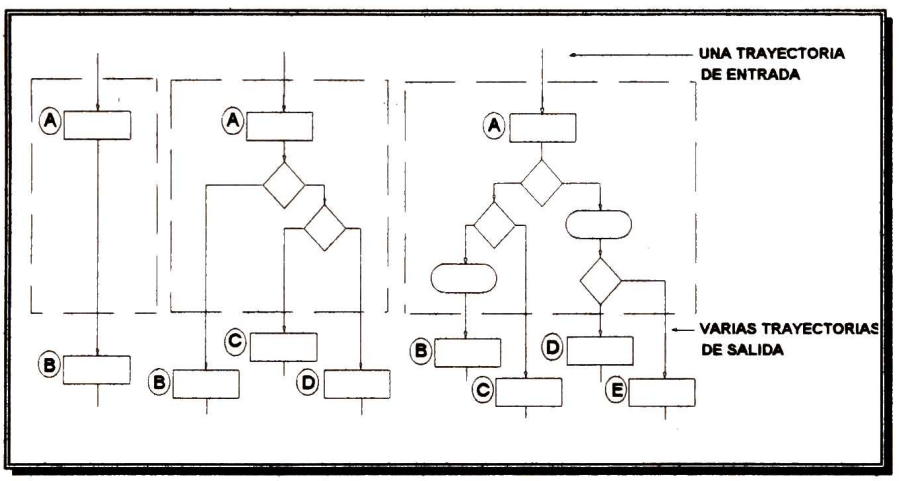


Fig. I.5.- Bloques ASM.

Las Cartas ASM contienen más información que los diagramas de estado de los tradicionales diseños de circuitos secuenciales, pues además de la secuencia del funcionamiento de la máquina de estados que ejecuta el algoritmo, muestra los componentes de las funciones requeridas en la síntesis de la carta. Para comparar lo anterior, se mostrará el diseño de un circuito para la detección de la siguiente secuencia -00 00 11 10- en dos variables de entrada. Se supondrá que, de alguna forma, se controla el número de datos introducidos, y que éstos serán sólo cuatro pares de valores (puesto que el diseño del circuito que controla el número de datos introducidos no es nuestro objetivo, éste no se mostrará). Cuando se detecte el final de esta secuencia, se tendrá un valor de "1" en la salida ISEQ del sistema, indicando la terminación de la secuencia válida.

La Fig. I.6 muestra los diagramas de estado de manera tradicional en el diseño de circuitos secuenciales.

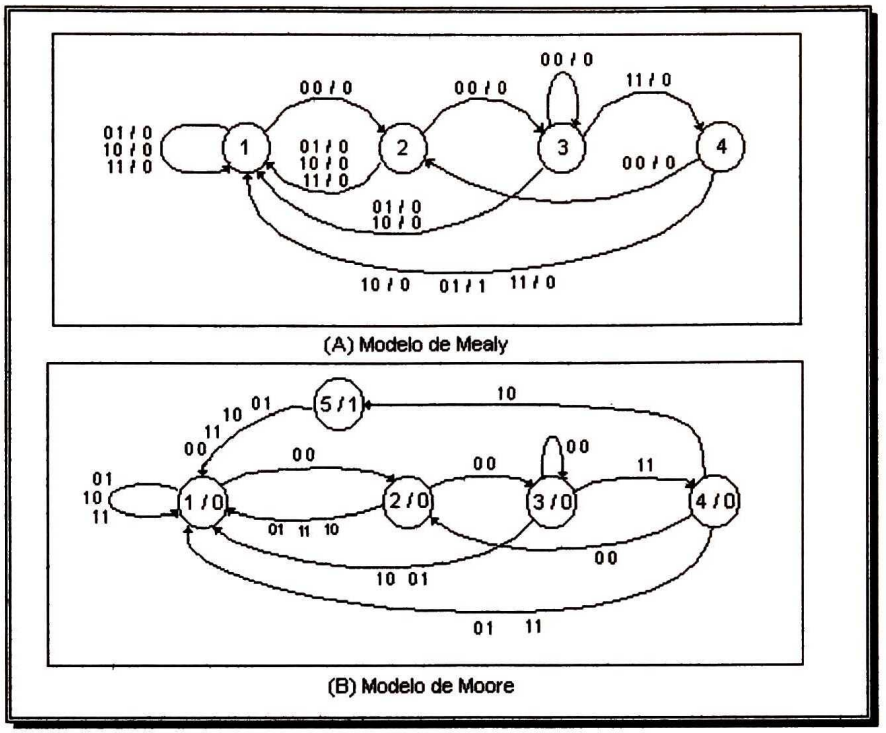


Fig. I.6. Diagramas de estado para un detector de secuencia.

La Fig. I.7, muestra el diseño con Cartas ASM para la detección de la misma secuencia, en ésta, se muestran los componentes X_1 y X_2 con los que se forman las funciones necesarias para la implementación. En la Fig. I.7a, las decisiones son hechas usando los elementos X_1 y X_2 como "entradas de control" del sistema. En la Fig. I.7b, se muestra una forma más compacta con menos diamantes de decisión pero con funciones de decisión más complejas. En cualquiera de los dos casos, se tiene una salida ISEQ (que indica que la secuencia correcta fue introducida) activa sólo en una de las trayectorias de enlace del estado D.

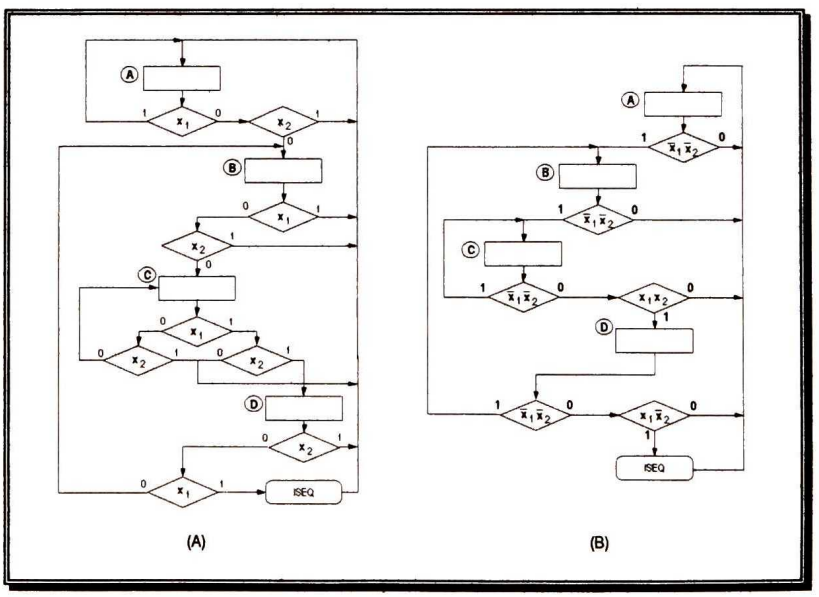


Fig. I.7. Cartas ASM para un detector de secuencia.

Como se ve en la Fig. I.6, existen dos modelos aceptados -con diferente estructura- en los diagramas de estado.

En la parte (A) de la figura, la representación del modelo de Mealy, las salidas son mostradas como funciones de las entradas y de las variables internas del estado. En la representación del modelo de Mealy, las trayectorias de transición están etiquetadas con la combinación de los valores de entrada que causan la transición, y la combinación de los valores de salida resultantes en la transición.

En la parte (B) de la figura, la representación del modelo de Moore, las salidas están asociadas con los estados particulares y así aparecen como etiquetas junto con el nombre del estado o número de nodo. La trayectoria de transición está etiquetada sólo con la combinación de entradas que da lugar a ésta. En este caso, se tiene un estado extra, porque se hace necesario pasar a él para activar la salida cuando la secuencia completa se ha detectado.

Como la descripción de estas representaciones nos desviaría del tema desarrollado, no se detallarán más.

I.2 Reglas de diseño

En el diseño de Cartas ASM se tiene una gran flexibilidad, que es lo que da la gran ventaja al método, pero para asegurar que el diseño sea correcto y por lo tanto se tenga un buen funcionamiento del sistema, debemos cuidar que el diseño de los bloques y de la carta en general, sea significativo y físicamente realizable; para esto debemos de cuidar algunos aspectos que podemos tomar como reglas para el diseño de las cartas y que son los siguientes:

Debemos asegurarnos de que cada estado se dirige a un único próximo-estado, para cada conjunto de condiciones estables en la entrada.

Un error en este punto, comúnmente es debido a un mal arreglo de los diamantes de decisión, donde las condiciones para continuar son imposibles de cumplirse, la trayectoria de enlace es ficticia, o la transición de los estados es ambigua. En la Fig. I.8a y I.8b, la aparente trayectoria de enlace entre los estados A y B no es factible, ya que ésta requiere que la variable X tenga simultáneamente un valor de 0 y 1. De la misma forma, pero menos obvio, tenemos el arreglo de la Fig. I.8c que tiene una trayectoria de enlace desde el estado A al estado B que es imposible de realizarse porque la expresión que define la trayectoria de enlace siempre tendrá un resultado de "cero". Esto es: $X\bar{Y}(\bar{X}+Y) = X\bar{Y}\bar{X} + X\bar{Y}Y = 0$

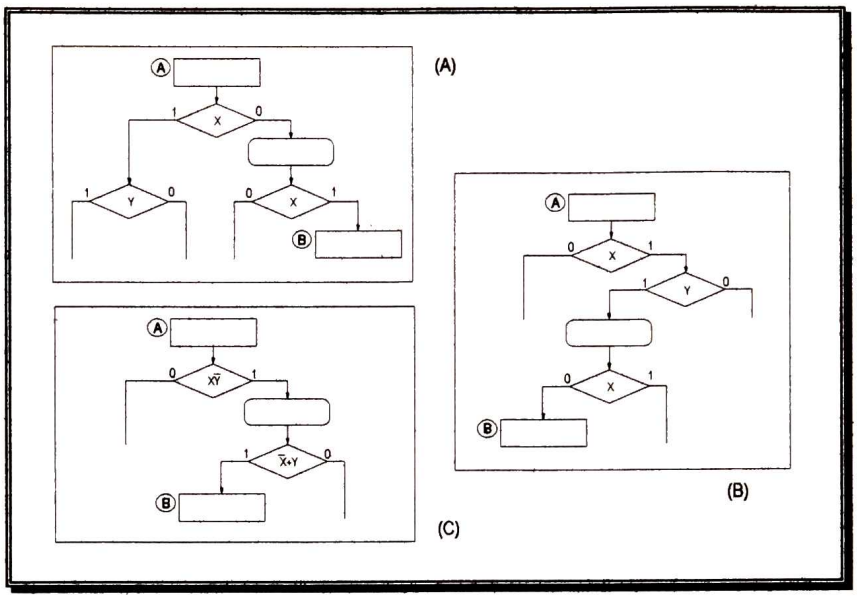


Fig. I.8. Estructuras de Cartas ASM con trayectorias de enlace imposibles.

La Fig. I.9 muestra algunos arreglos inválidos de elementos; la parte (A) de la figura, es claramente imposible de implementar ya que tiene, indicados con ambigüedad, dos próximos-estados simultáneos. La parte (B) de la figura es menos obvia; en ésta, si X y Y tienen un valor de 1, existe un próximo-estado único, sin embargo, si cualesquiera de X o Y tiene un valor de 0, se tendrán entonces indicados dos próximos-estados distintos, y no se sabrá a cuál de ellos se debe pasar.

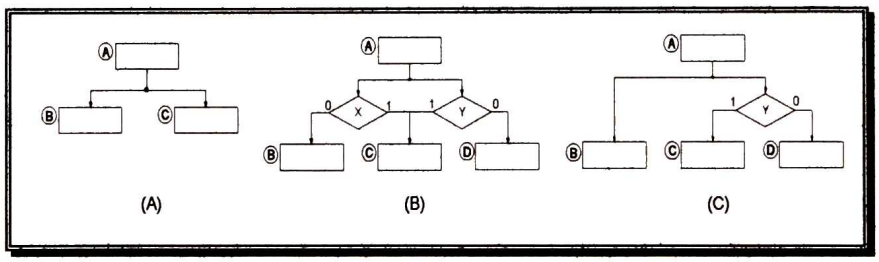


Fig. I.9. Algunas estructuras ASM inválidas.

Lo que probablemente se desea en la estructura (B), es mostrado en la Fig. I.10, donde la ambigüedad desaparece con un simple reacomodo de los diamantes de decisión.

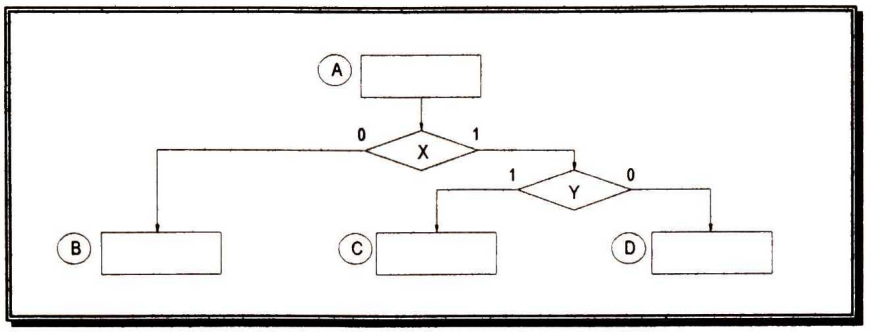


Fig. I.10. Estructura ASM correcta para la fig. I.9b.

Otro punto a cuidar, es que los ciclos que se forman deben tener siempre una última caja de estado, es decir, el bloque que forma el ciclo debe de tener al menos una trayectoria de salida para después reentrar a éste por la trayectoria normal de entrada, o dirigirse a otro bloque. El arreglo de la Fig. I.11a, muestra el caso inválido, y en la Fig. I.11b es redibujado en la forma correcta.

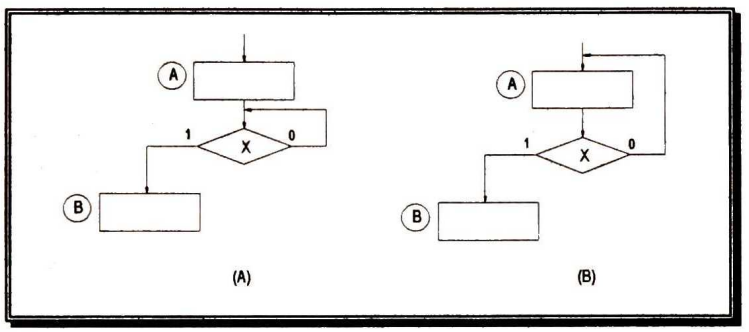


Fig. I.11. Representaciones (A) inválida y (B) válida de un ciclo.

Es posible tener la misma variable de prueba en distintos diamantes de decisión -dentro de un mismo bloque-, y tener una estructura o bloque ASM válido, siempre y cuando las trayectorias de enlace del estado actual estén definidas sin conflictos entre ellas; conflictos como dos posibles estados siguientes para una sola combinación de las variables de entrada, o que no exista un próximo estado definido para al menos una de las trayectorias de salida. En la Fig. I.12 se muestran dos ejemplos de estructuras válidas donde una variable aparece en más de un diamante de decisión.

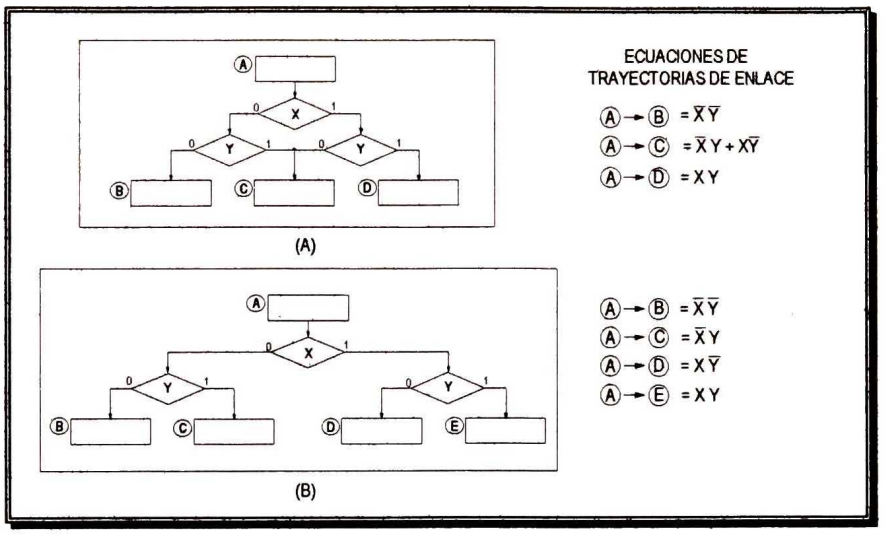


Fig. I.12. Algunas estructuras de Cartas ASM válidas.

Tenemos la libertad de utilizar en forma equivalente, series de diamantes de decisión interconectados en serie o en paralelo dentro de un bloque ASM. Como dentro de un bloque, todos sus elementos son evaluados simultáneamente, las decisiones y las salidas de las cajas de salida condicional, son evaluadas y aparecen -respectivamente- de manera simultánea, esto hace que las dos estructuras mostradas en la Fig. I.13 sean correctas y equivalentes.

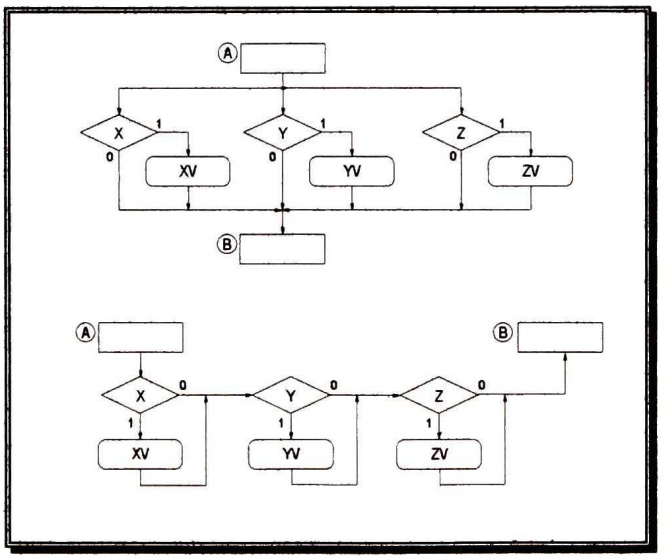


Fig. I.13. Estructuras de Cartas ASM válidas y equivalentes.

Con las estructuras en serie es menos probable que se presenten ambigüedades en las trayectorias de transición, pero con las estructuras en paralelo suelen construirse cartas más compactas.

En ocasiones es posible hacer Cartas ASM más compactas si los bloques comparten diamantes de decisión o cajas de salidas condicionales, evitando así duplicaciones. El hecho de que los bloques ASM puedan traslaparse no es problema, siempre y cuando sean identificables y sus trayectorias de enlace sean bien

definidas. Como un ejemplo de esto, se puede considerar la Carta ASM de la Fig. I.14, en la cual, un diamante de decisión es compartido entre el bloque del estado B y el del estado C. El hecho de que esto pueda ocurrir, es gracias a que las condiciones de transición para cada caso, se presentan en instantes distintos.

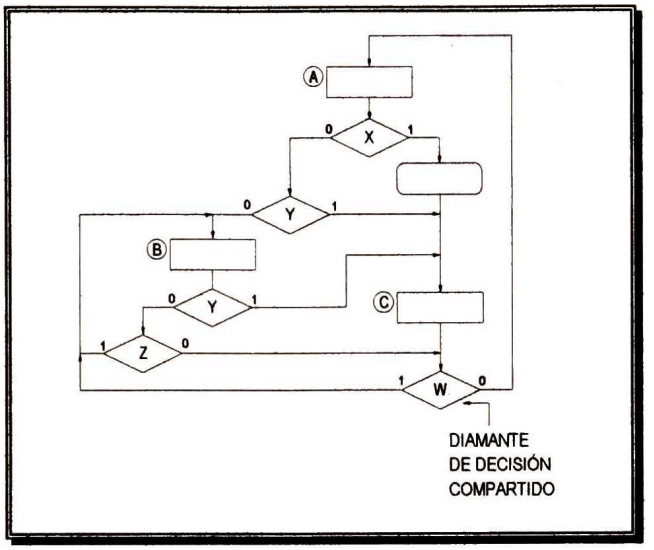


Fig. I.14. Ejemplo de Carta ASM.

En esta figura podemos fácilmente extraer los tres bloques ASM correspondientes a los tres estados y podemos también identificar sus trayectorias de enlace, como se muestra en la Fig. I.15. Con práctica esto puede lograrse sin redibujar los bloques.

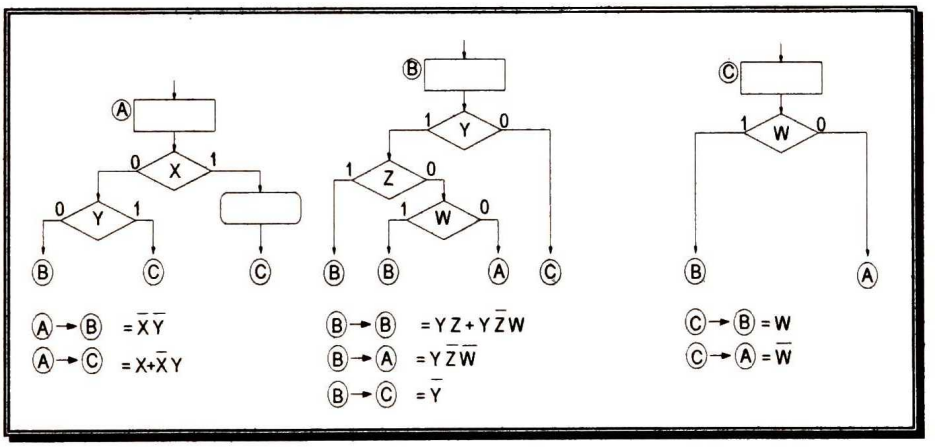


Fig. I.15. Bloques ASM y trayectorias de enlace de la carta de la Fig. I.14.

La conexión directa de un estado con una salida condicional no está permitida, puesto que las salidas condicionales están **siempre** asociadas a una de las trayectorias de salida de un diamante de decisión.

I.3 Ejemplo de aplicación

Para mostrar el método de diseño con Cartas ASM, se desarrollará un ejemplo paso a paso, siguiendo el algoritmo que da solución a un problema o requerimiento dado. En el siguiente procedimiento, se busca mostrar como se puede llevar a cabo la elaboración de una Carta ASM a partir de un problema dado.

Aunque el procedimiento de diseño puede variar dependiendo del diseñador, aquí se presentan los siguientes pasos a realizar para alcanzar nuestro objetivo:

1.- Planteamiento por escrito del problema (especificar requerimientos del problema).

2.- Planteamiento de un diagrama a bloques general del sistema (bloque de entradas, bloque controlador y bloque de salidas).

3.- Detallar en módulos el bloque controlador.

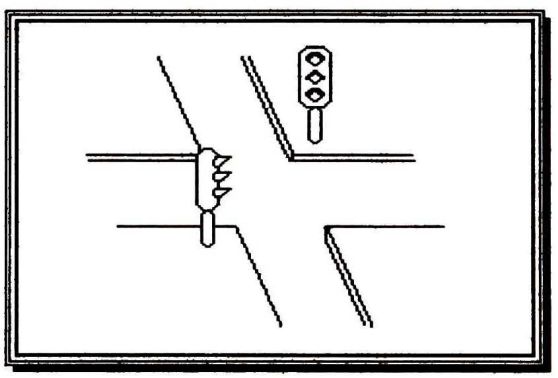
4.- Desarrollar un algoritmo de solución.

5.- Realizar la Carta ASM del controlador.

Solución:

1.- Planteamiento del problema.

El objetivo es crear la Carta ASM que describa el funcionamiento del circuito controlador de un semáforo ubicado en el cruce de dos avenidas, una con circulación de sur a norte (SN) y la otra con circulación de este a oeste (EO). El funcionamiento del semáforo tiene los siguientes requisitos:



En condiciones iniciales, tendremos: En la dirección sur-norte (SN) luz verde encendida, y en la dirección este-oeste (EO) luz roja.

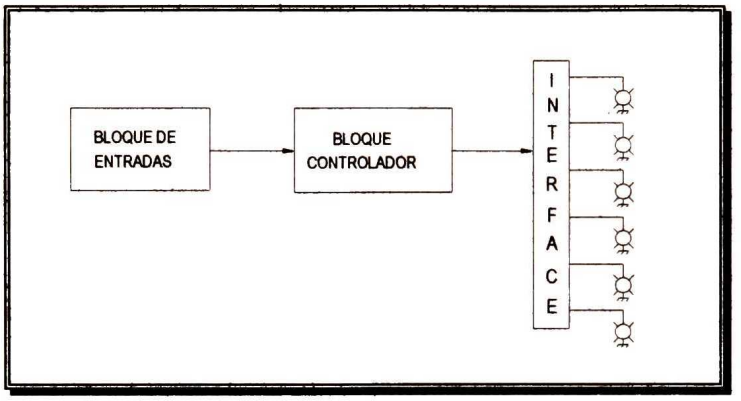
En la dirección SN las luces tendrán los siguientes tiempos de encendido: verde, 30 unidades de tiempo (UT); amarilla, 5 UT; roja, 30 UT.

Los tiempos de encendido para las luces de la dirección EO estarán de acuerdo a los de las luces de la dirección SN, considerando que la luz EO amarilla debe estar encendida por 5 UT mientras la dirección SN mantiene su luz roja.

Una vez cumplidos los tiempos anteriores, se volverá a condiciones iniciales para empezar de nuevo.

Las unidades de tiempo serán proporcionadas por un circuito temporizador.

2.- Diagrama a bloques general.



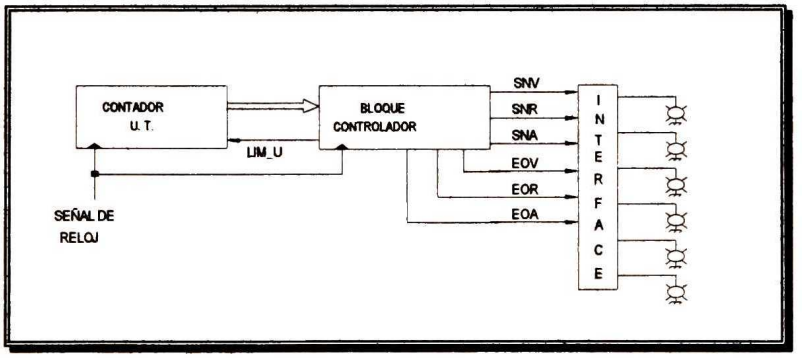
3.- Bloque controlador.

i. Del enunciado del problema, determinamos que el controlador sólo tendrá como entrada la señal proporcionada por el contador de unidades de tiempo, ya que deberá funcionar automáticamente una vez que se habilite, generando internamente las señales necesarias para su funcionamiento y para la inicialización del contador de unidades de tiempo.

ii. Se tendrá un contador de unidades de tiempo para determinar los períodos de tiempo requeridos en cada salida.

iii. Se contará con una interface para traducir las señales del controlador y activar las luces del semáforo.

Del análisis anterior, se obtiene el diagrama a bloques más detallado.



4.- Desarrollo del algoritmo de solución.

En un principio, se proporcionará el algoritmo en su totalidad para mostrar un panorama completo de los requerimientos del sistema a implementar, después se seguirá paso a paso para ir construyendo la Carta ASM y finalmente se mostrará ésta en forma completa.

⇒ El semáforo se encuentra en el cruce de dos avenidas, una con circulación de sur a norte (SN) y la otra con circulación de este a oeste (EO).

⇒ Inicialmente, la dirección SN tendrá luz verde y por lo tanto la dirección EO tendrá luz roja.

⇒ Estas condiciones permanecerán fijas durante 30 unidades de tiempo o ciclos de reloj.

⇒ Después de esto, la dirección SN pasará a luz amarilla durante 5 unidades de tiempo, y la dirección EO mantendrá su luz roja.

⇒ A continuación, la dirección SN mostrará luz roja y la dirección EO luz verde, esto durante 25 unidades de tiempo.

⇒ Una vez cumplido lo anterior, la dirección EO tendrá luz amarilla durante 5 unidades de tiempo, mientras que la dirección SN mantiene su luz roja.

⇒ Cuando los pasos anteriores se cumplan, el semáforo pasará a las condiciones iniciales, es decir, dirección SN en verde y dirección EO en rojo, para continuar con el ciclo descrito.

5.- Realización de la carta ASM.

Lo anterior describe el funcionamiento completo requerido para el semáforo, a continuación seguiremos el algoritmo paso a paso e iremos construyendo la Carta ASM que describe el circuito controlador.

⇒El semáforo se encuentra en el cruce de dos avenidas, una con circulación de sur a norte (SN) y la otra con circulación de este a oeste (EO).

En base a este párrafo, se determina que deben existir al menos las siguientes salidas:

SNV.- Luz verde para la dirección SN.

SNA.- Luz amarilla para la dirección SN.

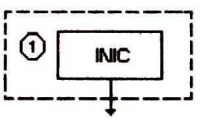
SNR.- Luz roja para la dirección SN.

EOV.- Luz verde para la dirección EO.

EOA.- Luz amarilla para la dirección EO.

EOR.- Luz roja para la dirección EO.

Una medida sana para asegurar que inicialmente todos los dispositivos (contadores, registros, elementos biestables, etc.) estén limpios de información que pudiera alterar el funcionamiento del circuito, es que la Carta ASM contenga como primer estado, uno donde exista una señal de salida (INIC) para inicializar o mandar a cero todo. Esto nos lleva a que generalmente, las Cartas ASM tengan como primer bloque el siguiente:



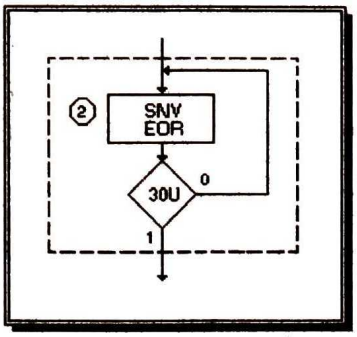
Ahora analicemos los dos siguientes pasos del algoritmo dado.

⇒ Inicialmente, la dirección SN tendrá luz verde y por lo tanto la dirección EO tendrá luz roja.

⇒ Estas condiciones permanecerán fijas durante 30 unidades de tiempo o ciclos de reloj.

Esto nos lleva a tener un estado donde existan las salidas SNV y EOR; como esto debe permanecer durante 30 unidades de tiempo, tenemos que verificar la entrada 30U que aparece cuando un contador de unidades de tiempo llegue a 30 (este contador se inicializa en cero con la señal de salida del primer estado), cuando esto ocurra seguiremos con el próximo paso del algoritmo.

Lo expresado en el párrafo anterior nos lleva a un bloque ASM como el siguiente:

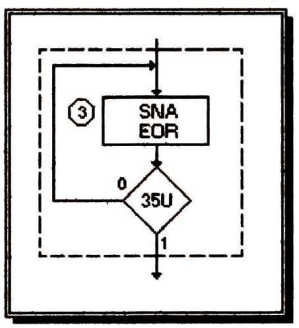


En este bloque, la trayectoria de entrada viene de la trayectoria de salida del estado anterior.

Continuando con el análisis del algoritmo tenemos:

⇒ Después de esto, la dirección SN pasará a luz amarilla durante 5 unidades de tiempo, y la dirección EO mantendrá su luz roja.

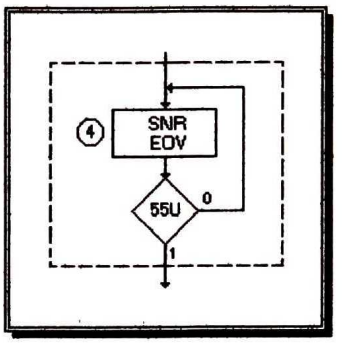
Esto lo realizaremos con un estado que contenga las salidas SNA y EOR; como se especifica que esto debe permanecer por 5 unidades de tiempo, la siguiente entrada a verificar debe activarse o presentarse cuando el contador de unidades de tiempo llegue a 35, esto nos dará las 5 unidades de tiempo requeridas, sin tener que inicializar de nueva cuenta el contador. El siguiente bloque ASM realiza lo anterior.



Continuando con el siguiente paso del algoritmo, tenemos:

⇒ A continuación, la dirección SN mostrará luz roja y la dirección EO luz verde, esto durante 25 unidades de tiempo.

Como la dirección SN debe mostrar luz roja y la dirección EO luz verde, entonces, en el siguiente estado deben existir las salidas SNR y EOV y estar activas durante 25 unidades de tiempo; para conseguir esto, dejaremos que el contador de unidades de tiempo llegue a 55 unidades y no a los 60 que podría esperarse, esto previendo que la luz roja de la dirección SN debe permanecer encendida aún cuando cambie la luz -de verde a amarillo- en la dirección EO; las 5 unidades de tiempo faltantes para completar el tiempo de la salida SNR, se presentarán en el siguiente bloque. El bloque que se muestra a continuación representa las anteriores condiciones:

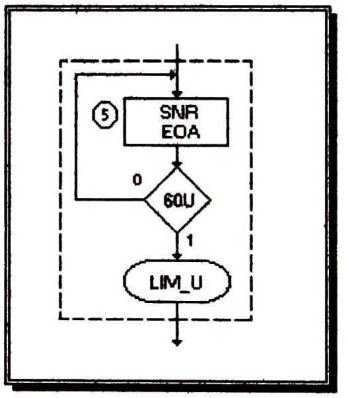


Por último analizaremos los dos últimos pasos del algoritmo:

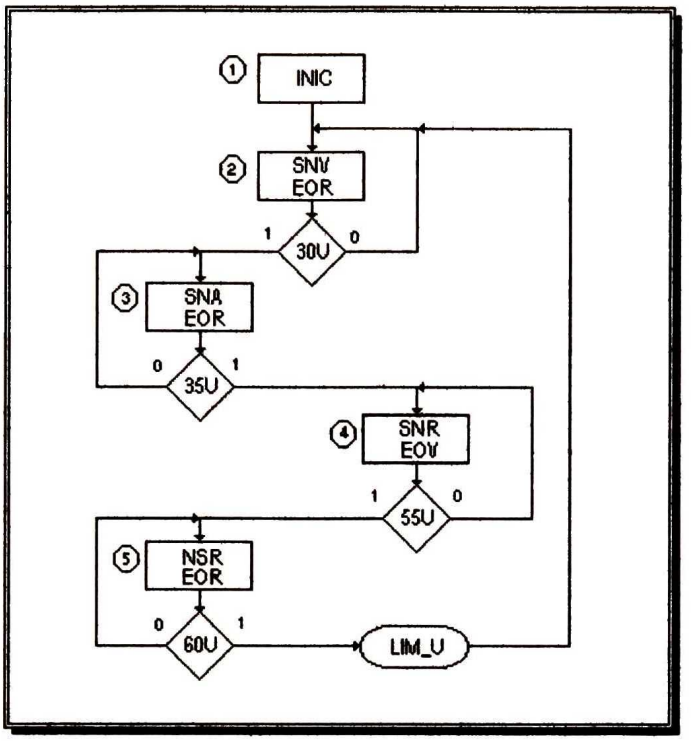
⇒ Una vez cumplido lo anterior, la dirección EO tendrá luz amarilla durante 5 unidades de tiempo, mientras que la dirección SN mantiene su luz roja.

⇒ Cuando los pasos anteriores se cumplan, las condiciones del semáforo pasarán a las condiciones iniciales, es decir, dirección SN en verde y dirección EO en rojo, para volver a iniciar el ciclo descrito.

El bloque ASM que sirve para describir lo anterior tendrá un estado donde se activarán las salidas SNR y EOA durante 5 unidades de tiempo, con esto se cumplirá el requerimiento para la luz amarilla en la dirección EO y se completará la duración de la luz roja de la dirección SN. Dentro de este bloque incluiremos también una caja de salidas condicionales donde se activará la salida LIM_U -sólo cuando la cuenta llegue a 60- que servirá para volver el contador de unidades de tiempo a cero ya que con esto podemos reiniciar el algoritmo bajo las condiciones iniciales, por lo tanto, en este instante, la trayectoria de salida de este bloque será la trayectoria de entrada al bloque del estado 2. Hacemos que la trayectoria de salida corresponda a la trayectoria de entrada del estado 2 y no a la del estado 1, porque en éste se tiene una señal de salida para limpiar o inicializar todos los elementos del sistema y para lo que buscamos sólo es necesario inicializar el contador de unidades de tiempo.



Al unir los bloques anteriores, obtenemos la Carta ASM completa.



Lista de variables:

INIC.- Salida para inicializar o mandar a cero todos los dispositivos del sistema.

SNV.- Salida para habilitar la luz verde de la dirección sur-norte.

SNR.- Salida para habilitar la luz roja de la dirección sur-norte.

SNA.- Salida para habilitar la luz amarilla de la dirección sur-norte.

EOV.- Salida para habilitar la luz verde de la dirección este-oeste.

EOR.- Salida para habilitar la luz roja de la dirección este-oeste.

EOA.- Salida para habilitar la luz amarilla de la dirección este-oeste.

30U.- Señal interna que indica que el contador de unidades de tiempo está en 30.

35U.- Señal interna que indica que el contador de unidades de tiempo está en 35.

55U.- Señal interna que indica que el contador de unidades de tiempo está en 55.

60U.- Señal interna que indica que el contador de unidades de tiempo está en 60.

LIM_U.- Salida activa cuando el contador de unidades de tiempo llegue a 60, utilizada para regresar el contador a cero y volver a condiciones iniciales.

CAPÍTULO II

SÍNTESIS DE ARQUITECTURAS DE DISEÑO

CAPÍTULO II

SÍNTESIS DE ARQUITECTURAS DE DISEÑO

II.1 Introducción

Un controlador, es un sistema que presentará un comportamiento determinado por su diseño, siempre y cuando reciba las señales de entrada previstas; es decir, es un sistema cuyas salidas responden a las entradas de acuerdo con su diseño.

Los controladores digitales cumplen con esta regla, ya que reciben señales digitales como entradas y responden cambiando de estado y/o presentando ciertas salidas.

Un sistema digital es un dispositivo físico, normalmente un circuito electrónico caracterizado por el hecho de tener un número finito de estados posibles.

El diseño de sistema digitales es un proceso complejo que en su forma más pragmática exige conocimiento técnico especializado y un fundamento matemático que permiten explotar las posibilidades de los sistemas digitales. Existen varias opciones para el diseñador, desde las más artesanales y empíricas, hasta las que utilizan herramientas matemático-metodológicas que ofrece la tecnología, sin embargo, en todas ellas la intuición y la experiencia juegan un papel más importante en el proceso del que generalmente se les reconoce.

Conforme los circuitos integrados se vuelven menos caros y más complejos, más y más funciones se agregan a sistemas anteriormente sencillos, posibilitando la construcción de sistemas digitales proporcionalmente complejos y económicos que se incorporan ahora y cada vez en mayor medida, a la industria y a la vida diaria.

En el capítulo anterior, se mostró la teoría de Cartas ASM y se desarrolló un ejemplo; hasta aquí tenemos la mitad del camino recorrido para construir controladores digitales, para completarlo necesitamos traducir la Carta ASM obtenida, en un circuito físico diseñado en base a ésta. Para ello, como ya se dijo, se tienen distintas opciones, desde el diseño con elementos biestables (flip-flop's), hasta el empleo de arquitecturas microprogramadas (de las que más adelante se hablará).

Para exponer las opciones de diseño presentadas en este trabajo, clasificaremos a los controladores en dos grupos: **cableados** y **microprogramados**.

Consideraremos como **controladores cableados** a aquellos que no pueden modificar su funcionamiento sin alterar su estructura física, es decir, son controladores dedicados, que realizan una determinada función y nada más, si se desea modificar, se tendrán que cambiar algunos de sus componentes o cambiar la forma de interactuar entre ellos; en otras palabras, el concepto de controlador cableado define a aquellos controladores digitales cuyo funcionamiento (entradas-salidas) depende exclusivamente de sus conexiones internas, es decir, de los elementos usados en su construcción y de la relación (cableado) entre ellos.

En los **controladores microprogramados** se puede modificar su funcionamiento sin tener que modificar su estructura física, esto es posible gracias a que cuentan con un elemento de memoria que almacena un programa que gobierna el funcionamiento del controlador, con esto, cuando queremos modificar el funcionamiento del controlador sólo tenemos que modificar el programa almacenado en la memoria sin tener que modificar la estructura física del controlador.

Aunque la diferencia entre estos tipos de controladores es clara, se puede diseñar un controlador requerido mediante cualesquiera de los tipos mencionados. La elección del tipo de controlador a utilizar, dependerá de la flexibilidad necesaria y del mantenimiento.

En este capítulo, dentro de los diseños con arquitectura cableada, contemplaremos el **Diseño con Registros de Corrimiento**, y el **Diseño con Contadores;** dentro de los controladores microprogramados se expondrán el diseño de **Controladores Microprogramados Básicos**, controladores con **Arquitectura MICA I**, y controladores con **Arquitectura MICA II.** Lo anterior se hará sin profundizar en detalles de funcionamiento, buscando dar un panorama general de las opciones de diseño, para al final del capítulo presentar a detalle el funcionamiento de la **Arquitectura MICA I** que será utilizada para el desarrollo final del sistema propuesto.

En este trabajo partiremos de la base que el lector conoce las características a detalle de los circuitos mencionados, por lo que no profundizaremos en sus características físicas ni de funcionamiento (para obtener información acerca de los circuitos empleados, consulte el manual correspondiente).

II.2 Opciones de diseño (generalidades)

Como se dijo antes, se presentará el funcionamiento de las arquitecturas listadas anteriormente sólo de una manera general, esto para mostrar un panorama de las opciones que se tienen al diseñar controladores, sin tener que detenernos en detalles de implementación. Sólo más adelante, dentro de este mismo capítulo, se detallará la arquitectura MICA I puesto que ésta será la desarrollada en el sistema.

II.2.1 Diseño con registros de corrimiento

Esta arquitectura de diseño se basa en el funcionamiento de un registro de corrimiento universal (como el 74194) con las siguientes características: líneas de entradas en serie, bus de entradas en paralelo, bus de salidas, desplazamiento de la información a la derecha, desplazamiento de la información a la izquierda.

En un registro de corrimiento del tipo requerido podemos realizar las siguientes instrucciones:

INSTRUCCIÓN	COMENTARIOS	MNEMÓNICO
Limpiar	Fija a un nivel bajo las terminales del bus de salida.	LMP
Retener	Mantiene el estado actual.	RTN
Desplazamiento a la izquierda	Desplaza la información de las salidas, a la izquierda e introduce el valor presente en la terminal de entrada izquierda en serie.	DI
Desplazamiento a la derecha	Desplaza la información de las salidas, a la derecha e introduce el valor presente en la terminal de entrada derecha en serie.	DD
Cargar	Carga el dato presente en el bus de entradas en paralelo.	CRG

Si la operación se realiza sólo cuando se cumple cierta condición (determinada por el valor lógico de una variable de entrada), tendremos como resultado instrucciones condicionales como las siguientes:

INSTRUCCIÓN	COMENTARIOS	MNEMÓNICO
Desplazamiento condicional a la izquierda	Desplaza condicionalmente la información de las salidas, a la izquierda e introduce el valor presente en la terminal de entrada izquierda en serie.	DCI
Desplazamiento condicional a la derecha	Desplaza condicionalmente la información de las salidas, a la derecha e introduce el valor presente en la terminal de entrada derecha en serie.	DCD

La siguiente figura muestra la estructura de este tipo de controladores:

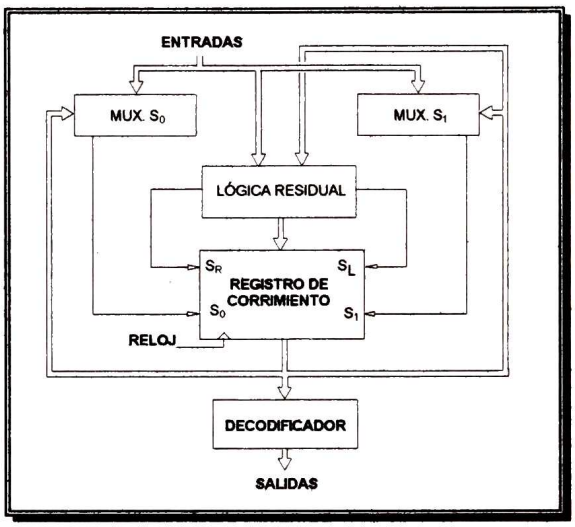


Fig. II.1 Estructura de un controlador digital basado en un registro de corrimiento.

Los pasos a seguir en el diseño con registros de corrimiento y partiendo de una Cartas ASM ya diseñada los podemos desglosar como sigue:

1. Asignar a cada estado un código binario que favorezca las operaciones de corrimiento (p. ej., del estado 0010 pasar al estado 0100, donde se tiene un corrimiento a la izquierda con carga de cero en el "hueco" que se crea), puesto que éstas son las operaciones naturales de estos circuitos, el tener operaciones de brincos o carga de entradas en paralelo (p. ej., del estado 0011 pasar al estado 1100) significa el tener que utilizar compuertas básicas para construir bloques de lógica residual que obtengan los valores de las entradas en paralelo, en base al valor de las variables de estado.
2. Una vez asignados los códigos de estados, elaborar una tabla que mostrará para cada estado el código de éste y la operación que se realiza en él, esta tabla queda de la siguiente manera:

NOMBRE DEL ESTADO	CÓDIGO DE ESTADO	OPERACIÓN

3. De acuerdo a la tabla anterior y conociendo la "tabla de control de modo" (consultar el manual correspondiente) del registro, elaborar "mapas de acción" para cada terminal de control del registro de corrimiento (S_L, S_R, S_0, S_1).
4. Elaborar mapas de carga en paralelo para determinar el dato que se deberá cargar cuando se presente un brinco.
5. Elaborar mapas de carga en serie para determinar el dato que se deberá cargar cuando se presente un corrimiento.
6. Elaborar un mapa de las salidas para relacionarlas con las salidas del decodificador de salida, que estará direccionado por las salidas del registro (que definen el estado actual).

El manejo de las entradas se realiza a través de multiplexores direccionados utilizando directamente las salidas del registro, las salidas de los multiplexores están relacionadas con las entradas de control del registro, evitando así tener que alambrar la función obtenida de los "mapas de acción" con compuertas básicas.

II.2.2 Diseño con contadores

Otra posibilidad de estructurar un controlador, es basándonos en un contador (aquí utilizaremos el 74161) con capacidad de cargar un dato y retenerlo para comenzar, cuando sea necesario, una cuenta a partir de él.

La Fig. II.2 muestra la estructura de un controlador basado en un contador.

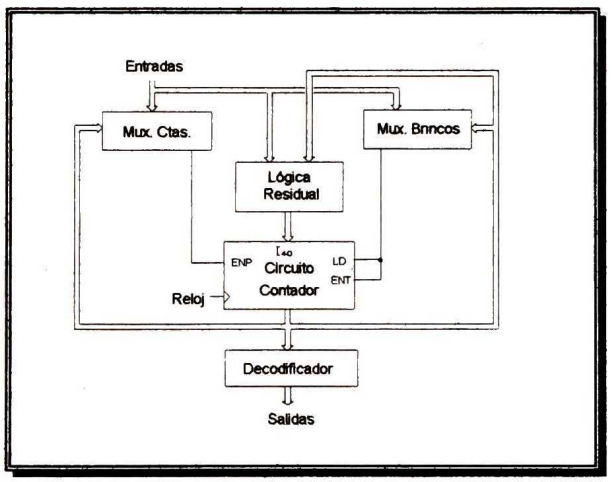


Fig. II.2 Estructura de un controlador digital basado en un contador.

Los estados de nuestro controlador serán generados por el contador, el cual cuenta con tres terminales de control (LD, ENT y ENP) que permiten habilitarlo para cargar un nuevo dato, retenerlo o contar.

Esto permite que estemos en la posibilidad de utilizar las siguientes instrucciones:

CI: Cuenta Incondicional. El estado actual se incrementa en uno.

CC: Cuenta Condicional. El estado actual se incrementa en uno, sólamente si se presenta cierta variable. Si esta variable no se encuentra en estado activo, el contador retendrá el dato presente actualmente en la salida.

BI: Brinco Incondicional. El estado actual cambia por el dato presente en el bus de entradas en paralelo.

BC: Brinco Condicional. El estado actual cambia por el dato presente en el bus de entradas en paralelo, siempre y cuando se presente cierta variable. Si esta variable no se encuentra en estado activo, el contador retendrá el dato actual.

RT: Retención. El estado actual es mantenido.

Estas instrucciones se generan a partir de la "tabla de control de modo" del contador (ver manual correspondiente). Los pasos de diseño del controlador con contadores siguen la misma línea del diseño con registros de corrimiento, a saber:

1.- Asignar códigos a los estados elaborando la "tabla de estados presentes e instrucciones" aplicadas en cada estado.

Las instrucciones se asignarán dando evidente prioridad a las cuentas y tratando de evitar los brincos puesto que éstos implicarán el uso de compuertas básicas para definir los valores correspondientes en las entradas en paralelo del contador. No está de más el comentar que la solución de un controlador varía con la asignación de estados y que un controlador basado en un contador se simplifica cuando la asignación de estados se presta para evitar los brincos, puesto que la operación "natural" de un contador es precisamente contar. La tabla referida tendrá la siguiente estructura:

NOMBRE DEL ESTADO	CÓDIGO DEL ESTADO	OPERACIÓN

2.- De acuerdo a la "tabla de instrucciones" anterior, y conociendo la "tabla de control de modo" del contador, elaborar "mapas de acción" para cada terminal de control del contador.

Del mismo modo que con el registro de corrimiento, en este paso estamos realizando la programación de nuestro controlador.

3.- Elaborar mapas de carga en paralelo para determinar el dato que se deberá cargar cuando se presente un brinco.

4.- Elaborar un mapa de las salidas para relacionarlas con el decodificador de salidas.

5.- Al igual que en el diseño con registros de corrimiento, aquí utilizaremos multiplexores (direccionados con las salidas del contador) para considerar las variables de entrada en el momento en que se presente el estado con el que está relacionada. Las salidas de los multiplexores están conectadas a las terminales de control del contador.

II.2.3 Controladores microprogramados básicos

Hasta este momento, los controladores expuestos basan su funcionamiento en las conexiones (el cableado) entre los circuitos que lo componen; en ocasiones, los controladores así diseñados son difíciles y complejos de construir, y si consideramos que sí queremos modificar el funcionamiento del controlador -por pequeño que sea el cambio- tendremos que volver a diseñar y realambrar por lo menos los bloques de lógica residual, se hace deseable una arquitectura flexible desde este punto de vista.

Se puede agregar flexibilidad al sistema incluyendo una memoria que nos permita almacenar los datos que requerimos para trabajar mediante la estructura mostrada en la Fig. II.3; en esta figura, los bloques de lógica

residual desaparecen gracias a la utilización de la memoria, y con esto ya no es necesario realambrar el controlador si deseamos cambiar su funcionamiento, sólo tendremos que reprogramar la memoria de programa. Cabe señalar que la desaparición de los bloques de lógica residual constituyen la diferencia básica entre los controladores cableados y los microprogramados.

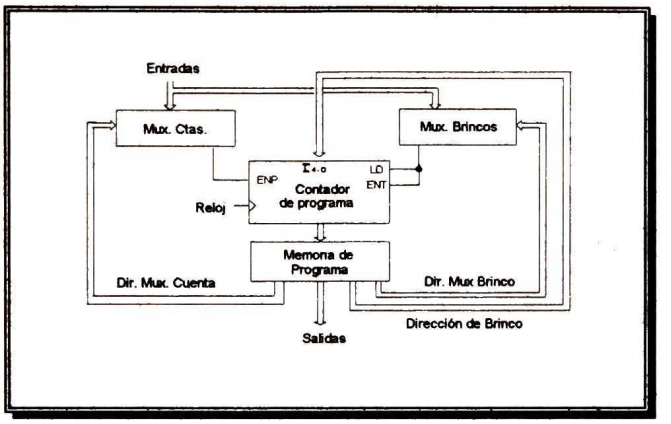


Fig. II.3 Estructura de un controlador microprogramado básico.

Los controladores microprogramados básicos, se tratan de una variante de un controlador basado en un contador. La diferencia es que ahora el contador se llamará "CONTADOR DE PROGRAMA" (PC), y direccionará a la memoria de programa, la cual contiene ahora -además de las salidas- las direcciones de selección de los multiplexores de variables de entrada y las direcciones de los brincos posibles. La estructura de los datos almacenados en la memoria se muestra más adelante.

La arquitectura básica de controladores microprogramados, permite modificaciones en su funcionamiento mediante el cambio del programa que se tiene en memoria, es decir, se puede modificar por **"software"**.

Se puede resolver el diseño con controladores microprogramados -partiendo de una Carta ASM- de acuerdo con los siguientes pasos:

1. Asignar códigos a los estados elaborando la "tabla de estados presentes e instrucciones" aplicadas en cada estado.

   En los controladores cableados, para la asignación de estados se tiene que considerar el circuito en que se basa para favorecer las operaciones naturales de éste y así evitar bloques de lógica residual, con los controladores microprogramados no tenemos este problema, ya que los datos para las cargas en paralelo (que podrían generar bloques de lógica residual) están almacenados en la memoria de programa. La "tabla de estados presentes e instrucciones" tiene la siguiente estructura:

NOMBRE DEL ESTADO	CÓDIGO DEL ESTADO	OPERACIÓN

2. De acuerdo a la tabla anterior y conociendo la "tabla de control de modo" del contador, se elaborará una tabla donde se muestren los valores de las terminales de control del contador (ENP, LOAD), necesarias para realizar la operación definida en cada estado. La tabla anterior tendrá la siguiente estructura:

NOMBRE DEL ESTADO	VALOR DE ENP	VALOR DE LOAD

Como se puede ver, los dos primeros pasos del diseño del controlador microprogramado son los mismos que para el diseño del controlador basado en un contador.

Los pasos 1 y 2 pueden realizarse simultáneamente dando como resultado una tabla como la siguiente:

NOMBRE DEL ESTADO	CÓDIGO DEL ESTADO	OPERACIÓN	ENP	LOAD

3. Crear los mapas de acción para los multiplexores (MUX. CUENTAS y MUX. BRINCOS).

   Con respecto a este paso, no es necesario alambrar los multiplexores tal y como indican los mapas de acción; bastará con colocar las variables de entrada en los multiplexores sin repetirlas y anotar en el programa su dirección tantas veces como las necesitemos. Esto se puede hacer porque el contador ya no controla las direcciones de selección de los multiplexores como en las arquitecturas anteriores, sino que ahora las direcciones para estos multiplexores están contenidas en la memoria de programa, así cada localidad de la memoria (que corresponde a un estado de la carta) contendrá la dirección (en los multiplexores de entrada) de la variable que determina la operación a realizarse.
4. La carga en paralelo (o dirección de brinco) será alimentada directamente desde la memoria, por lo que no necesitaremos mapas de carga en paralelo y nos ahorraremos funciones para alambrar.

   De acuerdo a los pasos anteriores y siguiendo un formato compatible a la distribución de datos mostrados en la memoria dentro de la Fig. II.3, tendremos finalmente una tabla con los siguientes datos:

DIRECCIÓN DEL PC	DIR. MUX. CUENTA	SALIDAS (0→n)	DIR. DE BRINCOS	DIR. MUX. BRINCOS

La DIRECCIÓN DEL PC, es realmente el valor que envía el contador a la memoria, es decir, realmente significa o indica una localidad de memoria, los datos que se deben grabar en está están dados por: DIR. MUX. CUENTA, SALIDAS (0→n), DIR. DE BRINCOS y DIR. MUX. BRINCOS.

5. Escritura del programa. Los datos: DIR. MUX. CUENTAS, SALIDAS (0→n), DIR. DE BRINCO y DIR. MUX. BRINCOS; se agrupan de cuatro en cuatro -de derecha a izquierda- y se convierten en su valor hexadecimal para grabarlos en la memoria.
6. Alambrado. Este paso se realiza respetando la estructura mencionada en la Fig. II.3, teniendo cuidado de colocar correctamente las variables de entrada en los multiplexores correspondientes.

II.2.4 Controladores con número fijo de instrucciones (arquitectura MICA I)

La estructura de un controlador microprogramado vista en el punto anterior, genera las instrucciones mediante la combinación de las variables de entrada (filtradas o seleccionadas a través del multiplexor de cuentas y el de brincos) conectadas a las terminales de selección del contador (ENP y LOAD). Para cambiar una instrucción por otra, necesitamos escoger otra combinación desde los multiplexores CUENTAS y BRINCOS. Se puede aumentar la versatilidad del controlador si podemos escoger la instrucción que se dará al contador mediante un código de operación (CÓDIGOPER) de tal manera que las decisiones dependan de una sola variable de entrada (bandera o BND) que es elegida por un multiplexor de entradas, que no aparece en la arquitectura microprogramada básica.

Con esta estructura estaremos usando un módulo MICA I que generará la siguiente dirección de la memoria gracias al código de operación recibido de la memoria misma, considerando el valor de la bandera. Una vez construida la arquitectura, las instrucciones que se generarán serán más independientes de los dispositivos físicos, lo que nos permitirá concentrarnos más en el trabajo de diseño del programa que en el diseño físico del controlador.

Las instrucciones que recibe un modulo MICA I son:

MNEMÓNICO	BANDERA	ACCIÓN		DESCRIPCIÓN
CC (BND)	0	RETIENE:	PC ← PC	CUENTA
	1	CUENTA:	PC ← PC + 1	CONDICIONAL
CI	0	CUENTA:	PC ← PC + 1	CUENTA
	1	CUENTA:	PC ← PC + 1	INCONDICIONAL
BC (BND) <DIR>	0	RETIENE:	PC ← PC	BRINCO
	1	BRINCA:	PC ← DIR	CONDICIONAL
BI <DIR>	0	BRINCA:	PC ← DIR	BRINCO
	1	BRINCA:	PC ← DIR	INCONDICIONAL
C/BC (BND) <DIR>	0	CUENTA:	PC ← PC + 1	CUENTA / BRINCO
	1	BRINCA:	PC ← DIR	CONDICIONAL

Esta tabla es utilizada para diseñar la estructura interna del módulo MICA I.

Es de hacer notar que una condicionante en el diseño con MICA I, es que sólo se pueden tener hasta dos trayectorias de salida en cada bloque ASM presente, y además, no puede existir más de un diamante de decisión en cascada.

La arquitectura de trabajo para diseñar un controlador auxiliados de un módulo MICA I se muestra en la siguiente figura:

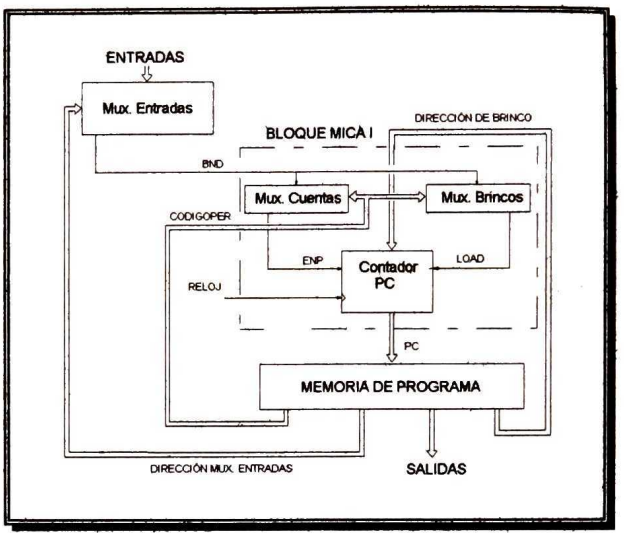


Fig. II.4 Estructura de un controlador con arquitectura MICA I.

En la MEMORIA DE PROGRAMA se encuentran grabadas las salidas de nuestro sistema, además, contiene el código de la operación (CÓDIGOPER) que se aplicará al módulo MICA I para determinar la localidad de la siguiente instrucción del programa en memoria. La instrucción derivada del CÓDIGOPER se ejecuta tomando en cuenta la bandera (BND) proveniente del multiplexor de variables de entrada, cuya selección se realiza por medio de la DIRECCIÓN DE LA VARIABLE DE ENTRADA, localizada también en la memoria.

El CÓDIGOPER es un valor binario asignado a cada una de las operaciones definidas; como tenemos cinco operaciones, necesitamos sólo 3 bits para definirlas. El código que le asignemos a cada una de las operaciones puede variar, pero una vez definido lo fijaremos para todo el diseño.

A continuación se muestra una tabla de operaciones con una asignación de códigos:

CÓDIGOPER	OPERACIÓN	MNEMÓNICO
000	Cuenta condicional.	CC
001	Cuenta incondicional.	CI
010	Brinco condicional.	BC
011	Brinco incondicional.	BI
100	Cuenta o Brinco condicional	C/BC

Cabe señalar que si cambiamos la asignación de los códigos de operación, el alambrado de la estructura interna de la arquitectura también cambiará.

En esta arquitectura, como en la anterior, tendremos cuidado de colocar correctamente las variables de entrada en las correspondientes entradas del multiplexor de entradas, de acuerdo al diseño que se haga; con la diferencia que aquí sólo tendremos un multiplexor, ya que los dos de la arquitectura anterior (mux. de cuentas y mux de brincos) ahora son parte interna de esta nueva arquitectura.

El proceso para diseñar un controlador mediante la técnica MICA I es más sencillos si los comparamos con el de los controladores cableados. Partiendo de una Carta ASM ya diseñada tendremos:

1.- Asignación de estados y determinación de las instrucciones MICA I necesarias para producir el cambio de estado deseado, en asociación con una única bandera.

2.- Escritura del programa con mnemónicos mencionando la bandera asociadas y las direcciones de los brincos.

3.- Traducción de los mnemónicos a datos binarios. Éste es el programa que se grabará en memoria.

4.- Alambrado del controlador colocando las variables de entrada en las posiciones mencionadas en el programa.

En este punto no mencionaremos más sobre esta arquitectura, ya que más adelante la analizaremos a detalle.

II.2.5 Controladores que manejan subrutinas (arquitectura MICA II)

Al igual que en un programa de computadora, la capacidad de manejo de subrutinas es una herramienta importante dentro del diseño de controladores digitales, podemos manejar esta capacidad utilizando la arquitectura MICA II, que puede manejar también subrutinas anidadas.

Existen casos en los que dado un sistema controlador, sería deseable que se tuviera la posibilidad de evaluar una condición en un estado X para luego saltar hacia un conjunto separado de instrucciones, y volver después al estado siguiente a X. En el desarrollo de software, este tipo de operaciones se conoce como "salto a subrutina", lo que se aplica al manejo de rutinas necesarias en problemas de control que deben ser repetidas una y otra vez bajo distintas condiciones y en distintos periodos durante el ciclo de vida de una secuencia de control. El manejo de subrutinas permite la fragmentación de rutinas, en elementos menores más sencillos.

La arquitectura de trabajo para diseñar un controlador auxiliados de un módulo MICA II se muestra en la siguiente figura:

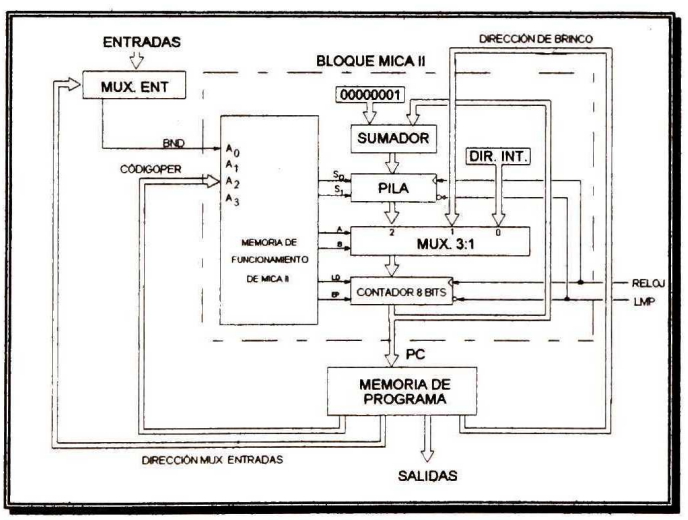


Fig. II.5 Estructura de un controlador con arquitectura MICA II.

El desarrollo de la capacidad del manejo de subrutinas requiere de algunas características adicionales a las presentadas en MICA I, y éstas son:

1.- La introducción de la instrucción "salto a subrutina" (SAS) en presencia de una bandera.
2.- La introducción de una instrucción de "regreso de subrutina" (RDS) en presencia de una bandera, y continuación del flujo original.
3.- Hardware adicional que facilite los pasos 1 y 2.

Adicionalmente, se necesita de la definición de la secuencia principal de control que facilite el uso de las subrutinas y asegure el retorno adecuado de las mismas. Un regreso de subrutina colocará al controlador en un estado X + 1, si la instrucción que evalúa a la condición de salto se halla en el estado X.

Para cumplir con el punto (1) definiremos:

I.- Una instrucción de "salto a subrutina" (SAS) en condición de bandera, que de acuerdo al valor de la bandera (BND), cargue la nueva dirección y almacene la original en un área de memoria tipo pila.
II.- La inserción de una instrucción de "salto a interrupción" (INT) será útil para el establecimiento de interrupciones.
III.- Una instrucción de "retorno de subrutina" (RDS), que recuperará la dirección del salto almacenada en el área de memoria tipo pila.

En cuanto al hardware, se añaden tres bloques fundamentales que hacen posible el cumplimiento de los incisos (I) y (II):

⇨ Una estructura de multiplexor 3:1 de 8 bits hará posible la elección de la dirección siguiente desde tres puntos distintos, en caso de brincos:

   1.- La entrada de la dirección de interrupción DIR. INT., para la instrucción INT.
   2.- La entrada de la dirección normal de salto (DIRECCIÓN DE BRINCO) que proviene de la memoria de programa.
   3.- La entrada proveniente del nivel superior del área de memoria tipo PILA.

⇨ Un sumador que proporcione el valor de la localidad de memoria a la que se regresará cuando se presente una llamada a subrutina y ésta termine. El valor requerido se obtiene al sumar la unidad a la dirección donde se encontró la instrucción de salto.

⇨ Un stack que es una memoria tipo PILA de acceso secuencial, usada para almacenar y proporcionar información. Existe un circuito (el 745194) con esa función y que además permite anidar cuatro niveles de subrutinas.

Con estos cambios se incrementa la flexibilidad y el poder de control de la arquitectura.

Note que la estructura externa de este sistema es semejante a la de MICA I, los pasos de diseño de un controlador basado en esta arquitectura también son muy semejantes, de tal manera que la diferencia real entre estos dos tipos de controladores estriba en su capacidad de funcionamiento relacionado con el aumento de

instrucciones (aquellas que manejan subrutinas), por lo que la verdadera diferencia está en la estructura interna, resultante del aumento de instrucciones que maneja.

Analizando el hardware que nos permite el manejo de subrutinas y comparándolo con el de la arquitectura MICA I, encontramos diferencias substanciales; una muy importante es la existencia de una memoria dentro del módulo MICA II (más adelante será explicada), también observamos que los multiplexores, mux. cuentas y mux. brincos, de MICA I, ya no existen dentro del módulo MICA II, donde en cambio tenemos un circuito sumador, una pila y una arquitectura de multiplexor especial 3:1, donde la línea de datos y la línea de salidas son buses de 8 bits cada una. La forma de obtener el código de los estados, en las dos arquitecturas, es mediante un contador, al que llamamos contador de programa o PC.

Las entradas al sumador son, por una parte el valor actual del contador de programa (PC) y por la otra, el valor binario uno (00000001); con esto, la salida del sumador (que es la entrada a la pila) siempre tendrá el valor del estado actual más uno, que en caso de que en el estado actual se realice un salto a subrutina, entrará a la PILA, y posteriormente cuando se encuentre la instrucción de regreso de subrutina (RDS) este valor será tomado de la PILA para realizar el regreso a la instrucción siguiente de la llamada.

Las entradas a la arquitectura de multiplexor 3:1 son tres buses de 8 bits; mediante las terminales de dirección del circuito se elegirá cuál bus de datos llegará a las entradas en paralelo del contador. Se elegirá el bus 0 si se realiza una llamada a una interrupción (el manejo de la interrupción es independiente de la arquitectura), el bus 1 si se requiere un salto a subrutina, dentro de la subrutina debe existir una instrucción de regreso de subrutina (RDS) para devolver el control a la secuencia principal, cuando esto ocurre, se direcciona el bus 2 en donde se encuentra el valor que sale de la PILA.

La memoria interna identificada como MEMORIA DE FUNCIONAMIENTO DE MICA II contiene los datos de control de los elementos internos, como son: terminales de control para el circuito de la PILA, terminales de direccionamiento para la arquitectura de multiplexor 3:1, y terminales de control para el contador de programa. El direccionamiento de esta memoria está formado por el código de control (proveniente de la memoria de programa) y por el valor de la bandera (BND), proveniente del multiplexor de entradas. La estructura de la información guardada en la memoria es la siguiente:

OC_2	OC_1	OC_0	BND	ENP	LD	S_a	S_b	S_0	S_1
LOCALIDAD DE MEMORIA			TERMINALES DE CONTROL DEL PC			TERMINALES DE DIRECCIONAMIENTO DEL MUX. 3:1		TERMINALES DE CONTROL DE LA PILA	

Tabla 1.- Estructura de la información contenida en la MEMORIA DE FUNCIONAMIENTO DE MICA II.

Las instrucciones para el diseño con MICA II incluyen las cinco de MICA I, más tres para el manejo de subrutinas, que nos dan un total de ocho. La definición y características de estas instrucciones se graban dentro de la MEMORIA DE FUNCIONAMIENTO DE MICA II como se muestra a continuación.

DIR	MNEMÓNICO	CÓDIGOPER	BND	ACCIÓN	INSTRUCCIÓN
0	CC (BND)	000	0	RETIENE: PC ← PC	CUENTA
1			1	CUENTA: PC ← PC + 1	CONDICIONAL
2	CI	001	0	CUENTA: PC ← PC + 1	CUENTA
3			1	CUENTA: PC ← PC + 1	INCONDICIONAL
4	BC (BND) <DIR>	010	0	RETIENE: PC ← PC	BRINCO
5			1	BRINCA: PC ← DIR BRC	CONDICIONAL
6	BI <DIR>	011	0	BRINCA: PC ← DIR BRC	BRINCO
7			1	BRINCA: PC ← DIR BRC	INCONDICIONAL
8	C/BC (BND)	100	0	CUENTA: PC ← PC + 1	CUENTA / BRINCO
9	<DIR>		1	BRINCA: PC ← DIR BRC	CONDICIONAL
10	SAS (BND) <DIR>	101	0	CUENTA: PC ← PC + 1	SALTO A
11			1	BRINCA: PC ← DIR SUB; TOP ← PC + 1	SUBRUTINA
12	RDS (BND) <DIR>	110	0	BRINCA: PC ← DIR BRC	REGRESO DE
13			1	BRINCA: PC ← TOP	SUBRUTINA
14	INT (BND) <DIR>	111	0	CUENTA: PC ← PC + 1	SALTO A
15			1	BRINCA: PC ← DIR INT; TOP ← PC + 1	INTERRUPCIÓN

Tabla 2.- Instrucciones de la arquitectura MICA II.

Nota: TOP es el tope de la pila, que se actualiza automáticamente.

De acuerdo con la tabla anterior, siguiendo la estructura mostrada en la tabla 1, y considerando la tabla de control del contador, la de la pila y el modo de direccionamiento de la arquitectura de multiplexor 3:1, obtenemos el contenido de la memoria de funcionamiento de MICA II, que se muestra a continuación

ENP	LOAD	ACCIÓN
1	1	CUENTA
0	0	BRINCA
0	1	RETIENE

TABLA DE CONTROL DEL 74161

S_1	S_0	ACCIÓN
0	0	RETIENE
0	1	INTRODUCE
1	0	SACA

TABLA DE CONTROL DE LA PILA

	LOCALIDAD DE MEMORIA				TERMINALES DE CONTROL DEL PC		TERMINALES DE DIRECCIONAMIENTO DEL MUX. 3:1		TERMINALES DE CONTROL DE LA PILA		VALOR HEXADECIMAL DEL CONTENIDO
DIR	OC_2	OC_1	OC_0	BND	EP	LD	S_A	S_B	S_0	S_1	HEX
0	0	0	0	0	0	1	0	0	0	0	10
1	0	0	0	1	1	1	0	0	0	0	30
2	0	0	1	0	1	1	0	0	0	0	30
3	0	0	1	1	1	1	0	0	0	0	30
4	0	1	0	0	0	1	0	0	0	0	10
5	0	1	0	1	0	0	0	1	0	0	04
6	0	1	1	0	0	0	0	1	0	0	04
7	0	1	1	1	0	0	0	1	0	0	04
8	1	0	0	0	1	1	0	0	0	0	30
9	1	0	0	1	0	0	0	1	0	0	04
10	1	0	1	0	1	1	0	0	0	0	30
11	1	0	1	1	0	0	0	1	0	1	05
12	1	1	0	0	0	0	0	1	0	0	04
13	1	1	0	1	0	0	1	0	1	0	0A
14	1	1	1	0	1	1	0	0	0	0	30
15	1	1	1	1	0	0	0	0	0	1	01

La información mostrada puede variar si se cambia el contador con que se diseñe, si se implementa la pila de otra manera cambiando su tabla de control, o simplemente si cambiamos la posición de las entradas a la arquitectura de multiplexor 3:1.

Resumiendo, el controlador denominado MICA II es un módulo que genera la dirección de la siguiente instrucción en una memoria gracias al código de operación recibido de ella en combinación con una bandera BND representativa de las variables de entrada. La característica a resaltar en este módulo, es su capacidad de brincar a una dirección para iniciar una rutina que será ejecutada cuantas veces sea necesario, hasta que encuentre una instrucción de retorno y la bandera esté encendida. En este caso se dará por terminada la subrutina y se continuará con el programa en la instrucción siguiente a la que se dejó al pasar a esta subrutina. Además, podemos detener la ejecución del proceso con una instrucción de interrupción (INT).

Cuando se ejecuta la instrucción SAS, mientras que se salta a la dirección donde se inicia la subrutina, se realiza también una instrucción PUSH a la pila, en donde se guarda el PC + 1; esto equivale a decir que se está guardando (en este caso en la variable TOP) la dirección de la siguiente instrucción a ejecutarse cuando se retorne de la subrutina. Esto se demuestra en la instrucción RDS en donde el PC se restaura a la siguiente

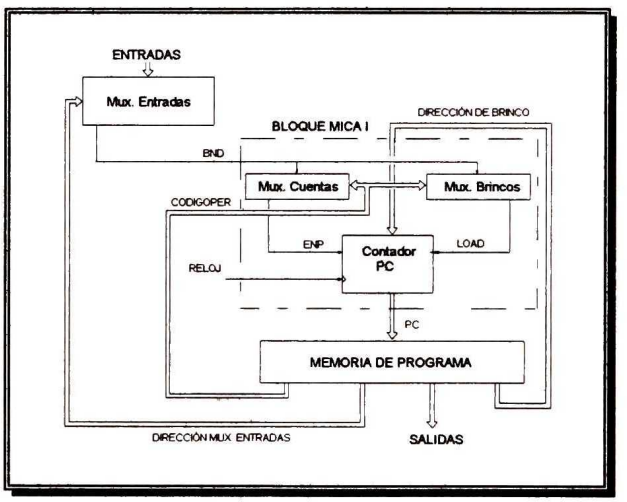
instrucción mediante una instrucción POP a la pila, lo cual significa que se regresa (en este caso mediante la variable TOP) a la siguiente instrucción de la que se ejecutó al saltar a la subrutina.

II.3 Arquitectura MICA I

En el punto II.2.4 se dio una introducción a esta arquitectura y se mostró su diagrama, ahora haremos una descripción detallada y se mostrará el proceso de desarrollo del controlador hasta la obtención del diagrama del circuito físico. Para empezar se hará una descripción de los elementos de la arquitectura, sus características, su funcionamiento y el modo en que interactuan; para al final mostrar los pasos de diseño.

II.3.1 Descripción de los elementos de la arquitectura

Para llevar a cabo la descripción de la arquitectura, recurriremos al diagrama físico de un controlador basado en ella, mostrado en la Fig. II.4 y que lo reproducimos a continuación.



Estructura de un controlador con arquitectura MICA I.

Aunque la estructura interna del controlador es sencilla -sólo contiene tres módulos-, proporciona gran flexibilidad al diseñador y es la base para arquitecturas más completas.

A continuación listaremos los elementos internos que forman la arquitectura.

- **Mux. Cuentas**. Circuito multiplexor 8:1 (74151). Controla o define el valor de la terminal ENP del contador PC.
- **Mux. Brincos**. Circuito multiplexor 8:1 (74151). Controla o define el valor de la terminal LOAD del contador PC.
- **Contador PC**. Circuito contador (74161). Contador de 4 bits con bus de entradas en paralelo, capaz de recibir un dato y continuar su cuenta a partir de éste, este circuito cuenta con terminales de control (LOAD, ENP y ENT) para elegir la operación a realizar (a saber, CUENTA, RETIENE o CARGA), tiene también una entrada de reloj y una de reset (RST) para inicializar su conteo (empezar de cero). En este trabajo utilizaremos el circuito integrado (CI) 74161, aunque podemos utilizar cualquiera otro que cuente con características semejantes y que permitan la realización de las instrucciones de MICA I. Cabe señalar

que de utilizar un circuito diferente al 74161, los resultados y la forma de alambrar la estructura interna del controlador puede cambiar, debido a que la forma de controlar al contador puede ser diferente, pero el procedimiento de diseño se mantiene. A este circuito lo identificaremos como CONTADOR DE PROGRAMA o PC, y como se dijo antes, el contador 74161 es un contador de cuatro bits, lo que limita el número de estados posibles a manejar en la Carta ASM a 16, esta limitación se puede superar si construimos un arreglo de contadores de manera tal que podamos tener una secuencia de conteo mayor.

Junto a los anteriores elementos se tienen también los siguientes:

- **Memoria de Programa.** Este módulo en realidad, puede ser un conjunto de circuitos integrados de memoria, organizados de tal manera que satisfagan nuestras necesidades, por ejemplo, si el número de localidades de memoria excede a los disponibles por un solo circuito, tendremos que formar un arreglo de memorias de manera tal que incrementemos el número de localidades direccionables; de forma semejante, si la cantidad de información que necesitamos almacenar en una localidad de memoria es mayor al tamaño de la palabra manejada por el circuito, también tendremos que formar un arreglo que satisfaga nuestras necesidades. Todo lo anterior dependerá en gran medida del circuito que utilicemos; y como podemos recurrir a varios de ellos (inclusive combinarlos), no mencionaremos a ninguno en especial, pero si diremos que debe tratarse de una memoria que nos permita modificar los datos almacenados en ella, ya que de no ser así, perderíamos parte de la flexibilidad de la arquitectura, puesto que para modificar el funcionamiento del controlador tendríamos que cambiar el circuito físico de memoria; es conveniente decir que no es recomendable usar memorias RAM debido a las característica de éstas; utilizaremos memorias tipo EPROM (Erase Programmable Read Only Memory) o E²PROM (Electrical Erase Programmable Read Only Memory).

- **Mux. Entradas.** Circuito multiplexor. Las características de este circuito dependerán de la cantidad de entradas manejadas en el diseño.

II.3.2 Interacción entre elementos

Como se dijo anteriormente, el CONTADOR DE PROGRAMA o PC cuenta con terminales de control (ENP, ENT y LOAD) para definir la operación a realizar; si consultamos el manual correspondiente podemos obtener la tabla de control del circuito, que nos indica cuales son los valores correspondiente de las terminales de control para realizar cada una de las operaciones disponibles. En la Fig. II.6(A) se muestra la tabla de control completa, podemos ver que los valores para las terminales LOAD y ENT son iguales, por lo que las manejaremos como una sola y la llamaremos LOAD, reduciendo así la tabla, a la mostrada en la Fig. II.6(B).

ENP	ENT	LOAD	ACCIÓN
1	1	1	CUENTA
0	0	0	BRINCA
0	1	1	RETIENE

(A)

ENP	LOAD	ACCIÓN
1	1	CUENTA
0	0	BRINCA
0	1	RETIENE

(B)

Fig.II.6.- Tabla de control del 74161.

La terminal de control ENP está conectada al MUX. CUENTAS que junto con el MUX BRINCOS -cuya salida está conecta a la terminal del control LOAD del contador- proveerán los valores necesarios para que éste realice la operación requerida.

El MUX. CUENTAS y el MUX. BRINCOS trabajan de la misma manera entre ambos, con la diferencia de que el MUX. CUENTAS define el valor de la entrada de control ENP del contador y el MUX. BRINCOS define el valor de la entrada LOAD del mismo contador. Las entradas a los dos multiplexores provienen de la salida del MUX. ENTRADAS y son direccionadas por el CÓDIGO DE OPERACIÓN que se obtiene de la memoria de programa, es decir, cada localidad de memoria (o en otras palabras, cada instrucción del programa) determinará que operación deberá realizarse y de que variable de entrada dependerá la acción a tomarse, esto se obtiene gracias a que el MUX. ENTRADAS también está direccionado desde la memoria de programa y ésta misma proporciona el código de la operación a realizarse. La posición de las variables de entrada en el MUX. ENTRADAS debe corresponder con la posición que indiquemos dentro del programa grabado en la memoria, más adelante se verá como se logra esto.

El MUX. ENTRADAS nos proporcionará el valor de la entrada de la que depende cada operación, este valor es proporcionado en forma de una bandera BND a las entradas de los multiplexores de cuentas y brincos.

Por lo anterior, los valores de ENP y LOAD para el contador serán determinados tanto por el CÓDIGO DE OPERACIÓN como por la DIRECCIÓN MUX. ENTRADAS.

El bloque de MEMORIA DE PROGRAMA contiene la siguiente información:

CÓDIGO DE OPERACIÓN (CÓDIGOPER).- Que como se dijo antes, direcciona al MUX. CUENTAS y al MUX. BRINCOS. Es un código binario de 3 bits, pues sólo necesitamos identificar 5 códigos.

DIRECCIÓN MUX. ENTRADA.- Que determina cual es la variable de entrada que debe evaluarse en cada estado.

SALIDAS.- Es el bus de datos de salida. El número de datos depende de las salidas que hallamos definido en el diseño.

DIRECCIÓN DE BRINCO.- Es un bus de cuatro bits (en el caso de utilizar un solo circuito integrado 74161 como PC) que definirá la dirección a la que se debe pasar en caso de que exista una operación de brinco; si la operación a realizar no involucra brinco alguno, el valor de la dirección de brinco no importa ya que ni siquiera será considerada en ese estado.

La estructura de la información almacenada en la MEMORIA DE PROGRAMA tiene el siguiente formato.

DIR. MEM.				CÓDIGOPER			DIR. MUX. ENT				SALIDAS				DIRECCIÓN DE BRINCO			
D	C	B	A	C	B	A	D	C	B	A	0	1	...	N	I_D	I_C	I_B	I_A

II.3.3 Definición de las instrucciones de la arquitectura

En este punto determinaremos cuales son las instrucciones disponibles en la arquitectura. Podemos decir que las operaciones disponibles están basadas en las características de un tipo de contador como el 74161, por lo que basándonos en la tabla de control de este circuito -mostrada en la Fig. II.6- determinamos que un contador de este tipo puede realizar básicamente tres operaciones: CUENTA, BRINCA y RETIENE.. En base a esto podemos derivar cinco instrucciones donde cada instrucción tendrá dos posibles acciones determinadas por el valor de la

variable de entrada o bandera BND proporcionada por el MUX. ENTRADAS. Las instrucciones las podemos definir como sigue:

INSTRUCCIÓN	ACCIÓN SI LA BND = 0	ACCIÓN SI LA BND = 1
CUENTA CONDICIONAL	RETIENE	CUENTA
CUENTA INCONDICIONAL	CUENTA	CUENTA
BRINCO CONDICIONAL	RETIENE	BRINCA
BRINCO INCONDICIONAL	BRINCA	BRINCA
CUENTA / BRINCO CONDICIONAL	CUENTA	BRINCA

Las acciones mostradas anteriormente las definimos como sigue:

RETIENE: PC ← PC : El PC mantiene su valor actual.

CUENTA: PC ← PC + 1 : El PC incrementa su valor una unidad.

BRINCA: PC ← DIR : El PC toma el valor presente en el bus de entradas en paralelo DIR.

Para identificar cada instrucción le asignaremos un código formado por la combinación de tres variables binarias (C, B, A), a la combinación de estas variables le daremos el nombre de CÓDIGO DE OPERACIÓN (CÓDIGOPER). Se utilizan sólo tres variables puesto que con éstas podríamos identificar hasta ocho instrucciones.

De acuerdo con lo anterior y asignando un código a cada instrucción podemos complementar la tabla anterior y presentarla como sigue:

CÓDIGO C B A	INSTRUCCIÓN	MNEMÓNICO	BND	ACCIÓN
0 0 0	CUENTA CONDICIONAL	CC (BND)	0	RETIENE: PC ← PC
			1	CUENTA: PC ← PC + 1
0 0 1	CUENTA INCONDICIONAL	CI	0	CUENTA: PC ← PC + 1
			1	CUENTA: PC ← PC + 1
0 1 0	BRINCO CONDICIONAL	BC (BND) <DIR>	0	RETIENE: PC ← PC
			1	BRINCA: PC ← DIR
0 1 1	BRINCO INCONDICIONAL	BI <DIR>	0	BRINCA: PC ← DIR
			1	BRINCA: PC ← DIR
1 0 0	CUENTA / BRINCO CONDICIONAL	C/BC (BND) <DIR>	0	CUENTA: PC ← PC + 1
			1	BRINCA: PC ← DIR

II.3.4 Alambrado interno de la arquitectura

De acuerdo a la tabla anterior y considerando la tabla de control del contador, podemos obtener los valores de las terminales ENP y LOAD para cada una de la instrucciones.

MNEMÓNICO	CÓDIGO C B A	BND	ENP	LOAD	ACCIÓN
CC (BND)	0 0 0	0	0	1	RETIENE
		1	1	1	CUENTA
CI	0 0 1	0	1	1	CUENTA
		1	1	1	CUENTA
BC (BND) <DIR>	0 1 0	0	0	1	RETIENE
		1	0	0	BRINCA

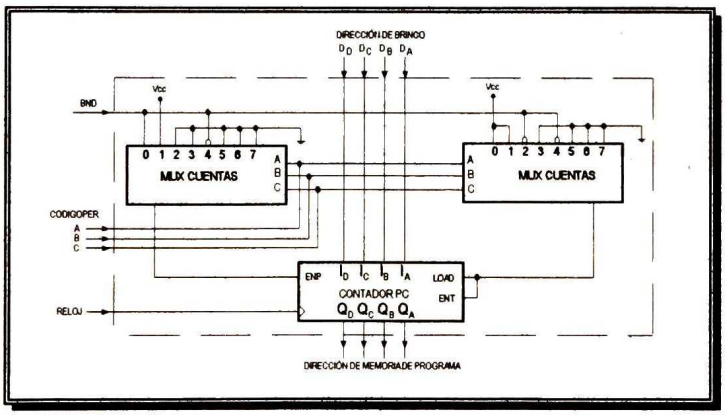
MNEMÓNICO	CÓDIGO C B A	BND	ENP	LOAD	ACCIÓN
BI <DIR>	0 1 1	0	0	0	BRINCA
		1	0	0	BRINCA
C/BC (BND) <DIR>	1 0 0	0	1	1	CUENTA
		1	0	0	BRINCA

De esta tabla, relacionando los valores de ENP y LOAD con los de BND, obtenemos para cada instrucción lo siguiente:

CÓDIGO C B A	ENP	LOAD
0 0 0	BND	1
0 0 1	1	1
0 1 0	0	BND
0 1 1	0	0
1 0 0	BND	BND
1 0 1	*	*
1 1 0	*	*
1 1 1	*	*

Como se puede observar los valores de ENP y LOAD para los códigos 101, 110 y 111 no importan (*) puesto que no existen instrucciones definidas con esos códigos. Los valores (*) los tomaremos como cero.

Una vez hecho lo anterior, podemos definir las conexiones de los elementos internos de la arquitectura como se muestra a continuación:



Como se dijo antes, esta arquitectura cambiará sólo si cambiamos el contador o si cambiamos los códigos de las operaciones, no así si lo que queremos cambiar es el comportamiento del controlador.

II.3.5 Procedimiento de diseño

Una vez que hemos explicado la estructura de la arquitectura y la forma en que se definen las instrucciones usadas en el diseño, estamos en posibilidad de llevar a cabo los pasos necesarios para el diseño de controladores basados en esta arquitectura. Para mostrar los pasos de diseño partiremos de una carta ya

definida en el capítulo anterior (Cartas ASM) y reproducida en la Fig. II.7, donde la salida condicional asociada al estado cinco se cambia por otro estado, de esto podemos obtener otra condicionante de la arquitectura y es que no se pueden tener salidas condicionales, pero esto no es una restricción porque el estado la sustituye sin ningún problema. Así tenemos:

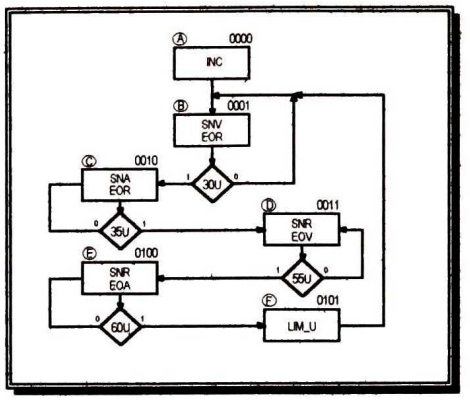


Fig. II.7.- Carta ASM para el controlador de un semáforo.

Paso 1.- DEFINICIÓN DE ESTADOS. Se crea una tabla donde se define un CÓDIGO DE ESTADO para cada uno de los definidos en la carta.

Debemos recordar que una condicionante en el diseño con MICA I es que no se puede tener más de dos trayectorias de salida en cada bloque ASM, y que no puede existir más de un diamante de decisión. La asignación de los códigos de estado se hace buscando que las operaciones que se deban realizar en cada estado estén definidas dentro de MICA I. En nuestro ejemplo tenemos seis estados que definimos como se muestra en la siguiente tabla:

ESTADO	C	B	A
A	0	0	0
B	0	0	1
C	0	1	0
D	0	1	1
E	1	0	0
F	1	0	1

Paso 2.- DEFINICIÓN DE SALIDAS. En una tabla se enlistan las salidas y se relacionan con los estados en los que aparecen.

#	NOMBRE	EDOS. DONDE APARECE
0	INC	A
1	SNV	B
2	SNA	C
3	SNR	D, E
4	EOV	D
5	EOA	E
6	EOR	B, C
7	LIM_U	F

Paso 3.- DEFINICIÓN DE ENTRADAS. En una tabla se enlistan las entradas y muestra los estados con los que se relacionan. El número de la entrada (#) es su posición en el MUX. ENTRADAS.

#	NOMBRE	EDOS. CON QUE SE RELACIONA
0	30U	B
1	35U	C
2	55U	D
3	60U	E

Paso 4.- DEFINICIÓN DE OPERACIONES. Para cada estado se determina la operación a realizar, mostrando el código de ésta y su mnemónico.

EDO	CÓDIGOPER	MNEMÓNICO
A	001	CI
B	000	CC < 30U >
C	000	CC < 35U >
D	000	CC < 55U >
E	000	CC < 60U >
F	011	BI (0001)

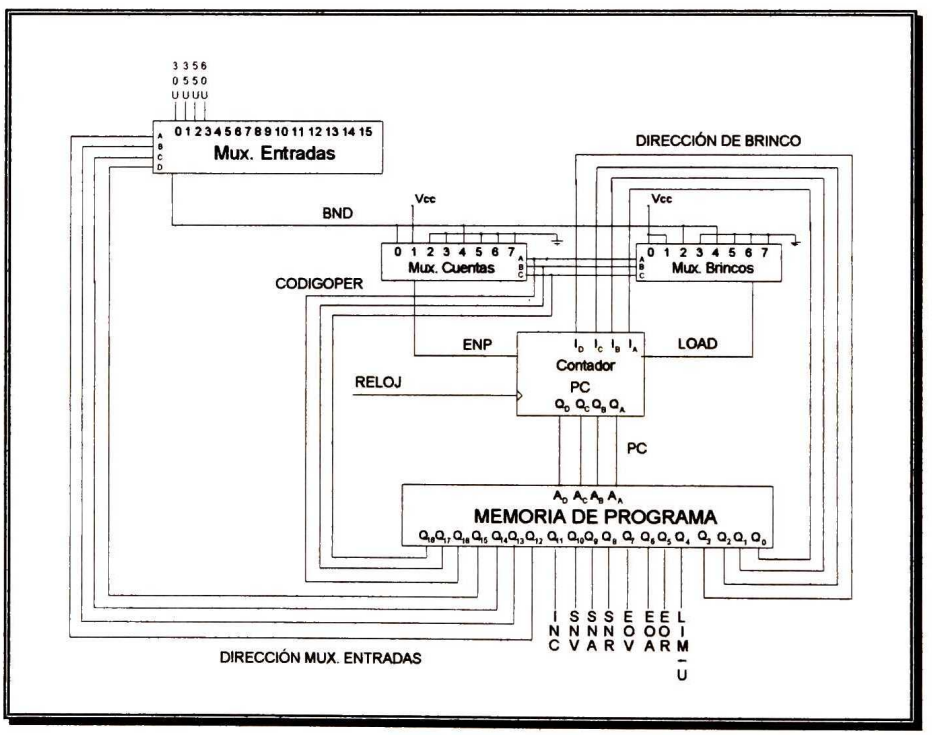
Paso 5.- TABLA DEL MICROPROGRAMA (CÓDIGO BINARIO). En esta tabla se recopila la información de las anteriores y muestra la información en código binario del contenido de la memoria de programa.

L. MEM.	CÓDIGOPER			DIR. MUX_ENT			SALIDAS								DIR. BRINCO			
(EDO)	C	B	A	C	B	A	0	1	2	3	4	5	6	7	D	C	B	A
0000	0	0	1	*	*	*	1	0	0	0	0	0	0	0	*	*	*	*
0001	0	0	0	0	0	0	0	1	0	0	0	0	1	0	*	*	*	*
0010	0	0	0	0	0	1	0	0	1	0	0	0	1	0	*	*	*	*
0011	0	0	0	0	1	0	0	0	0	1	1	0	0	0	*	*	*	*
0100	0	0	0	0	1	1	0	0	0	1	0	1	0	0	*	*	*	*
0101	0	1	1	*	*	*	0	0	0	0	0	0	0	1	0	0	0	1

Paso 6.- TABLA DEL MICROPROGRAMA (CÓDIGO HEXADECIMAL). La información binaria de la tabla anterior se traduce a un código hexadecimal. Los valores * los tomamos como cero.

L. MEM. (EDO)	CONTENIDO HEXADECIMAL
0000	0 8 8 0 0
0001	0 0 4 2 0
0010	0 1 2 2 0
0011	0 2 1 8 0
0100	0 3 1 4 0
0101	1 8 0 1 1

Paso 7.- DIAGRAMA DEL CONTROLADOR. Una vez realizado lo anterior, estamos en la posibilidad de obtener el diagrama final del controlador. Recordemos que es muy importante el colocar las entradas en la posición correcta del multiplexor de entrada, de acuerdo a la "tabla de definición de entradas", de lo contrario el controlador tendrá un funcionamiento no esperado.



CAPÍTULO III

ORIENTACIÓN A OBJETOS COMO METODOLOGÍA DE DISEÑO

CAPÍTULO III

ORIENTACIÓN A OBJETOS COMO METODOLOGÍA DE DISEÑO

III.1 Contexto histórico y enfoque paradigmático

Desde sus inicios en 1950 y hasta los 70´s, la tecnología de la información1 consistía en el proceso de datos, que veía a los sistemas como entidades compuestas de dos partes: información a procesar (datos) y una secuencia de pasos para lograr un fin (proceso), para dar solución a la búsqueda de reducción de costos y tiempo que las empresas solicitaban. Rápidamente los sistemas llegaron a formar parte importante de la planeación estratégica básica de las empresas que adoptaron esa tecnología y una meta dentro de las que no.

El esfuerzo dedicado a poner la tecnología al servicio de los procesos productivos dio por fruto el nacimiento de los lenguajes de alto nivel, y con ellos las técnicas de programación. Comenzaron a desarrollarse los profesionales en el área y fue apareciendo una extensa bibliografía para explicar y facilitar el acceso al innovador uso de las computadoras y sus posibilidades aplicativas.

Conforme se fueron automatizando, las empresas trasladaron el poder de cómputo empleado en sus procesos internos, a la impartición de servicios y productos para sus clientes, en esta traslación, los sistemas cambiaron la base sobre la que eran diseñados, puesto que el usuario final ya no era un especialista, sino una persona que exigía del sistema una facilidad de operación que, en términos del proceso de datos se antojaba impensable.

Durante la década de los 70´s, -bajo la ahora llamada primera era de la informática- los sistemas que se mantenían y desarrollaban en cada instalación fueron creciendo en volumen; cada vez que un nuevo proceso se incluía en el universo local de cómputo, se realizaban y modificaban los parámetros de interacción entre los distintos sistemas ya existentes. La programación se tornó tan compleja, que no era tan fácil hacer software a la medida, pues cada pieza era diseñada y producida para satisfacer las necesidades específicas del usuario y de los propios sistemas; un software hecho por especialistas que intentaban utilizar al máximo las herramientas de que disponían para cumplir con el tiempo requerido, dejando de lado toda facilidad posible con tal de no incrementar las largas listas de instrucciones, escritas línea por línea para interpretar los requerimientos básicos. Para entonces la ingeniería del software[2], comenzaba a imponer orden en la planeación y control del proceso de desarrollo de los sistemas.

En la década de los 80´s, los sistemas se fueron comprometiendo en procesos cada vez mayores y, aún con la ingeniería aplicada, la vasta mayoría de los programas seguían teniendo características de arte; complejos y a menudo grandes, su desarrollo exigía mucho tiempo, codificándose de acuerdo a un conjunto específico de requerimientos y partiendo de cero para su construcción; haciendo uso de técnicas que ni son mensurables ni pueden repetirse consistentemente. El autor de cada programa se convertía así en su único intérprete y para el mantenimiento ulterior, podía resultar mejor opción un nuevo desarrollo a desenmarañar las líneas de un estilo

---

1 La tecnología de la información es el conjunto de procesos y técnicas que, conjuntamente con la tecnología electrónica especializada han permitido la incorporación creciente del manejo electrónico de datos a los procesos productivos, incrementando la eficiencia, velocidad y productividad. En la práctica, la tecnología de la información se traduce en el análisis de requerimientos, selección del equipo adecuado y todo el proceso de desarrollo de sistemas que traen por resultado una herramienta software-hardware altamente especializada y que el usuario final del sistema podría usar.

2 La aplicación de un aprovechamiento sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento del software.

de programación ajeno. Es un aprovechamiento preindustrial, haciendo una analogía a la revolución industrial del siglo XVIII.

En la práctica, el proceso de desarrollo de software cada vez requería de más tiempo, recursos de cómputo y dinero, la planeación y organización no eran ya suficientes para asegurar la creación de software efectivo dentro del tiempo y presupuesto adecuados. Los sistemas se hacinaban, crecían, se volvían obsoletos, y se remendaban dentro de los sistemas de almacenamiento. Las herramientas CASE (Ingeniería de Software Auxiliada por Computadora), pensadas para minimizar esa tendencia no encontraban eco en los nuevos desarrollos, pues su aprovechamiento requería un nuevo software capaz de asimilarlas.

Para finales de la década, las amistosas interfaces gráficas de usuario (GUI -Graphical User Interface-, por sus siglas en inglés) vinieron a poner un nuevo requerimiento al software; iniciando una tendencia que ha popularizado el uso de la computadora en sectores de la sociedad que se oponían al manejo de CUI (Interface de usuario basada en caracteres).

Una interface de usuario es un ambiente que pone a disposición del operador un medio para comunicarse con la computadora; desde sus inicios, la programación y operación de los equipos de cómputo han contado con un conjunto de instrucciones, operadores y comandos codificados, que interactúan con un hardware diseñado para poder "leer" los códigos correspondientes a los botones que conforman un teclado; por otra parte la capacidad de vídeo de los equipos extendió el concepto de las CUI´s en términos de menús, ventanas de texto, barras de mensaje y líneas de comando. Los ambientes CUI´s dominaban la escena, haciendo menester aprender conjuntos de instrucciones, comandos especiales y técnicas de manejo de la información especializadas, hasta el momento en que fue técnicamente posible el empleo de elementos gráficos en los monitores de vídeo.

Fue así como las instrucciones pudieron traducirse en iconos -pequeños dibujos representando la acción buscada-, llenando la pantalla de figuras familiares que podían ser manejadas con un dispositivo apuntador que, análogo al dedo índice, permitía señalar acciones, en lugar de deletrearlas usando el teclado. Con la nueva facilidad, las personas se acercaron con menos temor, y hasta con entusiasmo a las computadoras; la demanda de las GUI´s creció rápidamente, empujando a los programadores de vanguardia un poco más hacia el enfoque de una mayor facilidad de uso, lo que para un Sistema Orientado al Proceso (SOP), significó mayor complejidad en los algoritmos, herramientas de programación e interacción con el hardware (Fig. III.1).

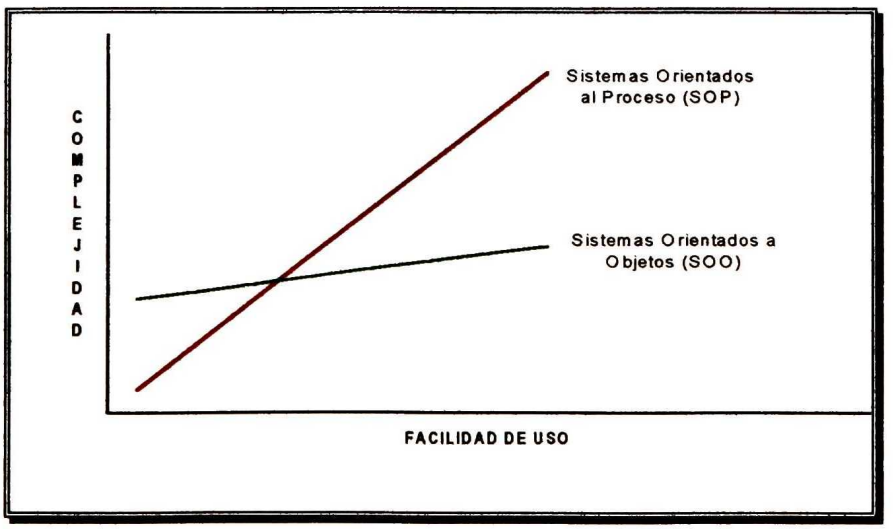


Fig. III.1.- Gráfica facilidad de uso/complejidad del software.

Adicionalmente a la problemática de orientar los nuevos desarrollos al usuario final, se presentó el problema de actualizar los sistemas ya existentes, los que al estar ubicados en un solo centro de cómputo, centralizaban también su actualización, cosa que facilita por una parte el mantenimiento pero que generaliza problemas que de otra forma serían vistos como locales. De ese modo la creciente complejidad de los sistemas comenzó a provocar costosos retardos en la entrega del producto final, la cancelación de proyectos o hasta su rediseño completo, siendo precisamente este último punto el que obligó a la comunidad informática a identificar los elementos que aparecían como aristas comunes a todos los desarrollos de sistemas basados en un punto de vista procedimental (SOP) y que influían grandemente en la decisión de un rediseño:

1.- Existe una limitada reusabilidad y portabilidad del software.

2.- El mantenimiento es engorroso y caro.

3.- Los sistemas forman islas de tecnología.

4.- La curva del aprendizaje del sistema es asintótica.

5.- Existe una dependencia con la plataforma de desarrollo.

6.- Sólo los especialistas pueden crear el software.

7.- El software se enfoca al proceso.

Este conjunto de características conforman la problemática producida por una forma de reflejar la realidad dentro de los sistemas, un esquema que permite comprender el mundo que nos rodea, un marco de referencia o un modelo, es decir, un **paradigma**. Para el caso que nos ocupa, ésta "Crisis del Proceso de Datos" conformó el ambiente que dio origen a nuevos enfoques alternativos (ver Tabla "Identificación de Paradigmas"), que de acuerdo a las nuevas características de los problemas, replantearon las necesidades y su solución, constituyéndose en nuevos estándares que más tarde se reconocerían como nuevos paradigmas, entre los que se encuentra la Orientación a Objetos (OO) que, como muestra la Fig. III.1 tienden a disminuir la componente de la complejidad propia de los sistemas altamente especializados o mayormente orientados al usuario.

IDENTIFICACIÓN DE PARADIGMAS EN LA CRISIS DEL PROCESO DE DATOS		
ASPECTO	**PARADIGMA ANTERIOR**	**NUEVO PARADIGMA**
Método del desarrollo de software	La calidad, tiempo y costo de desarrollo del software está en función de las habilidades y la creatividad del profesional que lo desarrolla.	Los desarrolladores usan y reutilizan módulos o partes de módulos previamente estandarizados y que trabajan juntas. * Paradigma de la orientación a objetos
Diseño y construcción de Interface de usuario	La CUI (Character User Interface) encripta las instrucciones disponibles en forma de tablas de códigos, para utilizar y administrar el sistema.	En la GUI (Graphical User Interface) el usuario manipula intuitivamente imágenes o iconos directamente relacionados con acciones. * Paradigma de la interface gráfica de usuario.
Diseño de aplicaciones Comerciales	Se diseña la aplicación particular con sus propios estándares y facilidades. Se generan sistemas aislados.	Las aplicaciones comparten facilidades y estandarizan sus características funcionales. * Paradigma de la Manufactura Integrada del Software.
Redes y sistemas distribuidos	El proceso productivo se centraliza junto con la administración de la empresa en una computadora central o Host.	El proceso se distribuye a lo largo de las distintas entidades informáticas de la empresa. Arquitectura Cliente-Servidor. * Paradigma del proceso distribuido.
Multimedia	Cada presentación de la información se trata por separado con su propia tecnología.	La digitalización generalizada y los estándares acordados permiten el manejo simultáneo dentro de las aplicaciones. * Paradigma de la integración de medios.

III.2 El paradigma de la orientación a objetos

En un sistema informático, los elementos del universo de un problema intentan ser reflejados o "modelados" como entidades internas mediante técnicas que, dentro de los métodos de desarrollo del software, se presentan en tendencias claramente distinguibles entre sí y que reciben su nombre de acuerdo a la forma de conceptualizar al objeto de estudio. Históricamente el método llamado Procedural o Procedimental, que se enfoca a los procedimientos, ha dominado por sobre el resto, ya que sigue los preceptos tradicionales para la resolución de problemas de la ingeniería, suponiendo a un proceso formado de varios subprocesos que al analizarse facilitan el conocimiento del problema y permiten visualizar la solución como una interconexión de ellos (orientación al proceso). Para este enfoque, el "modelo" y su solución son integrados finalmente, después del proceso de desarrollo específico, en un conjunto de archivos que contendrán la información y programas o procedimientos encargados de manipularla.

La actual inclinación de la tecnología por el paradigma de la OO puede interpretarse como una preferencia a observar los problemas de una forma en la que se facilite la reutilización de elementos previamente desarrollados, derivando su poder de modelado de la funcionalidad y manipulación subyacentes, en el concepto abstracto de objeto.

Un SOO, en oposición a la descomposición funcional -base de la orientación al proceso- modela al sistema como una colección de "prototipos" de los objetos que componen la realidad. El análisis y diseño se efectúa sobre los prototipos, vistos como objetos o unidades atómicas encapsuladas que contienen a la vez datos y su proceso, y sobre la manera en que interactúan entre sí mediante "mensajes", en la forma en que son agrupados en colecciones, en la forma en que éstas serán manipuladas y en su jerarquía estructural. Su diseño detallado evoluciona y se difiere hasta después, en el proceso de desarrollo.

El paradigma de la OO observa, como el paradigma procedural, técnicas para el análisis, el diseño y la programación y, como aquel, recomienda su aplicación de manera uniforme a lo largo del proceso de desarrollo de un sistema para lograr mayor efectividad. En la práctica, la OO, no garantiza la realización del mejor software, si no que ofrece una herramienta para tener mayor control en su producción y mantenimiento acelerados.

Los beneficios de la utilización de este paradigma se reconocen mayormente al aplicar su técnica a sistemas masivos, adaptándose rápidamente a los cambios significativos en la aplicación, lo que empleando el paradigma procedural, necesitaría más rigor en el control del desarrollo de módulos, documentación, estandarización, mantenimiento y manejo de errores. La siguiente es una tabla comparativa de los paradigmas identificados por Henderson-Seller, B:

TIPO DE PARADIGMA	CARACTERÍSTICAS
Procedural	El lenguaje es esencialmente imperativo. Se dispone de estructuras de control como la secuencia, la iteración y la selección. El diseño basado en este paradigma y en la descomposición funcional es la base del método de desarrollo y su aprovechamiento.
Lógico	Se enfoca a las reglas y relaciones de inferencia implícitas en el cálculo proposicional. Se caracteriza por su manipulación de listas. La recursión usualmente es utilizada y resulta evidente en su aplicación. También se le conoce como programación declarativa (Prolog).
Funcional	Existe un conjunto de primitivas predefinidas por el lenguaje. Se reconocen un conjunto de formas funcionales disponibles. (Miranda y Lisp).
Orientación a Objetos	El sistema se descompone en entidades (objetos) jerarquizadas y clasificadas que corresponden a elementos específicos. El sistema se enfoca a los elementos del negocio y a su interacción, restándole importancia a los procedimientos y enfatizando el encapsulamiento.

III.3 Historia de los sistemas orientados a objetos

La historia de los SOO incluye las estructuras OO que han ido apareciendo en la ciencia, así como el pensamiento OO relacionado a la computación. Inicialmente la ciencia buscó identificar los distintos elementos que conviven en la resolución de un problema, obteniendo dos elementos; el primero es el proceso de búsqueda de la solución, que puede ser inductivo (de lo particular a lo general), deductivo (de lo general a lo particular) o eductivo (derivando nuevos elementos de las características previas o potenciales). Este proceso requiere en primer lugar de los requerimientos del problema (enunciado inicial), luego un procedimiento aplicado a la búsqueda de relaciones, experiencias y conexiones para llegar finalmente a una especificación (enunciado solución), que es el segundo elemento y puede ser representado por su "función", codificando sus características (basado en el proceso o función) o por su "forma", clasificando sus elementos (basado en clases).

La especificación de una solución por "forma", se reduce a reconocer las distintas entidades que componen el universo del problema, llamadas objetos, junto con sus características funcionales e informativas, para luego someterlas a una clasificación, todos los elementos que componen la clasificación conforman el universo donde nuestra solución tomará forma a partir de la interacción de todas y cada una de ellas.

La unión de objetos y software inició a principios de la década de los 60´s, comenzó a experimentarse en "inteligencia artificial" introduciendo la clasificación y la metáfora[3] de objeto a los sistemas de representación del conocimiento.

En 1967, Simula, un lenguaje desarrollado por Ole'Johan Dahl y Kristen Nygaard en Noruega, incluyó estos conceptos; Simula nunca se hizo popular entre la comunidad de sistemas, lo que hizo que, aunque era un lenguaje de propósito general, su uso se suscribiera solamente al modelado de problemas y simulación, y sus características de objetos eran conocidas sólo por unos cuantos investigadores.

La idea de objetos como elemento de construcción de software de propósito general se atribuye a Alan Kay, un graduado de la Universidad de Utah a finales de los 60´s. Kay concibió la idea de verdadera computadora personal, a la que llamó Dynabook. La Dynabook seria una computadora del tamaño de un cuaderno que podría ser utilizada de distintas formas, esta idea por sí misma era una demostración de que la metáfora de objeto podía extenderse más allá de las estructuras de programación.

En 1970, un grupo de la investigadores, también de la Universidad de Utah en EUA, Dan Ingalls y Adele Goldberg, unieron su trabajo al de Alan Kay en el Palo Alto Research Center (PARC) de Xerox, formulando los razonamientos básicos para el desarrollo de SmallTalk y su ambiente, permaneciendo como herramienta de investigación hasta mediados de los 80´s cuando las subsidiaria de Xerox, ParcPlace Systems convirtió a SmallTalk en un producto comercializable donde los objetos gráficos podían ser manipulados por un dispositivo apuntador o Ratón.

Un equipo de diseño de Apple Computer visitó PARC y sus instalaciones a inicios de los 80´s y salieron con nuevos conceptos radicales de diseño para sistemas operativos y métodos de interacción con el usuario. Apple inicialmente contrató a Dan Ingalls, y sus ideas que formaban la investigación en Xerox, se convirtieron en la base del sistema operativo de la Apple Macintosh, que popularizó inicialmente la interface gráfica de usuario.

El creciente interés en el lenguaje C, a inicios de los 80´s, llevó a programadores de lenguajes, como Bjarne Stroustrup de los laboratorios Bell AT&T, a comenzar a trabajar en una versión de C orientado a objetos, conocida como C++, o como a Brad Cox de StepStone Corp. y Objective C, al Departamento de Defensa de EUA con ADA orientado a objetos o, en otros casos desarrollar un nuevo ambiente basado totalmente en el concepto, como Bertrand Meyer con Eiffel, C. Shaffert con Trellis/Owl y Ungar-Smith con Self.

Se comenzaron a notar diferencias de fondo entre los lenguajes especializados, pues se denominó "BASADOS EN OBJETOS" a aquellos que solamente los utilizan, sin posibilidades de modificación o de creación, mientras que se llamó "ORIENTADOS A OBJETOS" a los que además de utilizarlos permiten la definición de las clases y de su herencia.

Desde finales de los 80's, los conceptos afines a la OO, fueron integrándose al diseño de aplicaciones como en el Diseño Asistido por Computadora (CAD), dedicado al manejo de texto y gráficos. Mientras tanto la potencia de hardware en las computadoras personales se elevó hasta un nivel en el que soportaba cabalmente la operación bajo los nuevos requerimientos gráficos de aplicaciones como aquellas.

Las herramientas OO para desarrollar rápidamente sistemas de manipulación gráfica comenzaron a surgir a inicios de los 90´s, así como la formalización de las distintas metodologías de OO (Booch-1990, Wirfs-Brock-

[3] Metáfora: Recurso literario mediante el cual se identifican, sin comparación expresa, dos objetos que guardan entre sí una relación o semejanza. En este contexto, hace referencia al modelo obtenido mediante la aplicación del enfoque de la OO.

1990, Coad y Yourdon-1990). Los sistemas operativos modernos han tomado las ideas de la OO, y actualmente (1996), se han desarrollado nuevos productos que en su mayoría presentan una interface gráfica de usuario con una fuerte relación entre objetos gráficos de pantalla y construcciones internas del sistema operativo, simplificando su uso, organizando su complejidad y facilitando su extensión en forma de nuevos objetos. Microsoft Windows NT, IBM OS/2 v2.1, PenPoint y NextStep son algunos ejemplos de ello.

Por otra parte, los sistemas de objetos distribuidos están conformando también las características del futuro cercano del proceso distribuido. Soportado bajo los modernos ambientes de red de área local (LAN) y de área amplia (WAN), la computación distribuida complementa los sistemas centralizados, llevando a cabo procesos específicos bajo la arquitectura Cliente-Servidor. La transición hacia los SOO consiste en utilizar los objetos como una herramienta para manejar la complejidad de los sistemas. Los objetos con su combinación natural de código y datos, y su estricta separación de interface, constituyen un útil instrumento para distribuir datos entre usuarios finales.

Empresas como IBM, con su System Object Model (SOM/DSOM) y el Component Object Model (COM) de Microsoft intentan resolver los problemas de un alto acoplamiento binario entre una aplicación y sus objetos subyacentes. Por su parte el Object Management Group, fundado en 1989 por 11 compañías entre las que se incluyen Digital, Hewlett Packard, Hyperdesk, NCR y SunSoft, adoptaron el estándar desarrollado por ObjectDesign, el Common Object Request Broker Architecture (CORBA) -liberado inicialmente en 1991 y actualmente desde 1992 en su versión 1.1- para regular la interoperabilidad entre objetos y aplicaciones siguiendo la arquitectura llamada Object Management Architecture (OMA).

La utilización de la Tecnología de Orientación a Objetos (TOO.- Conjunto de técnicas, métodos y herramientas OO) ha ido creciendo a ritmo acelerado, impulsada en gran medida por la habilidad de la industria del hardware de producir CPU's más pequeños, más poderosos y más baratos que soporten la sobrecarga inherente a su manejo y, sobre todo el de los ambientes integrales que, sin duda irán emergiendo para beneficio de un mejor y más eficiente manejo de la información.

III.4 Elementos de los sistemas orientados a objetos

El principal objetivo para cualquier sistema de software, será siempre el presentar un modelo de la realidad lo más cercano posible a nuestra percepción. Bajo este contexto resulta interesante el hecho de que el software sea un elemento intangible que se presenta como una poderosa herramienta para el modelado de la realidad. Para nosotros el universo está compuesto de "cosas" que nos rodean y que se manifiestan en cuanto se ponen al alcance de nuestros sentidos, a éstos "objetos", nuestra experiencia indica que se llaman de algún modo, que hacen actividades que los identifican y que tienen propiedades que los caracterizan.

Nuestra percepción entonces se inclina por la clasificación sistemática de todo, identificando la existencia de caracteres comunes entre perros, autos, edificios, personas, pájaros, piedras, etc. La clasificación de los elementos u objetos de un problema es una herramienta que utilizamos cotidianamente; agrupando a los objetos de acuerdo a una o más de sus características, reducimos la definición del problema a la interpretación de las necesidades de una clase de objeto en particular. El reflejar este esquema de pensamiento en una herramienta abstracta como lo es el software nos permite interactuar más "naturalmente" con los problemas, pues sus elementos se convierten en algo familiar y que por experiencia podemos conocer.

El potencial de la OO reside en la capacidad de modelado derivada de la funcionalidad y sinergia[4] de tres conceptos: encapsulamiento y/u ocultamiento de información (cuya unidad atómica es el objeto), abstracción por clasificación (que administra colecciones de objetos), y polimorfismo (logrado a través de herencia), que estructura colecciones de clases (ver Fig. III.2).

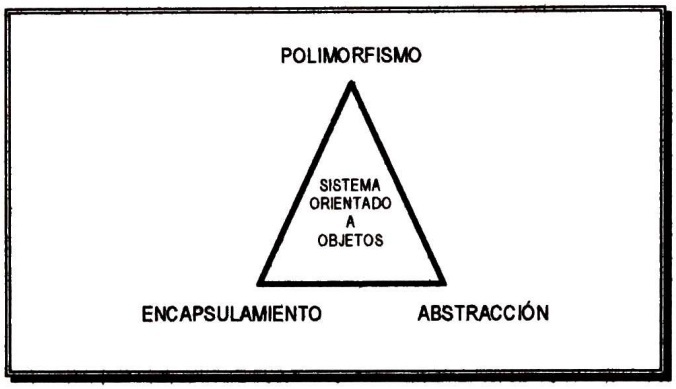


Fig. III.2.- Triángulo de la Orientación a Objetos.

Las actuales tendencias de los sistemas hacia los tipos de datos complejos (manejo de imágenes y datos) y en sistemas integrados donde se comunican e interrelacionan los sistemas, ha favorecido al modelo de objetos sobre los métodos convencionales de análisis y diseño.

III.4.1 Objeto, interface y métodos

Los SOO aprovechan nuestro conocimiento de la realidad para llevar a cabo su tarea de **modelado**, observan al problema como un conjunto de entidades que cumplen una función y que están subordinadas a su contexto, comunicándose entre sí. Un **objeto** es la analogía del software a un objeto del mundo real. Es una entidad **autocontenida**, es decir que contiene tanto código como los datos sobre los que opera. Cada objeto tiene la habilidad de recibir mensajes de otros objetos, almacenar la información de su estado actual y de efectuar un número limitado de operaciones basadas en los datos.

En el dominio del software un **objeto** es un elemento de información autónomo que contiene una estructura de datos **privada** y procesos llamados **operaciones** o **métodos**, que son los únicos que pueden transformar los valores de sus variables; las operaciones contienen las construcciones procedimentales que pueden ser llamadas por un **mensaje** (una petición al objeto para que realice una de sus operaciones), o ser utilizadas por el objeto mismo para realizar sus funciones (métodos locales). El **objeto** es también una unidad que presta sus **servicios** a todo objeto que los solicite, para ello cuenta, como ya se dijo, con **métodos** claramente especificados que determinan su comportamiento. Los **métodos** dedicados exclusivamente para uso propio del objeto son denominados como métodos pertenecientes a su **parte privada**, mientras que aquellos que se ponen a disposición de otros objetos se denominan como pertenecientes a la **parte pública** o como constituyentes de la **interface del objeto**.

La **interface** se convierte entonces en el único medio por el que la información del objeto podrá tener contacto con el exterior o con la interface de otros objetos y, ya que se presenta como una lista de métodos necesarios para establecer una comunicación interobjetos, dándole una fuerte protección a la parte privada, se le denomina también como el **protocolo de comunicación** del objeto.

[4] Sinergia: Producto notable resultado de la acción conjunta de dos o más entidades que por sí solas producen a su vez un efecto distinto

Los objetos se relacionan entre sí al enviar y recibir **"mensajes"** a través de su **interface**. Un **mensaje** podría definirse como un requerimiento para efectuar una operación. Un **objeto** responde a un mensaje al elegir el método apropiado, ejecutar su procedimiento y regresar el control a su "cliente". Sin embargo, dentro del contexto de los SOO, un mensaje puede ser:

- Informativo.- presenta el estado local resultante de una operación.
- Interrogativo.- Solicitando información sobre el estado actual.
- Imperativo.- Requiriendo la ejecución de una operación.

III.4.2 Encapsulamiento y ocultamiento de información

Si como se ha dicho, un objeto es una unidad atómica consistente en datos y métodos, ningún objeto podría considerarse definido sin hallarse descritos el qué es, y el cómo es. De ésta forma un objeto empaca la información acerca de lo que es (estructura) y de cómo es (funcionalidad). A lo que se denomina **encapsulamiento de la información.**

Los elementos internos de un objeto mantienen un **estado local**, compuesto por variables de uso permitido solamente a él mismo. Las operaciones de un objeto pueden ser **locales** (que sólo pueden ser llamadas por otra operación local o por una de interface de él mismo) y **no locales o de interface** (sólo pueden ser llamadas por medio de un mensaje originado por un objeto externo). Ambas, comparten el estado local del objeto, de forma que los cambios de estado realizados por una operación pueden ser apreciados por otra; sin embargo, cuando un objeto externo trate de accesar a la información del estado actual, sólo podrá hacerlo mediante los procesos no locales (o de **interface**), que proporcionarán la información solicitada o ejecutarán el procedimiento requerido; la **interface** es una lista que muestra los servicios que un objeto puede proporcionar a otro. De ésta forma, para el sistema, un objeto es una caja negra a la que sólo se tiene acceso por su **interface**, y tanto el código como las variables internas son invisibles. A esta propiedad se le denomina **ocultamiento de información**.

El **encapsulamiento** reúne los elementos de un objeto dentro de él mismo (autocontención) y no a lo largo del sistema, como en los SOP; lo que permite ver al objeto como un sistema cerrado (Fig. III.3) que se comunica con los demás por su interface-protocolo. Podemos asumir que el objeto "sabe" como hacer sus funciones, solamente indicándole qué método, de entre los que nos presenta, queremos que haga. Esto no significa que al definir el objeto no hace falta indicarle cómo hacer su trabajo, sino que necesita hacerse solamente una vez, evitando actualizaciones constantes y personalización de los métodos.

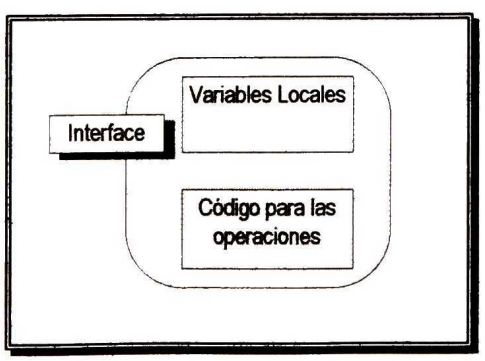


Fig. III.3.- Encapsulamiento.

Al encapsular, permitimos que el objeto sea un elemento funcional dentro de un esquema Cliente-Servidor, ocultando los datos locales a la vista del Cliente y respondiendo sólo a los requerimientos de su protocolo. También permite que el objeto sea un elemento altamente modularizado, que separa efectivamente al usuario del elemento de software de su autor.

Es importante mencionar que **el encapsulamiento no garantiza ocultamiento de información**, de tal suerte que algunas técnicas de programación recurren al encapsulamiento en la forma de modularización de procesos sin evitar accesos no planeados al conjunto de datos encapsulados. En la OO, esto no sucede, pues el objeto aparece como una entidad monolítica y atómica que maximiza el encapsulado.

A diferencia de las entidades de un SOP -que presentan la autonomía al nivel de los procedimientos y mantiene su interacción a través de variables no locales- las de un SOO sacrifican la autonomía de sus procesos y su reusabilidad para llevarlos a un nivel superior de organización, donde cada objeto puede ser visto como una caja negra y ser usado dondequiera que sus propiedades sean útiles. Sin tener que redefinir nuevos objetos para cada sistema, podemos usar nuevamente un objeto predefinido y a lo sumo, definir nuevos métodos; estaremos entonces reusando nuestro objeto.

La **reusabilidad** es la habilidad que tiene un elemento del sistema para ser aprovechado en la construcción de nuevos sistemas, entendiéndose como elemento del sistema a cualquier definición de objeto, jerarquía de clase, diseño o análisis propios de un problema. La reusabilidad es consecuencia directa del encapsulamiento y ocultamiento de información, es la manifestación de la autonomía de un elemento, que incide directamente en el ciclo de desarrollo del sistema, es el objetivo principal del análisis, el elemento del diseño y la herramienta en la programación.

III.4.3 Clasificación, clase y abstracción

Si agrupamos a los objetos de acuerdo a alguna característica que nos sea de interés y luego particularizamos las características que los diferencían entre sí, obtendremos una jerarquía progresiva de clasificación en la cual podremos encontrar un lugar donde nuestro objeto encaje, mostrándonos el panorama de su situación dentro de la clasificación. A este conjunto de características que definen exactamente a un objeto, diferenciándolo del resto se le denomina **clase**.

La clasificación siempre ha sido una herramienta útil en la resolución de problemas, pues nos permite observar sus elementos objetivamente, yendo desde una descripción en común que se aplica a muchas clases especializadas, hasta una descripción específica para una sola clase; es un medio descriptivo que define a cualquier ordenación de objetos, realizado para facilitar su análisis.

El clasificar objetos puede obedecer a alguno de los siguientes métodos:

- ✓ Diferenciar objetos y sus atributos, separando el concepto (p. ej., Árbol) de sus características (altura, grosor, etc.).
- ✓ Clasificar objetos y componentes (observando la distinción entre el árbol y sus ramas).
- ✓ Agrupar conjuntos y familias afines (conformar clases de árboles en función de su forma, origen, tipo, etc.).

Durante el análisis y diseño de un SOO, pueden reconocerse claramente la aplicación de algunos -y a veces de la totalidad- de los métodos aquí indicados. Si bien es cierto que el proceso de clasificación es una forma intuitiva de descomposición del problema, proporciona un esquema fácilmente adaptable y -sobre todo- descriptivo, sobre el que pueden bosquejarse los elementos de todo un sistema.

Al especificar tanto los atributos conceptuales como las características distintivas de cada elemento, explícita e independientemente de la realización de los objetos mismos, la clasificación introduce el concepto de **abstracción**, que permite la apreciación de los caracteres de un elemento, destacando aspectos de interés y discriminando los detalles más profundos o menos evidentes.

El pensamiento abstracto complementa al concreto (su antónimo funcional), en un proceso que utilizamos cotidianamente en nuestra comunicación social, de tal suerte que la "percepción" de un objeto dado (abstracción) se convierte en la "descripción" de un elemento (concretado). La **abstracción** es una propiedad del proceso de definición de una clasificación, e interviene en la conceptualización de cada uno de sus niveles.

Así, para clasificar un conjunto de objetos inicialmente "abstraemos" su estado actual a lo más obvio, estaremos entonces en el primer nivel de abstracción y a medida que avancemos en la descripción, nuestra percepción del objeto será menos abstracta, hasta el grado de puntualizar todas y cada una de sus características. Sin embargo, el nivel de abstracción depende de la necesidad de detalle en la información requerida.

Para un SOO la clasificación culmina en la definición de clases de objetos que componen al sistema. Cada **clase** es un conjunto de enunciados que justifican su separación del nivel anterior. La clase de un objeto determina:

- ✓ La denominación del tipo de objeto.
- ✓ La información asociada al objeto.
- ✓ Las funciones que pueden operar en el objeto.
- ✓ La clase de la que es derivado el objeto.

III.4.4 Instancia, clase base, clase derivada y tipo abstracto de datos

Una clase no puede efectuar operación alguna, puesto que no es una entidad concreta, en su lugar, la clase proporciona una plantilla que define las características que tendrán los objetos creados en base a ella; cuando esto ocurre se dice que el nuevo objeto creado es una **instancia** de la clase y ésta se denomina **clase base**.

El instanciamiento de una clase produce por resultado una entidad que comparte con su clase base el conjunto entero de su definición. Dentro de un sistema, cuando se instancía una clase para obtener un nuevo elemento, se dice que ocurre una definición dinámica (dynamic binding), puesto que ocurre siempre en tiempo de ejecución.

Como un ejemplo supongamos que se ha identificado dentro de nuestro universo del problema un elemento poseedor de una característica A, podemos comenzar definiendo una clase que, a falta de una denominación en particular, llamaremos Clase A. Si entrando en más detalle, reconocemos a un mismo nivel dos características *B* y *C*, podremos "derivar" de nuestra Clase A, un nivel inferior pero con mayor detalle, compuesto de dos clases derivadas (que llamaremos Clase derivada B, que reúne las características de la Clase Base A con las propias y la Clase derivada C, que hace lo propio con las características de la Clase Base A y las de ella misma). Ahora, si decidimos definir un objeto del tipo de la Clase B, instanciaremos la clase en la forma de un Elemento X, que presentará las características que ya definimos como propias de la Clase B, que se convierte en la clase base

del objeto X. Nuestro objeto recibe los atributos de su clase base, y se le conoce como un objeto de nombre X del tipo B (Fig. III.4).

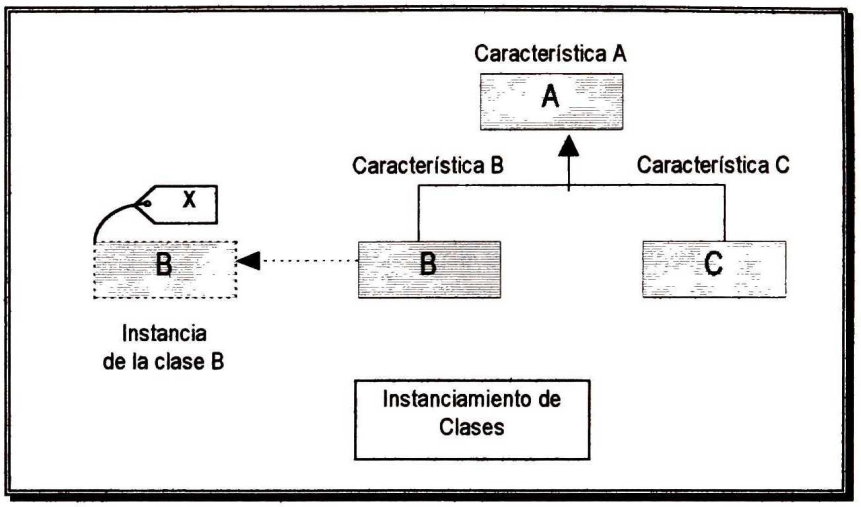


Fig. III.4.- Instanciamiento de Clase e Instancia de un Elemento de una

En términos de programación, una clase de objetos puede verse también como un tipo de objetos y ser usado como cualquiera otro tipo de datos del sistema; en esas condiciones la clase se relaciona fuertemente con el concepto del **tipo abstracto de datos** creado por el usuario o tipo de datos definido por el usuario. En un sistema procedural, si pudiésemos definir un tipo de datos distinto al entero, real, punto flotante o alfanumérico que tenemos disponibles en un lenguaje, determinando cada una de sus características y operaciones afines, estaríamos creando un tipo de datos a partir de la abstracción hecha al imaginar cómo sería. Dentro de un SOO existen las facilidades para realizar esta tarea.

III.4.5 Herencia

En Taxonomía, ciencia que clasifica a los seres vivos, se denomina "**Taxón**" al conjunto de especificaciones que configuran un sistema jerárquico, y que encuadra la condición que un ser mantiene dentro de su clasificación. Para un SOO, una clase es el producto del detallado sucesivo de un objeto, que destaca su importancia cuando se le enmarca bajo su jerarquía clasificatoria respectiva; para llegar a ella el **proceso de abstracción** ha pasado por la definición de varios niveles desde la clase más básica hasta su nivel jerarquico. En cada nivel el taxón de la clase acumula las características de los niveles superiores y las de la clase misma.

Se denomina, entonces, **herencia de clase** a las características provenientes de los niveles jerárquicos superiores que influyen directamente en el comportamiento de nuestro objeto, y **definición de clase** a las especificaciones distintivas de su clase. La herencia es el análogo a la herencia taxonómica, con la diferencia de que aquí la herencia puede recibirse de más de dos ancestros o inclusive de uno solo y no de dos.

La **herencia** es un nuevo concepto que, en contraparte al encapsulamiento y a la clasificación, solamente está disponible en los SOO. Es una propiedad de las clases, que complementa su funcionalidad al hacer uso de las características definidas para su clase base en la ejecución de algunas o todas sus operaciones. La

herencia emerge del hecho de que el objeto se compone de datos y sus funciones conservados dentro de un ámbito local o **contexto** bien definidos. Ya que la derivación de una clase se efectúa sobre el contexto de la clase base, la clase derivada presentará un nuevo conjunto de datos y funciones, extendiendo su ámbito más allá del de la clase base, que se convierte en un subconjunto de ésta. Es por este concepto que no se utiliza en este trabajo el término de subclase, pues supone un subconjunto de la clase inmediata superior, lo que resultaría contradictorio con el significado de herencia aquí expuesto. Sin embargo, autores como James Martin -en su libro Object Oriented Design- prefieren hacer una clara división de los conceptos asociados a la definición contextual. Para Martin, la generalización lleva al concepto de un supertipo de objeto, el cual incluye a uno o más tipos o contextos, y cuya definición es más general que aquellos a los que contiene, mientras que define a un subtipo de objeto como aquel cuyos miembros están contenidos en otro tipo de objeto, cuya definición es más especializada cada vez, es decir utiliza un supertipo de objeto para definir herencia y al subtipo para la especialización.

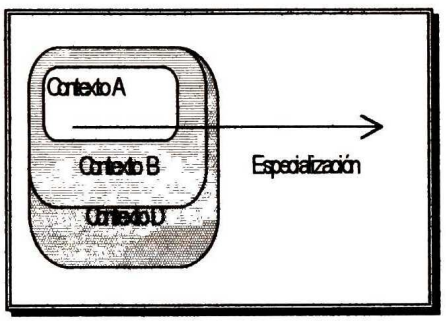


Fig. III.5.- Figura de "Lattice".

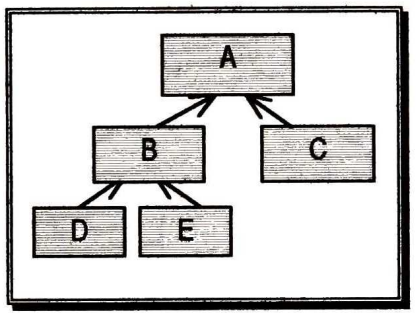


Fig. III.6.- Estructura de árbol

Como se muestra en la Fig. III.5 (llamada figura de "Lattice"), el contexto de la clase D se construye complementando la herencia obtenida de la clase base, que consta del contexto B que a su vez ha heredado el contexto A. Podríamos decir que la herencia de la clase D está compuesta por las características de clase de B y de A juntas y el Taxón se definiría como la suma de las propiedades de D y su herencia.

La herencia es estrictamente progresiva, es decir, sólo puede ir de un nivel de especialización inferior a uno superior o, en términos de abstracción, del menor al mayor (Fig. III.6). De ésta forma provee un modo de relacionar a las clases y de compartir la definición de cada nivel reflejando la especialización como atributos adicionales a los proporcionados por las clases. En el dominio del software la herencia también evita la duplicidad de código al compartir sus métodos con toda su "descendencia".

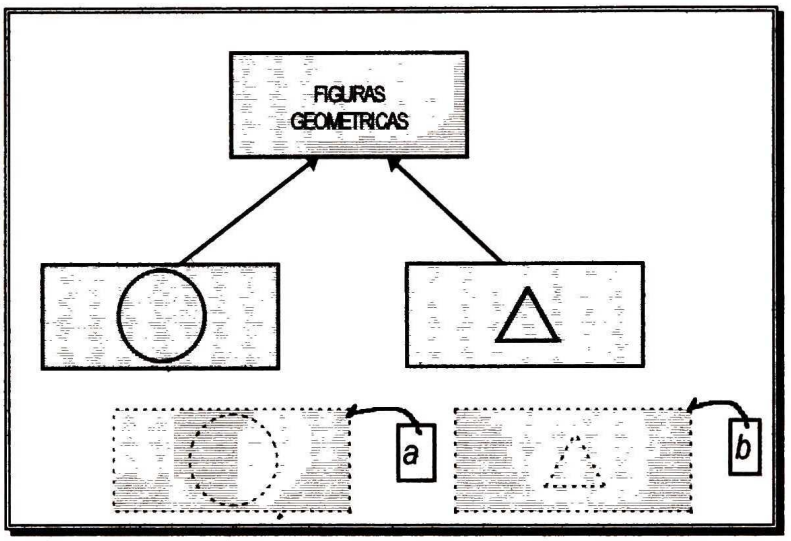
III.4.6 Polimorfismo

Del griego Poly (muchos) y morphos (forma): Múltiples Formas; en el ámbito de los SOO, puede definirse como la capacidad que tiene un **mensaje** de ser adoptado por la interface de más de un objeto y ser interpretado distintamente, según sea el tipo de objeto receptor. Es decir, es la propiedad del proceso de ejecución de una operación de ser sensitivo al tipo de objeto al que es aplicada.

Según Booch, "Es un concepto en el cual un nombre puede denotar a objetos de muy distinta clase relacionados por una clase base común. Así, cualquier objeto denotado por este nombre es capaz de responder al mismo conjunto de operaciones de distinto modo".

Mediante el **polimorfismo**, las abstracciones comparten elementos de su interface y presentan una homologación de operaciones que facilita al usuario el uso intensivo de los objetos obviando la discriminación de las características implícitas de la herencia. Es decir, se utiliza a la herencia en sustitución del análisis selectivo necesario para determinar si un objeto es candidato a recibir un mensaje.

El **polimorfismo** está fuertemente ligado con el concepto de construcción dinámica, ya que mediante ese mecanismo se permite la creación de objetos en tiempo de ejecución que reflejen el mismo conjunto de operaciones heredadas de su clase base, pero que también pueden implementar sus propias operaciones.



En la Fig. III.7, se presentan dos instancias, *a* y *b*, las cuales han heredado las características de la clase base "Figuras Geométricas", y comparten el conjunto de características de su herencia, pero cuando se les solicite una función que ambas deben ejecutar, por ejemplo la operación "dibujarse a sí mismo" llevará al objeto *b* a dibujar un triángulo, mientras que *a* dibujará una circunferencia.

Fig. III.7.- Instanciamiento de Clases y polimorfismo.

III.5 Ejemplo de modelado orientado a objetos

Ubiquemos un "antiguo" reloj Rólex de pulso y cuerda, dentro del contexto de nuestra realidad como un elemento perteneciente al conjunto de los Relojes (dispositivos que nos permiten conocer la evolución cuantitativa del tiempo). Nuestro objeto constará de distintas características:

- Aquellas que comparte con todos los objetos Reloj (mostrar la hora, resolución en segundos, es modificable o ajustable) y que cualquier reloj, por definición posee. (léase reloj atómico, de cuarzo, de pared, despertador, de mesa, de pulso, de arena, de agua, etc.).
- Aquellas compartidas con los objetos Reloj Analógico (manecillas, segundero, carátula, numerales, indicadores de minutos), como son los de pared, de péndulo, de cucú, el Big Ben, algunos despertadores y algunos de pulso.

- Las que son comunes a los objetos Relojes Analógicos Mecánicos (engranes, cuerda, tornillos, resortes, balancines, ejes, etc.), que agrupa a elementos como el Big Ben, relojes de pared y cuerda y, actualmente algunas piezas de relojería artesanal.
- Aquellas específicas de los Relojes Analógicos Mecánicos de Pulso (extensible, hebilla, pernos, caja, cristal de reloj, etc.) que comprenden a todos los relojes de las marcas conocidas que manejen o manejaron esa línea.
- Las que todos los relojes Rólex de ese tipo presentan (distintivo, logotipo, chapa de oro, mecanismo de diseño exclusivo).

Si continuamos nuestro proceso de identificación podremos concretar aún varios niveles más de especialización para nuestro objeto. Si decidimos, por el contrario omitir el resto de las posibilidades estaremos abstrayendo nuestro objeto hasta el nivel actual.

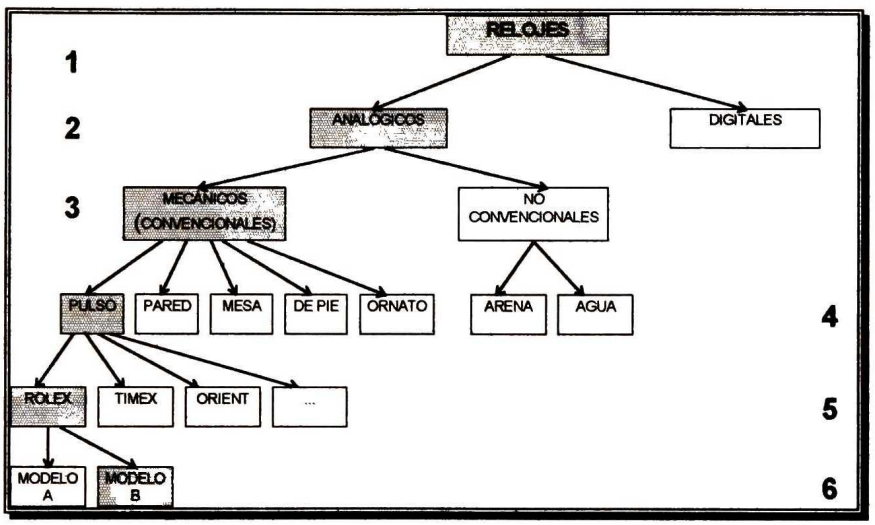


Fig. III.8.- Ejemplo de modelado orientado a objetos.

La abstracción nos permite omitir detalles de nuestra definición que no nos interesan. Así, si construimos una clasificación de las características que conocemos de los relojes, podemos armar una jerarquía que nos recuerda la taxonomía (ver Fig. III.8); donde, a menor especificación dentro de la clase corresponde una mayor abstracción. En nuestro caso, si decimos que tenemos simplemente un reloj estamos en el nivel máximo de abstracción para nuestro objeto, omitiendo los detalles más profundos, y si por otra parte, damos toda la información de nuestro objeto -propia de la máxima profundidad en nuestra clasificación- reflejamos el menor nivel de abstracción.

Nuestro objeto ha quedado ubicado dentro de una clasificación, es un objeto de clase Reloj y, más específicamente, Reloj Analógico Mecánico de Pulso Marca Rólex Modelo B (ver Fig. III.8 parte sombreada) y podemos decir que nuestro reloj pertenece a esta clase en particular o que es una instancia de ella. Una representación de nuestro objeto podría ser el diagrama de entidad mostrado en la Fig. III.9, empleado como un auxiliar en el diseño de SOO.

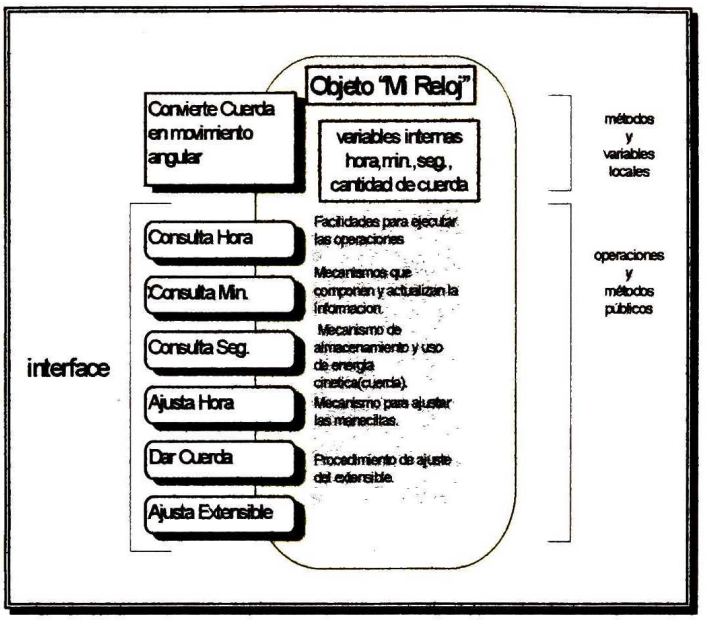


Fig. III.9.- Diagrama de Entidad del Objeto "Mi Reloj".

El objeto contiene variables que reflejan su estado dinámico que se mantiene encapsulado y oculto a nuestra vista, en este caso el tiempo, solamente disponible cada vez que lo consultamos -a través de alguna de las operaciones o métodos de su protocolo- que es cuando nuestro conocimiento de su estado se actualiza.

Las operaciones que podemos hacer con nuestro objeto son identificables fácilmente porque las conocemos, las hemos hecho intuitiva y hasta inconscientemente. Estas operaciones se agrupan en una interface que controla el acceso al estado actual, protegiendo los datos; mientras que el estado funciona como una memoria que afecta al estado futuro. Algunas de las operaciones son evidentes al tratar con la clase Reloj, esto se debe a que la clase especifica el conjunto de mensajes aceptados por los objetos que la componen, define una interface de encapsulamiento común a todos los relojes, sea el tipo que fuere. La clase también proporciona una plantilla, un molde para crear objetos con sus características predefinidas; cuando esto ocurre decimos que el nuevo objeto es una instancia creada en términos de esa clase base.

Cuando se creó nuestro objeto, todas las características definidas para los niveles que le antecedieron dentro de la jerarquía de clasificación se acumulan para formar su contexto, acompañándose con las características particulares, así nuestro objeto heredó la lista de operaciones válidas de la clase base y copió las de la clase propia.

El Polimorfismo en nuestro ejemplo queda manifiesto dentro de la jerarquía de clase del objeto, al comprobar que una operación, Consulta Minutos puede ser atendida por cada clase de reloj de distinta forma, un reloj de arena al mostrar un nivel de su contenido, uno de agua al mostrar la cantidad de agua desplazada, uno digital al cambiar los numerales y uno analógico por medio del movimiento del minutero sobre la carátula.

III.6 Elementos de beneficio en la OO

La adopción de la TOO permite observar ventajas inmediatas sobre los sistemas así desarrollados. La mayoría de esas ventajas no son un producto exclusivo de las TOO, sino que son el resultado de la materialización de tendencias de diseño y de características de la programación, que muchos desarrolladores encontraban convenientes o que facilitaban su trabajo en las tecnologías anteriores y que fueron conceptualizadas para interactuar estrechamente.

Reusabilidad.- Si se aplica adecuadamente, los mecanismos de la OO, tales como el encapsulamiento, herencia, polimorfismo y la definición dinámica, obvian las barreras técnicas para reusar el código de un programa, lo que adicionalmente propone la reutilización de los modelos de diseño o del marco de desarrollo y de los modelos de análisis para dominios relevantes de un problema. En el nivel de análisis y diseño, la reusabilidad puede tomar dos formas básicas: reuso de los componentes de un diseño previamente desarrollado y reuso de abstracciones de componentes de un programa preexistente. Con la reusabilidad, el análisis y diseño consumen más recursos que la programación y, como consecuencia principal, el presupuesto de desarrollo puede ser reducido tras un estudio de los elementos del sistema susceptibles de usar partes de un diseño previo. El mantenimiento del sistema por cambios extensivos se facilita al zonificar el impacto del cambio sobre unos cuantos objetos.

Modularidad.- La modularidad de los objetos separa como ya se ha mencionado, al usuario de una aplicación, de su autor, de forma en que no existe dependencia de un estilo de programación específico; favoreciendo al mantenimiento en la medida en que los módulos sean independientes.

Persistencia.- Otra característica es que la información que portan los objetos "persiste" a lo largo de la operación de un sistema haciendo accesible la "historia" del sistema a través de la información de sus objetos componentes. Cada objeto de un sistema conserva así su estado de operación aún después de ser utilizado, lo que le permite accesar a los datos anteriores en el desarrollo de sus actividades presentes para determinar su acción futura.

Estandarización.- La presencia de una estructura definida de inicio para la creación de los elementos del sistema hace que los objetos y clases presenten una apariencia homogénea que facilita la identificación de los elementos de las clases y la construcción de nuevos elementos basados en el conjunto preexistente.

Cabe hacer mención que existen aplicaciones que, sin ser totalmente OO, hacen referencia a conceptos OO. Para distinguirlas, se dice que un sistema es OO cuando su operación explota las características de creación dinámica, clasificación, herencia y polimorfismo, y por otra parte se dice que es basado en objetos, cuando su ambiente simula la presencia de objetos. Un SOO presentará verdaderamente las ventajas aquí mencionadas de una manera "natural", sin recurrir a manipulaciones especiales o simulación, mientras que un sistema basado en objetos intentará la convivencia simultánea de las ideas de la OO con las facilidades obtenidas en otras tecnologías, hecho que confunde fácilmente al usuario final; pero un sencillo análisis de su operación concluirá en la determinación exacta de su condición.

III.7 Análisis orientado al objeto (AOO)

La evolución de las metodologías modernas, comienza a finales de los 60´s con el concepto del ciclo de vida del desarrollo de sistemas o SDLC (por sus siglas en inglés). El incremento en la eficiencia del hardware y la

adopción de lenguajes de alto nivel, permitió construir sistemas más grandes y complicados y el SDLC trajo orden al proceso de desarrollo, que estaba excediendo los métodos del control de proyecto de entonces, descomponiendo al proceso en fases discretas de proyecto que entregan documentos formales a la siguiente fase. Una metodología de desarrollo de sistemas, combina herramienta y técnicas para guiar el proceso de desarrollo a gran escala.

El concepto del SDLC dio a los desarrolladores una medida de control, pero con poca ayuda para mejorar la productividad y calidad de análisis y diseño. Al inicio de los 70´s se desarrollaron las metodologías estructuradas para promover un análisis más efectivo y diseños más estables y mantenibles, éstas eran orientadas al proceso con un énfasis menor en el modelado de entidades y datos. Su orientación al proceso aparecía natural, dados los lenguajes de programación procedurales y las aplicaciones batch basadas en archivos. Los autores más representativos son: Yourdon y Constantine, De Marco, Ward y Mellor.

Si clasificamos las innovaciones tecnológicas como Incrementales, donde se introducen cambios relativamente menores a un proceso establecido, reforzando la tendencia actual y Radicales, que se basan en un diferente conjunto de principios estructurales y se desarrolla en un nuevo marco técnico y de solución de problemas, podemos decir que la etapa de análisis del paradigma de la OO representa un cambio radical sobre las metodologías orientadas al proceso del paradigma procedural, como la de De Marco, pero sólo representa un cambio incremental para las metodologías orientadas al dato como la ingeniería de la información de Martin.

Como en el análisis tradicional, la meta principal del AOO es la representación más completa y cercana al dominio del problema. También es meta del análisis el poner orden a nuestra percepción del mundo real para producir un modelado adecuado para el proceso de diseño. Debe simplificar el modelado de forma que se domine la complejidad al reformular el problema, removiendo el ruido y la sobreespecificación, encontrando inconsistencias, posponiendo la implementación, particionando el espacio del problema y documentándolo.

Sin embargo, como el mismo Yourdon señala, dentro de los autores que se han dedicado a estudiar los SOO, se dividen en un conjunto que se inclina por la visión sintesista, que ve a la OO como una simple acumulación de los principios de la ingeniería de software fácilmente adaptables a las metodologías existentes, mientras que otros, los revolucionarios, creen en la OO como un cambio radical que deja a la metodología y diseño convencionales en la obsolescencia.

Los participantes del sintesismo, como Wasserman, Pircher y Muller, toman la posición de que los métodos que utilizan son una elaboración de los métodos estructurados y consideran la terminología apta para el ambiente OO, de forma que declaran: *"El problema es que la OO ha sido ampliamente señalada como un enfoque revolucionario, una ruptura total con el pasado. Esto sería fascinante si fuera verdad, pero no lo es. Como muchos desarrollos en Ingeniería, el enfoque de la OO es el refinamiento de algunas de las mejores ideas de la ingeniería del software del pasado".*

Por su parte, los revolucionarios como Booch indican: *"No hay lugar a dudas de que el DOO es fundamentalmente diferente del enfoque tradicional del diseño estructurado, pues requiere una forma distinta de pensar acerca de la descomposición y produce arquitecturas de software separadas grandemente de la cultura del diseño estructurado".*

A lo que Yourdon añade: *"No dudamos que podríamos llegar al mismo resultado (como el producido por el análisis de Coad y Yourdon), usando diferentes métodos; pero ha sido también parte de nuestra experiencia*

que el proceso de pensamiento, de descubrimiento, y la comunicación entre el usuario y el analista es fundamentalmente diferente con un AOO al producido por un método de análisis estructurado".

Ciertamente, a simple vista se aprecia que la OO aprovecha algunas de las características de los sistemas estructurados, como por ejemplo la abstracción, el encapsulamiento, la modularidad y la clasificación, haciéndolas evolucionar junto con nuevos conceptos de apreciación de los problemas como lo son la herencia, el polimorfismo y, como apuntamos anteriormente, conformando un nuevo paradigma que no puede ser comparado si no es en el contexto de solución. De entre las metodologías, expondremos aquí tres de ellas que se consideran las mejor documentadas: Coad-Yourdon, Bailin y Shaeler-Mellor.

III.7.1 Especificación de requerimientos orientado a objetos de Bailin

En respuesta a la aparente incompatibilidad entre el análisis estructurado y el DOO, Bailin desarrolló su Especificación de Requerimientos Orientado a Objetos, donde la descomposición del sistema es efectuada usando la notación de diagrama de flujo. En el análisis estructurado esta notación implica agrupar las funciones sólo si son constituidas por pasos que llevan una función de mayor nivel. En cambio, Bailin propone agrupar las funciones sólo si operan en la misma abstracción de datos; en otras palabras, las funciones no pueden existir separadas de sus datos, pero deben subordinarse a una sola entidad. Esta noción es usada para promover el encapsulamiento de funciones y datos.

Existen dos aspectos importantes en el AOO de Bailin:

Hay una distinción entre entidades, poseedoras de un estado que puede persistir a lo largo de repetidos ciclos de ejecución, y las funciones, que existen solamente para transformar las entradas en salidas sin estados de persistencia. Las entidades pueden ser descompuestas en subentidades o funciones, mientras que las funciones sólo pueden ser divididas en subfunciones.

Bailin distingue dos clase de entidades: activas y pasivas. Las activas efectúan operaciones (sobre ellas mismas o en otras entidades) lo suficientemente importantes para ser consideradas en detalle durante la fase de análisis, mientras que las pasivas son de menor importancia y pueden ser tratadas como caja negra hasta la fase de diseño. Esta distinción es importante, pues las entidades activas, las pasivas y las funciones son modeladas de diferente forma durante el proceso de análisis.

Esta metodología consiste de un proceso de siete pasos:

1) Identificar las entidades del dominio del problema. Dibujar un diagrama de flujo para designar objetos que aparecen en nombre de proceso como entidades candidatas.

2) Distinguir las entidades pasivas de las activas. Distinguir entre las entidades con operaciones significativas en los términos de la descripción de los requerimientos del sistema (activas), contra las que pueden diferir su especificación hasta el diseño (pasivas), y construir un diagrama de entidad-relación.

3) Establecer flujos de datos entre entidades activas. Construir el diagrama de entidad-flujo (EDFD) de nivel cero, designando cada entidad activa como un nodo de proceso y cada entidad pasiva como un flujo de datos o información almacenada.

4) Descomponer las entidades (o funciones) en subentidades y/o funciones. Iterativamente, junto con los pasos 5 y 6, se determina si cada entidad de nivel 0 puede estar o no compuesta por entidades de nivel inferior, considerando que cada entidad hace y designa su propias operaciones como funciones. Para cada una de las subentidades identificadas, crear un nuevo EDFD y continuar el proceso de descomposición.

5) Revisar por nuevas entidades. A cada fase de la descomposición, considérese que cuando se implica a nuevas entidades, nuevas funciones se introducen y se le añaden, modificando el EDFD.

6) Agrupar funciones bajo nuevas entidades. Identificar todas las funciones ejecutadas por o en nuevas entidades.

7) Asignar a las entidades un dominio propio. Asignar a cada entidad un dominio de aplicación y crear un conjunto de DER's (Diagrama de Entidad Relación), uno para cada dominio.

El resultado del método de Bailin es un diagrama de entidad-relación, junto con una jerarquía de diagramas entidad-flujo de datos. La metodología conforma los principios elementales de la OO, a pesar de que no se utiliza terminología de la OO. Los diagramas de entidad-relación capturan la clasificación de objetos y su oportunidades de herencia, mientras que las funciones coinciden con el concepto de encapsulamiento.

III.7.2 Análisis orientado al objeto de Coad-Yourdon

El AOO es visto por los autores como "la construcción a partir de los mejores conceptos del modelado de la información, Lenguajes Orientados a Objetos (LOO) y sistemas basados en conocimientos".

El modelo es construido a partir de un proceso de cinco pasos:

1) Definir Objetos y Clases. Buscar estructuras, otros sistemas, dispositivos, eventos, procedimientos operacionales y unidades organizacionales. La estrategia de búsqueda puede hacerse de manera informal, dentro de un enunciado que define la solución del problema, presentado en lenguaje natural y a un nivel consistente de detalle, o como un proceso intuitivo de identificación de entidades, teniendo en cuenta más que a la solución, a los elementos del problema.

2) Definir estructuras. Buscar relaciones entre clases y representarlas como estructuras generales-a-específicas (donde se reconoce a los objetos susceptibles de especificarse aún más o si un objeto identificado es una especificación de otro) o como estructuras completas-a-elementales (identificando si los objetos requieren de otros para su definición completa o si a su vez son necesarios para lograr una definición).

3) Definir áreas de sujetos. Examinar los objetos de nivel superior con jerarquías Completa-a-elemental y marcarlas como candidatas a ser área de sujetos, refinándose al minimizar interdependencia entre los sujetos.

4) Definir atributos. Identificar las características atómicas de los objetos como atributos. Buscar también relaciones asociativas entre objetos y determinar la cardinalidad de esas relaciones.

5) Definir servicios. Para cada clase y objeto, identificar todos los servicios que efectúa, dentro de su ámbito o en beneficio de otras clases y objetos.

Las herramientas primarias para el AOO de Coad-Yourdon son los diagramas de objeto y las cartas de servicio. Los diagramas de clase y objeto tienen cinco niveles que se construyen incrementalmente durante cada una de las cinco fases descritas. Las cartas de servicio son más parecidas a un diagrama de flujo tradicional, se usan durante la última fase, para representar la lógica interna de los servicios.

Este método soporta explícitamente cada uno de los principios esenciales de la OO. Los diagramas de clase y objeto proveen de la clasificación de objetos y de las posibles relaciones de herencia; el encapsulamiento es modelado por el concepto de servicio exclusivo y la conectividad de los mensajes. El método hace énfasis en el modelado de la información.

III.7.3 Análisis orientado al objeto de Shaler-Mellor

Shaler y Mellor desarrollaron su metodología de análisis en el curso de varios años de asesoría en modelado de la información. Proponen una descripción completa del dominio del problema mediante tres formas de ver al sistema: información interrelacionada, modelo del estado y modelo del proceso, aplicadas durante seis etapas:

1) Desarrollar un modelo de la información. Consistente de objetos, atributos, relaciones y construcciones de objetos múltiples (basados en relaciones asociativas, y en el análisis de los enunciados "es un" y "es parte de"). Para los autores, un objeto es un equivalente a la noción convencional de entidad, es decir, una persona, un lugar o un evento que tenga lugar en el mundo real.

2) Definir el ciclo de vida de los objetos. Efectuar el análisis del ciclo de vida de cada objeto (desde su creación hasta su destrucción) y formalizar el ciclo de vida en colecciones de estados (algunas condiciones predefinidas de un objeto), eventos (señales que causan la transición de un estado a otro), reglas de transición (especifican transiciones disponibles entre estados) y acciones (actividades u operaciones que deben ser efectuadas por un objeto para llegar a un estado). También se definen los *timers*, mecanismos usados por las acciones para generar un evento futuro. La herramienta principal en esta etapa es el modelo de estado.

3) Definir la dinámica de las relaciones. Se desarrolla un modelo de estado de aquellas relaciones entre objetos que ocurren dinámicamente. Para cada relación dinámica se define un objeto asociado en el modelo de la información. Se definen modelos especiales de asignación para relaciones en que puede haber contención de instancias de objeto por los recursos de otra instancia de objeto.

4) Definir la dinámica del sistema. Aquí se produce un modelo del tiempo y control al nivel del sistema. Un modelo de comunicación de objeto (MCO) muestra el control síncrono, mientras que un modelo de acceso de objetos muestra el control asíncrono. Los autores describen un procedimiento para el trazado de flujos de control a alto y bajo nivel.

5) Desarrollo de modelos de proceso. Para cada acción, un diagrama de flujo de acciones se crea para mostrar todos los procesos para esa acción, y el flujo de datos entre el proceso y su almacenamiento. Se utiliza un diagrama de flujo de datos convencional (De Marco). Se definen cuatro tipos de proceso: accesantes, generadores de eventos, transformaciones y pruebas, y se proveen las guías para ubicar las acciones dentro de estos cuatro tipos.

6) Definición de dominios y subsistemas. Para problemáticas mayores, es útil descomponer al sujeto en dominios conceptualmente distintos, de los que se identifican cuatro: aplicación, servicio, arquitectura e implantación. Adicionalmente, algunas veces es útil descomponer el dominio de aplicación en múltiples subsistemas.

Shaler y Mellor otorgan un soporte más implícito que explícito para los tres principios esenciales de la OO. Los objetos y las relaciones contenidas en el diagrama de estructura de información, aunque no idénticas a los conceptos de clasificación y herencia, pueden fácilmente ser utilizadas durante el diseño.

III.8 Diseño orientado al objeto (DOO)

Diseño es el proceso de interpretar los requerimientos del sistema definidos durante la etapa del análisis en una representación abstracta de una conceptualización basada en el sistema, tomando en cuenta costos, calidad y eficiencia. El DOO transforma las clases del análisis en un modelo computarizado perteneciente al espacio de solución del problema que obedezca a una descripción de clase.

Las metodologías para el diseño de SOO presentan diferencias substanciales contra las metodologías tradicionales, siendo las principales:

- Diseño de datos que soporta la definición de clases, herencia y métodos.
- Utiliza descomposición modular OO; mientras que en el diseño tradicional, módulo puede ser un programa, subrutina o función que sólo contiene código procedimental, en el DOO, el objeto es la unidad primaria de modularidad.
- Se utiliza la descomposición OO; que resulta en una colección de métodos encapsulados dentro de los objetos, en lugar de la descomposición funcional que resulta en módulos de programa.

Dentro de la clasificación Incremental-Radical, el DOO tendría que colocarse como un cambio Radical para los paradigmas orientados al proceso y orientado al dato, al proveer una forma distinta de contemplar, dividir, estructurar y representar al problema. Cabe también anotar que el DOO exige el uso de herramientas adecuadas para la representación detallada de las definiciones de clases y herencias, de las relaciones entre clases y objetos, y de las conexiones entre mensajes y operaciones de objeto. Existen decisiones implícitas en el proceso del DOO que determinan el estilo de diseño y que pueden afectar grandemente al resultado, como es, por ejemplo, el balance correcto entre el máximo encapsulamiento (al enfatizar las responsabilidades de los objetos), la máxima herencia (enfatizando las similitudes entre clases), o la máxima reusabilidad (diseñando clases que sean independientes del contexto en el cual son usadas).

Actualmente la definición de métodos de DOO, en oposición a los del diseño estructurado, se considera un aspecto inmaduro de la TOO, encontrándose en una etapa de definición y refinamiento. A continuación se expondrán brevemente las características de tres de los métodos de diseño difundidos hasta 1994.

III.8.1 Diseño estructurado orientado al objeto (DEOO) de Wasserman

El DEOO fue desarrollado por Wasserman, Pircher y Muller, como una metodología que provee una notación detallada para describir el diseño estructural o de alto nivel que identifica los módulos individuales, pero sin

representar su detalle interno. Para Wasserman, la meta principal del DEOO, es proporcionar una notación de diseño estándar, capaz de soportar a cualquier diseño de software, tanto OO como convencional; para lo que ofrece una notación híbrida que incorpora conceptos de trabajos previos de diseño como las cartas de estructura, la notación de Booch para paquetes y tareas, jerarquía y herencia, y el concepto de monitores para la programación concurrente.

Wasserman propone el uso de símbolos y notaciones típicos del diseño estructurado, como las cartas de estructura, incluyendo concepto de módulos, datos parámetro y parámetros de control y añade nuevas notaciones OO basadas en el esquema de Booch para los elementos de diseño en ADA equivalentes a objetos, clases, herencia, métodos, instanciamiento y concurrencia (Cartas de Estructura Orientada a Objetos).

III.8.2 Diseño orientado a objetos de Booch

En la década de los 80's, Booch incursionó en el campo del DOO, al proponer una metodología basada en el lenguaje ADA, que posteriormente fue significativamente expandida y generalizada. Booch plantea su metodología más como una alternativa al diseño estructurado que como una extensión. En su planteamiento, se describen un conjunto de técnicas y herramientas, desde una lista informal hasta diagramas y plantillas formales; sin prescribir un orden fijo en las fases de su propuesta, recomienda seguir un trabajo iterativo e incremental para la construcción de los formalismos con técnicas informales. Booch delinea los cuatro pasos que, según él deben seguirse para realizar un DOO.

1) Identificar Clases y Objetos. Identificar abstracciones clave en el espacio del problema y etiquetarlas como candidatas a Clase/Objeto.

2) Identificar la semántica de clases y objetos. Establecer el significado de las clases y objetos identificados en el paso anterior usando varias técnicas, que incluye la creación de "scripts" que definen el ciclo de vida de cada objeto, desde su creación hasta su destrucción.

3) Identificar relaciones entre clases y objetos. Establecer interacciones de clase y objetos, tales como patrones de herencia entre clases y patrones de cooperación entre objetos. Este paso también captura algunos detalles de visibilidad entre clases y objetos.

4) Crear Clases y Objetos. Construir vistas detalladas internas de clases y objetos, incluyendo definiciones de sus servicios. Alojar las entidades (objetos y clases) de acuerdo al lenguaje usado y alojar programas para los procesadores (en los ambientes que soporten multiproceso).

Las herramientas primarias usadas durante el DOO son:

- ✓ Diagramas y plantillas de clase (muestran definiciones de clase y relaciones de herencia).
- ✓ Diagramas de Objeto y de tiempos (que enfatizan definiciones de mensaje, visibilidad y flujos de control).
- ✓ Diagramas de transición de estado (para modelar la transición y el estado de los objetos).
- ✓ Plantillas de operación (capturan la definición de servicios).

- ✓ Diagramas y Plantillas de módulo (para capturar decisiones de diseño físico acerca de la asignación de los módulos en objetos y clases).
- ✓ Diagramas y Plantillas de proceso (para asignar módulos a procesos en situaciones donde se usa una configuración multiproceso).

El DOO de Booch ha provisto a la mayoría de las metodología modernas de las herramientas de modelado básicas de la OO.

III.8.3 Diseño de manejo de responsabilidades (DMR) de Wirfs-Brock

Wirfs-Brock, Wilkerson y Wiener desarrollaron el método DMR como producto de varios años de experiencia en el desarrollo interno de Software corporativo. DMR se basa en el modelo computacional de Cliente-Servidor, en el cual el sistema es visto como una colección de Servidores que acaparan responsabilidades privadas y proveen servicios a los Clientes, de acuerdo a contratos que definen la naturaleza y ámbito de las interacciones Cliente-Servidor válidas.

En el contexto de la terminología OO, clientes y servidores son diferentes tipos de objeto, mientras que los servicios y responsabilidades son los métodos. Los contratos y las colaboraciones son metáforas para la idea de que para conservar el encapsulamiento, algunos objetos tienen la capacidad de efectuar ciertas tareas (p. ej., modificar el valor de sus variables internas) para el beneficio de otros objetos y de que algunos objetos necesitan los servicios de otros para alcanzar un resultado deseado.

Esta metodología se dice de "Manejo de Responsabilidades" porque se avoca, durante el diseño, a los contratos entre los objetos clientes y los servidores. Dichos contratos indican expresamente de qué acciones es responsable el objeto y qué datos está obligado a compartir. Los autores contrastan su método contra lo que llamaron Diseño de Manejo de Datos (DMD), que enfatiza el enfoque en las clases y herencia por medio del diseño de las estructuras de datos internas para los objetos y la construcción de relaciones de herencia basados en sus atributos comunes. Por el contrario, el enfoque del DMR intenta maximizar el nivel de encapsulamiento en el diseño resultante, poniendo mayor atención a las interacciones de objeto y su encapsulamiento.

Como Booch, Wirfs-Brock recomienda un desarrollo incremental-iterativo de su método, sin orden fijo. El DMR presenta seis pasos que se aplican en dos etapas: exploración, donde se busca por clases candidatas, responsabilidades, colaboraciones y construcción de jerarquías, donde se definen subsistemas y protocolos.

1) Encontrar Clases. Extraer sustantivos de las frases que componen la especificación de requerimientos y construir una lista de clases candidatas al observar los sustantivos que se refieran a objetos físicos, entidades conceptuales, categorías de objetos o interfaces externas. También se identifican los atributos de objetos y superclases candidatas.

2) Encontrar responsabilidades y asignarlas a las clases. Considerar el propósito de cada clase y examinar la especificación de las frases de acción para encontrar responsabilidades candidatas. Asignar responsabilidades a las clases de tal forma que la inteligencia del sistema se distribuya, las propiedades residirán en la información relacionada y las responsabilidades que se comparten entre clases afines.

3) Búsqueda de colaboraciones. Examinar responsabilidades asociadas con cada clase y considerar qué otras clases son necesarias y que su colaboración complemente la responsabilidad.

4) Definir jerarquías. Construir jerarquías de clase para las relaciones hereditarias de tipos comunes y para las clases que presentan responsabilidades parecidas.

5) Definir subsistemas. Dibujar un diagrama de colaboración para el sistema completo; observando aquellas colaboraciones frecuentes y complejas para considerarlas como candidatas a subsistemas. Las clases dentro de un subsistema deben soportar un conjunto de responsabilidades pequeño y fuertemente cohesionado, además de ser fuertemente interdependiente.

6) Definir protocolos. Desarrollar un diseño detallado, escribiendo las especificaciones para las clases, subsistemas y protocolos.

El DMR presenta un contraste significativo contra las metodologías anteriores, puesto que en su afán de enfocarse a las responsabilidades y características dinámicas de los objetos, intenta construir una simulación cercana al funcionamiento del sistema; en lugar de establecer de entrada una jerarquía de clase, haciendo énfasis en las relaciones estáticas de sus contrapartes del diseño.

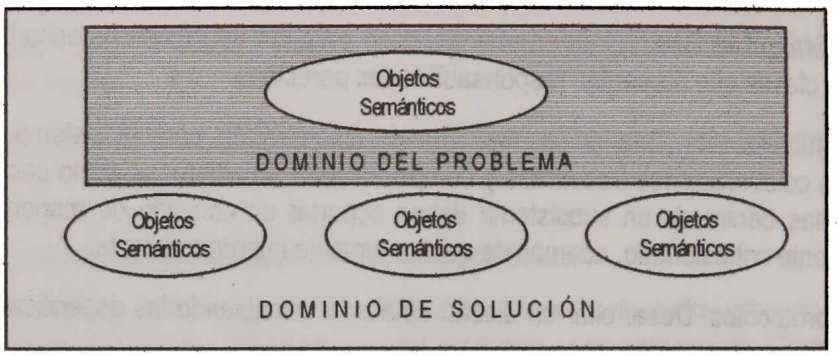
III.9 Comentarios a los métodos orientados a objetos

El DOO y el AOO conviven en la práctica de una forma en la que parece indistinguible una frontera real a partir de la que se pueda hacer una separación de los dos conceptos; y es que durante el proceso de desarrollo, se pasa constantemente de un lado a otro de la línea, como se aprecia al observar que la concepción de "objeto" es más bien dinámica a lo largo del ciclo de creación del sistema. Podemos notar cómo en el análisis se denomina objeto al elemento de modelado y que al pasar al diseño se le trata como un elemento de clasificación o clase, para luego, durante la programación y ejecución de los programas, regresar al concepto de objeto como una instancia.

En un esfuerzo por categorizar nuestra labor de desarrollo, podríamos distinguir nuestra estadía en un lado y en otro, al identificar actividades típicas, es decir, si queremos saber cómo se llama lo que estamos haciendo, debemos detenernos un poco y tratar de explicar qué es exactamente lo que hacemos y de qué manera. Para comprender este marco de referencia, observemos que:

AOO: Es la etapa donde se modela el dominio del problema, identificando y especificando un conjunto de objetos semánticos que interactúan de acuerdo a los requerimientos del sistema. Durante el análisis se especifican (abstraen) los objetos que forman parte de la descripción del problema y sólo tienen significado allí, perteneciendo al dominio del problema. La reespecificación de responsabilidades y los enunciados semánticos que detallan a un objeto, determinan la aplicación del análisis.

DOO: Modela el dominio de solución del problema, especificando la intercomunicación entre objetos, traduce los objetos semánticos que intervendrán con el usuario, en clases semánticas de interface; los que controlan directamente la aplicación o la secuencia de las funciones en clases de Aplicación y los objetos semánticos identificados como componentes independientes como clases de utilería. Durante esta etapa se organizan las clases desde el punto de vista dinámico y se estructura la complejidad.


Objetos
Semánticos
DOMINIO DEL PROBLEMA
Objetos
Semánticos
Objetos
Semánticos
Objetos
Semánticos
DOMINIO DE SOLUCIÓN

CAPÍTULO IV

DESARROLLO DE LA APLICACIÓN

CAPÍTULO IV

DESARROLLO DE LA APLICACIÓN

Dentro del panorama de la OO, al usar cualquiera de las metodologías antes mencionadas, podemos sentir cierto grado de libertad en varios de los puntos del análisis o del diseño que ponen de manifiesto una realidad dentro de las TOO: aún no existe un consenso generalizado sobre cuál es la técnica o combinación de ellas que represente significativamente al enfoque de objetos. Y es que ciertamente, la TOO es una disciplina nueva que demanda aún mucho tiempo de investigación y estandarización, sin embargo, existen autores que de manera entusiasta sugieren un camino a seguir, una lista de proposiciones que a juicio de ellos mismos es útil en ciertos tipos de problemas pero que de ninguna manera son definitivos.

Por todo lo anterior, el desarrollo de un sistema como el presente enfrenta al trabajo de decidir qué métodos de análisis y diseño usar y cómo deben interactuar entre sí. Tomando en cuenta que cada una de las técnicas propuestas ha sido formulada de acuerdo a un contexto de problemas específicos a los que sus autores se han enfrentado, puede considerarse adecuada la construcción de un método a la medida de la complejidad de nuestro problema, tomando elementos técnicos que ya han sido probados.

En 1990 en su libro "Ingeniería del Software", Pressman propone un proceso de DOO que reúne en sí mismo varias técnicas en un afán por conducir al desarrollo de sistemas por un camino sencillo y práctico de lo que es la TOO. El método propuesto por Pressman como DOO, se presenta de forma concisa en los siguientes pasos:

1. **Definir el problema.**
2. **Desarrollar una estrategia informal para la realización del software del dominio del problema en el mundo real.**
3. **Formalizar la estrategia usando los siguientes subpasos:**

    A. **Identificar objetos.**
    B. **Identificar las operaciones y atributos que pueden aplicarse a los objetos.**
    C. **Establecer interfaces para mostrar las relaciones entre los objetos y las operaciones.**
    D. **Decidir los aspectos del diseño detallado que harán una descripción de los objetos.**

4. **Repetir los pasos 2, 3 y 4 recursivamente hasta que se cree un diseño completo.**

Si sólo obedeciéramos al conjunto de pasos que Pressman expone, pasaríamos por alto el hecho de que se han reunido ahí al análisis y al diseño en una forma que, como ya hemos mencionado en el Capítulo de "Orientación a objetos como metodología de diseño", se vuelven indistinguibles. Sin embargo, para delimitar cada una de las etapas, observemos como los pasos 1 y 2 corresponden al método propuesto por Yourdon para el análisis; y notemos además cómo la técnica de Yourdon continúa su desarrollo y se sobrepone implícitamente a algunas de las etapas que componen al paso 3.

El DOO de Booch está claramente circunscrito en el propio paso 3, donde sus cuatro proposiciones son enunciadas exactamente como una serie de subpasos. En el paso 4, vale la pena hacer alto en la propuesta *recursiva* de los pasos precedentes. La recursividad en el método debe entenderse como la aplicación de las proposiciones 2 y 3 al producto de esos mismos pasos, de manera que, si partimos de un análisis inicial, acatando los pasos 1, 2 y 3 tendremos como resultado un conjunto de entidades

identificables, que en la primera recursión serán una a una sometidas a los pasos 2 y 3, que a su vez nos proporcionarán más elementos para el análisis y así sucesivamente hasta agotar los elementos obtenidos o llegar al grado de detalle deseado.

Por lo anterior se consideró que la adopción de este método implica ventajas que harán decididamente de éste nuestro método de trabajo. Entre las ventajas podemos mencionar las siguientes:

- Fácil de interpretar.
- Sencillo en su uso.
- Emplea herramientas estándar a la mayoría de los métodos.
- Hace énfasis en la identificación intuitiva de objetos, sin descartar conceptos útiles de las metodologías estructuradas.
- Flexible en su implantación y conciso en su definición.

De hecho, un método basado en la metodología de Booch, necesariamente cumple con la mayoría de los preceptos de OO -recuerde que fue uno de los pioneros en el área- y si además se cuenta con el respaldo de un metodologista como lo es Yourdon, no hacemos más que obedecer a una elección obvia. Por otra parte, puede no parecer tan claro que este método es adecuado para sistemas de tamaño moderado, puesto que el control de las características pertenecientes a los elementos tiende a dificultarse con el crecimiento o la expansión, haciéndose necesario un método más riguroso.

IV.1 Definición del problema

Puesto que no puede existir análisis sin problema a analizar, es requisito indispensable la existencia de uno que debe estar, dicho sea de paso, bien definido. Al hacerlo, estamos asentando el **requerimiento** que describe a algunos de los elementos dentro del universo del problema, el conocimiento intuitivo del ambiente del caso planteado completará el cuadro de manera que sea comprensible y el entendimiento del problema sea mejor. La generación de una solución al problema depende en gran medida de la exactitud de la **definición del problema**, que nos permitirá identificar al **dominio de solución** en base al requerimiento y el reconocimiento de los elementos de este dominio de solución que se relacionarán entre sí en una **definición de la solución**. Para fines prácticos debemos tomar en cuenta que la definición de la solución en software debe establecerse de modo sencillo y gramaticalmente correcto, para unificar el problema a un alto nivel de abstracción.

La primera tarea para la definición del problema en DOO es escribir una declaración en una sentencia del problema. Esto, por ejemplo, lo podemos hacer con la ayuda de un Diagrama a Bloques del sistema o de un Diagrama de Flujo de Datos (DFD) a un primer nivel.

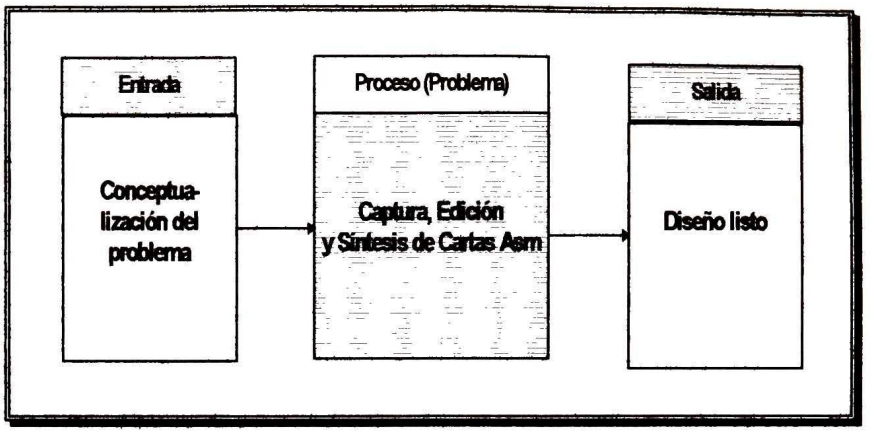


Fig. IV.1.- Diagrama a bloques del problema.

La Fig. IV.1 nos ayuda a obtener el siguiente enunciado:

Paso 1. Definir el problema: **"LA AUSENCIA DE UN SISTEMA DE SOFTWARE AUXILIAR EN EL PROCESO DEL DESARROLLO DEL MÉTODO DE DISEÑO DE CONTROLADORES DIGITALES DENOMINADO CARTAS ASM".**

IV.2 Una estrategia informal

El siguiente paso es escribir una estrategia informal para la solución del problema establecido en la definición. La estrategia es una descripción en lenguaje natural de la solución del problema que hay que resolver mediante software, representado a un nivel consistente de detalle. Como regla general, la estrategia informal tiene las siguientes características:

1. Se escribe como un párrafo sencillo y claro.
2. Se escribe al mismo nivel de abstracción, esto es, el nivel de detalle debe ser consistente a lo largo de todo el párrafo.
3. Está enfocado sobre lo que debe hacerse para resolver el problema, en vez de en los detalles procedimentales de cómo se obtiene la solución, ni de cómo se descomponen los bloques o cómo se procesan y codifican los elementos de control.
4. El párrafo debe intentar repetir los mismos términos para describir la información y el procesamiento en vez de usar sinónimos.
5. Se describe la estrategia informal usando una terminología a nivel de sistema, en vez de un vocabulario típico de software.

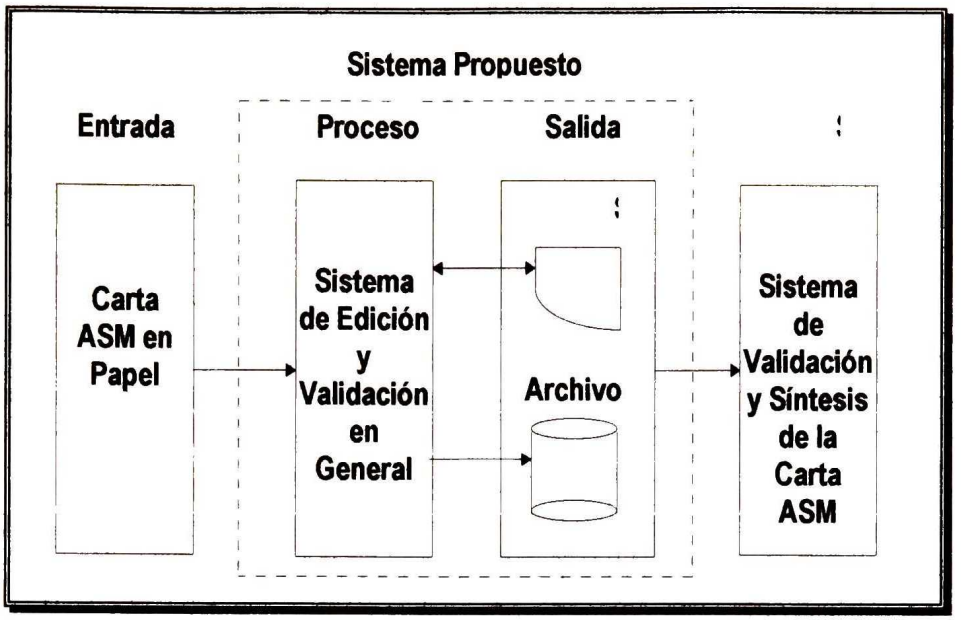


Fig. IV.2.- Detallado del diagrama a bloques del problema.

De acuerdo a la estrategia informal y con la ayuda de la Fig. IV.2, la solución al enunciado del problema quedaría representada de la siguiente forma:

Paso 2: **"DISEÑO Y CONSTRUCCIÓN DE UN ENTORNO GRÁFICO DE SOFTWARE QUE PERMITA LA EDICIÓN DE UNA CARTA ASM CONCEPTUALIZADA POR EL USUARIO".**

La función primordial del sistema a implementar, consiste en el procesamiento de la información derivada de una Carta ASM diseñada previamente en papel (ENTRADA), a fin de generar unos archivos que describan correctamente la estructura de dicha carta (SISTEMA PROPUESTO: PROCESO Y SALIDA), para que otro sistema (SALIDA: el complemento del proyecto propuesto) obtenga la solución al problema por medio de una técnica aplicable a los controladores digitales.

Es importante mencionar que pueden existir dos clases de enunciados de acuerdo a su intención, por una parte los que explican al objeto mencionando todo aquello que hace, y por otra parte los que especifican un objeto al decir de qué se compone y por qué. No es extraño encontrarse con algún enunciado específico-explícito o también enunciados complejos compuestos por más de una frase.

IV. 3 Formalización de la estrategia

Es en este punto donde comienza realmente el DOO. El objetivo es aislar e identificar los objetos, operaciones y sus interrelaciones, de forma que pueda derivarse un diseño.

IV.3.1 Identificación de objetos

La identificación de los objetos es el corazón del DOO. Abbott ofrece una valiosa discusión al respecto:

> *La OO enfatiza la importancia de la identificación precisa de los objetos y sus propiedades. Sin esta identificación cuidadosa, es casi imposible ser precisos en las mismas operaciones que han de ejecutarse y en los efectos de las mismas.*

Los nombres y frases nominales[1] de la estrategia informal son buenos indicadores de los objetos y sus clasificaciones (es decir, tipos de datos) en nuestra solución del problema.

Esta etapa consiste en reconocer los sustantivos que componen al enunciado de la etapa anterior, que son a su vez clasificados como:

Sustantivo	Significado	
	Contexto de la solución	Contexto del Diseño
Nombre común	Nombre de una clase de seres o cosas, por ejemplo "vehículo".	Una clase de objeto o abstracción de datos (clase "vehículo").
Nombre propio	Nombre de seres o cosas específicas, por ejemplo bicicleta o auto.	Una instancia de clase (de la clase "vehículo").
Nombre masivo o abstracto	Es el nombre de una cantidad o una medida, por ejemplo "tránsito".	Agrupación jerárquica de las clases de la solución.

Es importante tener en cuenta que contexto y semántica deben usarse para determinar las categorías nominales mostradas en el cuadro anterior. Una palabra puede ser un nombre común en un contexto, un nombre propio en otro y, en algunos casos, un nombre de masa o abstracto en un tercer contexto.

Después de que se han identificado todos los nombres y frases nominales, puede completarse una tabla de objetos. La tabla contiene cada nombre como un objeto; identifica si el objeto cae dentro del espacio del problema (los objetos que son parte del problema) o del espacio de solución (objetos que definen la solución del problema). Lo que puede ser mostrado en una tabla como la siguiente:

Tabla de objetos		
Objeto	Espacio	Comentario

Es importante observar que no todos los nombres y frases nominales serán de interés en la realización final de la solución. Algunos objetos, como ya hemos dicho, caerán fuera de los límites del espacio de solución. Otros objetos, aunque relevantes al problema, pueden ser redundantes o extraños cuando se refina la solución. Construyendo una tabla de objetos, se establecen los objetos potenciales y se determina el conjunto final de objetos.

Continuando con el desarrollo de la solución al problema, tenemos:

"DISEÑO Y CONSTRUCCIÓN DE UN **ENTORNO GRÁFICO DE SOFTWARE (EGS)** QUE PERMITA LA EDICIÓN DE UNA **CARTA ASM** CONCEPTUALIZADA POR EL **USUARIO**".

[1] Aquellas frases compuestas por un sustantivo y un adjetivo que no pueden ser separadas sin ofrecer un significado ambiguo

Tabla de objetos		
Objeto	Espacio	Comentario
EGS	Solución	
Carta ASM	Solución	
Usuario	Problema	

IV.3.2 Operaciones y atributos aplicados a los objetos

Una vez que se han identificado los objetos del espacio de solución, se selecciona el conjunto de operaciones que actúan sobre los mismos. Las operaciones se identifican examinando todos los **verbos** establecidos en la estrategia informal. Los verbos significan acciones u ocurrencias que en el contexto de la normalización del DOO, se consideran conjuntamente con las frases verbales[2] descriptivas y los predicados como operaciones potenciales. "Leer", "calcular", "es menor que", "determinar la diferencia entre", "es igual a", y "mantener", deberán ser todas operaciones candidatas.

Hay muchos casos en los que una operación se refiere a dos o más objetos. ¿A cuál de estos objetos debe asignarse la operación?. El Object Oriented Design Handbook presenta algunos criterios para completar este subpaso y establecer cada operación potencial:

- ✓ *Si sólo es necesario un objeto para que ocurra una operación, entonces ese es el objeto sobre el que tiene efecto la operación.*
- ✓ *Si una operación requiere el conocimiento de más de dos tipos (objetos), entonces la operación no es fundamentalmente coherente y debe rechazarse como parte de la estrategia informal.*
- ✓ *Si se requieren dos o más objetos para que ocurra una operación, entonces se debe determinar de que objeto es menester conocer la estructura interna (parte privada) por la operación.*

Aplicando estos criterios a las operaciones, derivaremos una relación entre los objetos ya identificados y sus operaciones, donde cada una de éstas se asigne a un único objeto y aquellos objetos y operaciones que existen en el espacio del problema sean desechados. Además, los objetos redundantes, extraños o abstractos (p. ej., un candidato a objeto, como el objeto "sistema" que abarca a todos los demás objetos y sus operaciones, y no añade nada a la solución del diseño) deben desecharse también; se combinarán las operaciones sinónimas y los nombres de las operaciones se extenderán para hacer más descriptivas las funciones de procesamiento.

Todos los objetos deben tener al menos una operación que actúa sobre él. Sin embargo, frecuentemente existen casos en los que un objeto parece que está solo; esto es, aparentemente no existe operación alguna que requiera información sobre la construcción subyacente del objeto. Alternativamente, se puede encontrar una operación que parezca que no se aplica a ningún objeto de la tabla. Esta situación puede ocurrir por distintas razones:

1) La estrategia informal es incompleta y una operación u objeto importante ha sido omitido.
2) Un objeto u operación que pertenece al espacio del problema ha sido asignado al espacio de soluciones (o viceversa).
3) Una operación mostrada en la tabla de operaciones de los objetos, requiere conocimiento del objeto que "permanece solo", pero esta relación no ha sido reconocida.

[2] Frases verbales: Aquellas compuestas por un verbo y un calificador y que no pueden desprenderse de cualquiera de ellos sin perder su significado

4) La estrategia informal se ha escrito a diferentes niveles de abstracción; esto es, se especifican los objetos u operaciones de bajo nivel, pero sólo se tratan las operaciones u objetos de alto nivel.

En resumen, es crucialmente importante escribir, editar, revisar y reeditar la estrategia informal dada, así como el tratar a los objetos y operaciones a un nivel consistente de abstracción.

Frecuentemente es necesario asociar atributos a los objetos y operaciones. Un atributo nos ayuda a comprender las características del objeto u operación. Los **adjetivos** y **adverbios** se convierten en atributos de los objetos y operaciones, respectivamente. Cada uno de los atributos debe ser definido completamente. Sin embargo, la definición puede posponer su refinamiento hasta que se restablezca la estrategia a un nivel más bajo de abstracción.

En algunos casos, los atributos de los objetos pueden combinarse directamente en el nombre del objeto. Esto ocurre sólo cuando se establece la estructura informal a un nivel relativamente alto de abstracción. Conforme se refinan estos objetos y sus respectivas operaciones, puede que los atributos haya que desmontarlos del objeto y especificarlos en detalle. A continuación se presenta el formato para este refinamiento:

Tabla de operaciones de objetos con atributos					
Objeto	Operación	Espacio	Atributo del Objeto	Atributo de la operación	Comentarios

Retomando el enunciado de la solución del problema, señalemos con cursivas la presencia de verbos, frases verbales, adjetivos y adverbios:

"*DISEÑO Y CONSTRUCCIÓN* DE UN **ENTORNO GRÁFICO DE SOFTWARE (EGS)** QUE *PERMITA LA EDICIÓN* DE UNA **CARTAS ASM** *CONCEPTUALIZADA* POR EL **USUARIO**".

Observemos que Cartas ASM no presenta una operación asignada a él, lo que se explica analizando las Figs. IV.1 y IV.2, así como su participación en la frase "permita la edición de Cartas ASM" y en "Carta ASM conceptualizada por el Usuario". En realidad este sustantivo forma parte del problema, lo que quedaría más claro si lo reemplazamos por "conceptualización de una Carta ASM".

A continuación presentamos la actualización de la tabla junto con las operaciones relacionadas con los objetos:

Tabla de operaciones de objetos con atributos					
Objeto	Operación	Espacio	Atributo Objeto	Atributo Oper.	Comentarios
--------	Diseñar, Construir	Problema			
EGS	Permitir edición	Solución			
Carta ASM		Problema			
Usuario	Conceptualizar	Problema			

IV.4 Componentes e interfaces del programa

Un aspecto importante de la calidad del diseño de software es la modularidad, esto es, la especificación de las componentes (módulos) que se combinan para formar un sistema completo. El DOO

define al objeto como una componente del sistema que se enlaza con otras componentes. Pero la definición de objetos y operaciones no es suficiente. Debemos también identificar las interfaces que existen entre los objetos y la estructura global de éstos (considerada en un sentido arquitectónico).

Aunque una componente de programa es una abstracción de diseño, debe ser representada en el contexto del lenguaje de programación con el que se implementará el diseño. Para acomodar el DOO, el Lenguaje de Diseño de Programa (LDP) que vaya a usarse para la construcción debe ser capaz de crear algo como lo siguiente:

```
PAQUETE nombre-componente-sistema ES
   TIPO especificación de objetos de datos
      ...
   PROC especificación de operaciones relacionadas
      ...
PRIVADO
   detalles de estructuras de datos para los objetos
CUERPO DE PAQUETE nombre-componente-sistema ES
   PROC operación.1 (descripción de interface) ES
      ...
   FIN
   PROC operación.n (descripción de interface) ES
      ...
   FIN
FIN nombre-componente-sistema
```

Refiriéndonos al LDP mostrado anteriormente, se especifica una componente del sistema indicando sus datos y operaciones. La parte de especificación de la componente indica todos los datos (declarados con la sentencia TIPO) y las operaciones (PROC para procedimiento) que actúan sobre ellos. La parte privada (PRIVADO) de la componente suministra otros detalles ocultos de las estructuras y procesamiento de los datos. En el contexto de la discusión anterior, el PAQUETE es conceptualmente similar a un objeto.

La primera componente del sistema que hay que identificar, debe ser el módulo de más alto nivel desde el que se originan todos los procesamientos y todas las estructuras de datos. Debe observarse que no se debe hacer ningún intento para especificar la información de la parte privada de los objetos de datos. Los detalles de construcción se considerarán más tarde.

Una vez que se han identificado las componentes del sistema, se está preparado para examinar el diseño subsecuente y establecer críticamente la necesidad de los cambios. Frecuentemente, una revisión de la definición de "primer corte" de los paquetes dará como resultado modificaciones, que añaden o devuelven nuevos objetos de datos a los primeros pasos del DOO (es decir, la estrategia informal), para establecer la entereza de las operaciones que se hayan especificado.

La interface de la componente deberá reconocerse como aquellas operaciones que estarán a disposición de otras componentes para accesar a su parte privada o pública, ya sea para consultar su estado actual o para requerir una operación.

IV.5 Representaciones gráficas para el DOO

Las componentes del sistema pueden representarse gráficamente para ayudar a establecer las conexiones de la interface y dar una forma sencilla de reconocer la representación del diseño. Booch propone una notación, llamada algunas veces diagramas de Booch (Fig. IV.3), para las componentes del sistema.

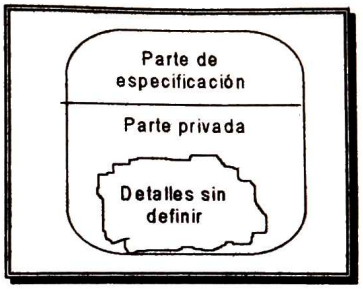


Fig. IV.3. Representación gráfica de una componente del sistema.

En la Fig. IV.3, una componente del sistema (objeto) se representa por una caja que puede ser dividida en una parte de especificación (visible al exterior) y una parte privada que permanece oculta al exterior. La "nube" sin forma representa los detalles de la parte privada o cuerpo del objeto sin especificar.

Una forma más específica la representamos como se muestra en la Fig. IV.4. Los objetos de datos se representan como rectángulos redondeados, mientras que las operaciones que actúan sobre los objetos se representan mediante rectángulos normales. De nuevo, la forma de "nube" se utiliza para representar los detalles de implantación actualmente indefinidos.

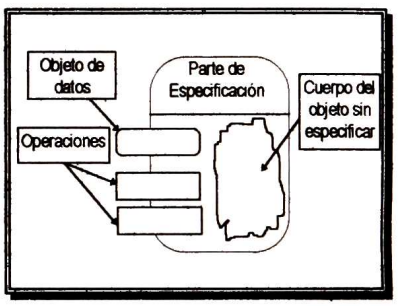


Fig. IV.4. Notación de paquete (objeto).

La Fig. IV.5 ilustra el uso del diagrama de Booch para representar las dependencias entre componentes del sistema. Las flechas de conexión indican dependencia, esto es, el paquete o componente en la punta de flecha, depende del paquete o componente de programa en el origen de la flecha. En la figura, la componente de sistema de más alto nivel, X, depende de los objetos y operaciones contenidas en el paquete 1 y paquete 2 para satisfacer su función. El paquete 2 depende de los objetos y operaciones contenidas en el paquete 3 y 4.

El uso de notación gráfica para el DOO no es esencial, pero da una indicación de la dependencia entre los paquetes (objetos y operaciones), que falta en una representación LDP. El diagrama de Booch se utiliza normalmente para representar las componentes del programa a un nivel relativamente alto de abstracción. Cuando comienza el diseño de detalle de construcción, se abandona la notación gráfica y se utiliza el LDP como representación del diseño.

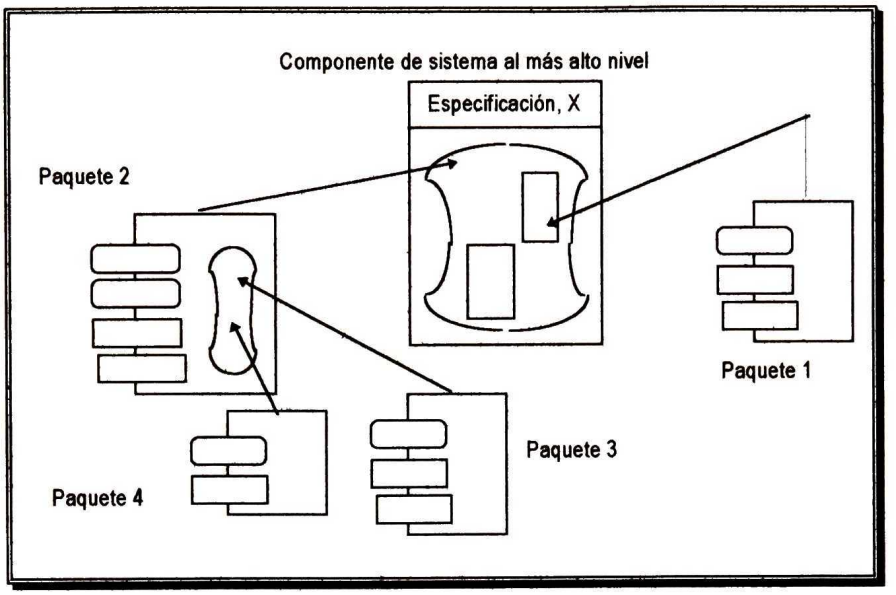


Fig. IV.5. Componente e interface del sistema y diagrama de Booch.

IV.6 Detalle de implementación

El paso de diseño detallado del DOO es similar en muchos aspectos al de cualquier metodología de diseño del software. Se describen las interfaces en detalle; se refinan y especifican las estructuras de datos; los algoritmos se diseñan para cada unidad de programa, usando los conceptos fundamentales de diseño, tales como refinamiento sucesivo y programación estructurada. La diferencia clave para el DOO es que el proceso descrito anteriormente puede aplicarse recursivamente. De hecho, la definición recursiva de la estrategia de la solución es esencial para adquirir un nivel de diseño y de abstracción de datos, a partir del cual pueda derivarse el detalle de implementación.

IV.7 Desarrollo simplificado del método

Para proceder a aplicar el método repasemos las actividades efectuadas hasta este momento:

1. El problema fue definido en términos de una necesidad y,
2. se estableció una estrategia informal.

Nos queda entonces el proceso de formalización de la estrategia -que dicho sea de paso, es el objeto de la recursividad-, para lo que contamos con las siguientes directivas:

a) Generar un enunciado que dé solución al problema o que describa a un objeto para reducir su abstracción, e identificar objetos señalando los sustantivos que componen al enunciado.
b) Obtener las operaciones a partir de los verbos, así como los atributos al reconocer los adjetivos y adverbios del enunciado.

c) Identificar y establecer las interfaces entre objetos con la ayuda de los diagramas de Booch y generar el código necesario.

d) Decidir el diseño detallado para codificar la funcionalidad de las operaciones, atributos y objetos.

Para efectos prácticos podemos elaborar una herramienta que nos ayude a la aplicación del método y que ilustre a la vez su evolución. Esta herramienta concentrará los pasos recursivos de la metodología, de forma que se facilite su entendimiento, a la vista de sus cuatro elementos:

SEGMENTO DE NIVEL SUPERIOR: Ilustra un enunciado o descripción realizado a un nivel "inicial" de abstracción, así como las operaciones, objetos y atributos desprendidos de allí.

ENUNCIADO DEL PROBLEMA O DESCRIPCIÓN DE OBJETOS DE NIVEL SUPERIOR	
Tabla de Operaciones, OBjetos y Atributos de Nivel Superior (TOOBANS)	Consideraciones y comentarios a la identificación de los elementos de la tabla adjunta de nivel superior.

El enunciado del problema debe apegarse a las reglas que del método informal se disponen, cada objeto será subindexado con un número de referencia progresivo que permitirá, dado el caso, aclarar redundancias. La identificación de los objetos, operaciones y atributos para llenar la TOOBANS se efectuará como sigue:

- **Objetos**: Se identificarán en primer lugar -en apego a la directiva 'a'- y se indicarán en **negrita** los sustantivos candidatos a objeto.
- *Operaciones*: Se identifican en segundo lugar -directiva 'b'- y se indicarán con los verbos candidatos en *cursiva o itálica.*
- Atributos: Se identifican en último lugar como los adjetivos y adverbios del enunciado, permaneciendo con tipografía ordinaria.

SEGMENTOS DE NIVEL INMEDIATO INFERIOR: Básicamente es un bloque formado por réplicas del segmento de nivel superior desarrolladas sobre la TOOBANS, mostrando la recursividad del método al presentar para todos y cada uno de los candidatos a objeto enlistados lo siguiente:

- Un enunciado que lo describe y reduce su abstracción.
- Una Tabla de Operaciones, OBjetos y Atributos de Nivel Inmediato Inferior (TOOBANII).
- Los comentarios y observaciones a las directivas aplicadas al enunciado del segmento para generar la TOOBANII.

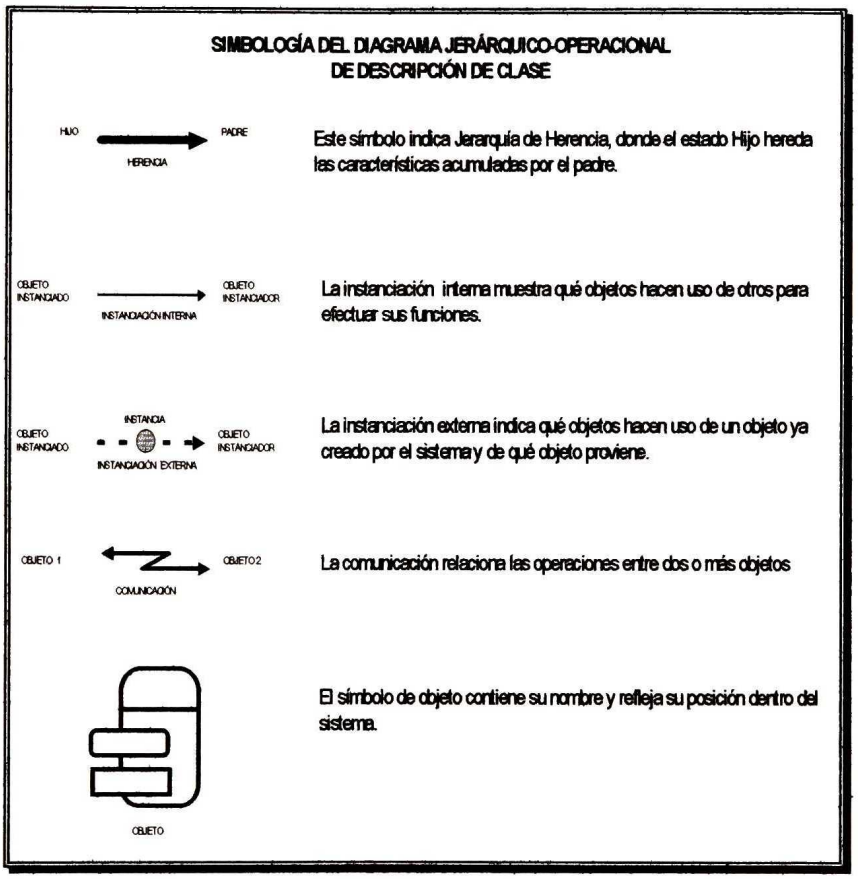
Lo anterior se repite hasta agotar los elementos de la TOOBANS.

Enunciado/Descripción del candidato a objeto miembro de la TOOBANS.	Tabla de Operaciones, Objetos y Atributos de Nivel Inmediato Inferior (TOOBANII).	Comentarios y Observaciones a la generación de la TOOBANII.

DIAGRAMAS DE BOOCH: Presenta un estado actual del proceso, mostrando gráficamente los elementos del diseño y la relación existente entre los mismos, auxiliando así a la definición de las interfaces descrita con

el LDP, cumpliendo con la directiva 'c'. Aunque este elemento se desarrolla con cada uno de los segmentos, se presentarán los diagramas hacia el final del ciclo (agotamiento de candidatos a objetos para todos los candidato a objeto primitivo). Se hará uso de dos tipos de diagramas, el de descripción de clase y el de definición de clase. En el primero se muestra la jerarquía de clase aunado a la relación existente entre las clases mismas, y en el segundo se mostrará el diagrama de objeto de Booch correspondiente a cada clase; este último puede también ser reemplazado por la definición completa de clase, desarrollada sobre el LDP en la última fase del método.

Se han tomado una serie de convenciones para mostrar con claridad las relaciones existentes entre las clases (para el diagrama de descripción de clase), que es conveniente explicar y que son presentadas en la siguiente tabla:



Adicionalmente al diagrama de descripción de clase, se presenta una discusión de objetos, donde se determinan las operaciones que se derivan del análisis-diseño o se modifica la estructura propuesta por el diseño.

DETALLE DE IMPLEMENTACIÓN: Es una lista con las instrucciones necesarias para realizar los procedimientos que llevarán a efecto las operaciones, y con las declarativas necesarias para realizar los procedimientos. Este elemento se realiza cuando las interfaces están ya definidas y dada su magnitud no se incluirán como parte impresa del presente trabajo.

A partir de aquí, haremos uso de la metodología sin detallar los pasos de la misma.

Diseño y *Construcción* de un **Entorno Gráfico de Software** (**EGS**) que permita la edición de una **Carta ASM** conceptualizada por el **Usuario.**

OBJETO	ESPACIO	OPERACIÓN	ATRIBUTO
----------	Problema	Diseñar, Construir	
EGS_1	Solución	Permitir edición	
Carta ASM_2	Problema		
$Usuario_3$	Problema	Conceptualizar	

Para el caso de ----------, *Diseñar* y *Construir* se consideran operaciones miembro del problema puesto que es una declaración del problema mismo y no hacen referencia a un objeto del presente enunciado.

EGS es una frase nominal que se considera indivisible y *Permitir edición* es un verbo que actúa directamente sobre **Cartas ASM**.

Cartas ASM aquí se refiere a la conceptualización de la **Carta ASM**, y se cataloga como parte del problema.

El EGS_1 será una interface que *presente* al **Usuario** un **Área** sobre la que serán *visualizados* y *manejados* los **Elementos De Edición** (**EDE**) y donde estará disponible un **Conjunto de Elementos Gráficos** (**CEG**) que permitan *manipular*, *organizar* e *interpretar* (PMOI) la **Información**.

OBJETO	ESPACIO	OPERACIÓN	ATRIBUTO
EGS_1	Problema	Presentar Área	interface
$Usuario_3$	Problema	Manejar Elem.	
$Área_4$	Solución	Visualizar Elem.	
EDE_5	Solución	Edición	
CEG_6	Solución	PMOI	Estará disponible
$Información_7$	Solución	----------	

El problema ahora es la definición del **EGS** y todo lo que hace referencia a él (**Usuario** y **EGS**) es parte de su dominio. Manejados se asigna a **Usuario** por contestación a la pregunta ¿Quién maneja x sobre el **Área**?.

PMOI, son operaciones del **CEG** y no de la **Información**, que es un sinónimo de lo que será visualizado y manejado dentro del **Área**.

La interface, en términos de un enunciado como éste, se considera un sinónimo y es colocado como atributo en la TOOBANII.

El EGS_1 será una interface que *presente* al **Usuario** un **Área** sobre la que serán *visualizados* y *manejados* los **Elementos De Edición** (**EDE**) y donde estará disponible un **Conjunto de Elementos Gráficos** (**CEG**) que permitan *manipular*, *organizar* e *interpretar* (PMOI) la **Información**

OBJETO	ESPACIO	OPERACIÓN	ATRIBUTO
EGS_1	Problema	Presentar Área	Interface
$Usuario_3$	Problema	Manejar Elem.	
$Área_4$	Solución	Visualizar Elem.	Información$_7$ a manejar, Estará disponible
EDE_5	Solución	Edición	
CEG_6	Solución	PMOI	Estará disponible

La forma en que la interface interactúa con el **usuario** supone la necesaria existencia de un dispositivo que permita la introducción de datos, teclado para el texto y un apuntador para las posiciones. Podemos intuir estos objetos y esperar a que sean definidos en otra fase de más detalle.

Área comparte el atributo dado por la frase estará disponible que en términos de una interface gráfica se traduce en "*Mostrar*" y la frase "*serán visualizados*" que reafirma este concepto.

El **EGS** por ser el problema mismo queda descartado, así como **Información**, que es tanto una redundancia del **EDE** como un atributo del **Área** y como **Usuario** que es parte evidente de la definición del problema original.

El **Área** deberá *mostrar* al **Usuario** una **Superficie** *delimitada* donde *construirá* el **Diagrama de la Carta ASM (D. Carta ASM)** *ensamblando* entre sí **EDE** e indicando su secuencia. Deberá también *mostrar* una **Lista con las Operaciones** (**Lista Oper.**) que *administrarán* al **EGS**, al Contenido del **Espacio Disponible** para la **Edición** y facilitará el *guardar* y *recuperar* el **Trabajo Hecho**.

OBJETO	ESPACIO	OPERACIÓN	ATRIBUTO
$Área_4$	Problema	Mostrar	
$Usuario_3$	Problema	Construir, Ensamblar, Indicar Sec.	
$Superficie_8$	Solución	Delimitar	Esp. Disp. Trab. Hecho
D. Carta ASM_2	Solución	Ensamblar	
EDE_5	Problema		
$Lista\ Oper_9$	Solución	Administrar, grabar, recuperar	
EGS_1	Problema	Presentar	

De este enunciado se desprende que el **Área** consta de una **superficie**, una **lista de operaciones** y un **espacio disponible**, que contendrá al **D. Carta ASM** compuesto por los **EDE** de la carta seleccionados de algún lugar dentro del **Área**. Llamémosle a este ambiente **Área de Trabajo**.

Área y **Usuario** quedan descartados de la tabla por pertenecer al problema, mientras que **Trabajo Hecho** también deberá descartarse por redundar con el contenido del **espacio disponible**. Por otra parte **Edición**, que puede confundirse con otro objeto puede deslindarse al responder a ¿Qué es lo que se edita? (La **Carta ASM**) y ¿Quién la edita? (El **Usuario**), por lo tanto es una operación de un objeto del problema, específicamente del **Usuario**. El **Espacio Disponible** es un sinónimo de superficie que contiene al **Trabajo Hecho.**

Los **Elementos De Edición (EDE)** son *definidos* por el **Diagrama de Carta ASM (D. Carta ASM)** como **Figuras Geométricas (Figuras Geo.)** que representan **Estado, Salida Condicional y Decisión**, que son los **Elementos de Cartas ASM** y deberán estar disponibles para su *selección* y *uso* en la *construcción* del **Diagrama de la Carta ASM (D. Carta ASM)**.

OBJETO	ESPACIO	OPERACIÓN	ATRIBUTO
EDE_5		----------	
D. Carta ASM_2	Problema	Definir	
Elemento ASM_{10}	Problema	Seleccionar, Construir (usar)	Figura Geo., Edo., Salida C y Decisión
	Solución		

Los **EDE** y **D. Carta ASM** son parte de la descripción del problema, deben ser eliminados.

Estado, **Salida Cond.** y **Decisión** son los elementos definidos en la teoría de **Cartas ASM**.

Figuras geométricas es un objeto vinculado estrechamente con los **Elementos ASM**, de hecho es una extensión que añade la característica geométrica a la definición, es por tanto, un atributo. Lo mismo puede decirse de **Estado**, **Salida** y **Decisión**, que sólo extienden la definición de **Elemento ASM** a este nivel de abstracción.

El **CEG** es un *grupo* compuesto de **Dispositivos** localizados en la **Pantalla** dedicados a *controlar la forma* en que *interactúa* con el **Área** al *modificar su escala*, *desplazándola* para ser observada por el **Usuario** y *mostrando* los ***Mensajes De las Operaciones (MDO)*** en una **Ventana** *desplegada* dentro del **Área**. Un dispositivo **Apuntador** y un **Cursor de Texto** ayudarán a *introducir* los datos y a *dirigir las acciones* del resto.

OBJETO	ESPACIO	OPERACIÓN	ATRIBUTO
CEG_6	Problema	Agrupar	Grupo Comp.
$Dispositivos_{11}$	Solución	Controlar forma, Interactuar, Modif. escala, Desplazar	Localizados en Pantalla
$Pantalla_{12}$	Solución	Mostrar	
$Área_4$	Problema	Mostrar	
$Usuario_3$	Problema	Observar	
$Ventana_{13}$	Solución	Mostrar MDO, Desplegar	
$Apuntador_{14}$	Solución	Seleccionar Dispositivos	
$Cursor\ Texto_{15}$	Solución	Introducir datos, Dirigir acciones	

El enunciado propone la existencia implícita de los objetos que interactúan con el **Área** y que se pueden identificar también como atributos del objeto **Dispositivos.**

La operación *Muestra* de **Pantalla** se deduce del atributo de **Dispositivos** localizados. Por otra parte, si **Dispositivos** designa a elementos que forman parte de la interface y **Pantalla** al elemento físico de vídeo del sistema, podríamos catalogar **Dispositivos** físico al componente de hardware y como **Dispositivo** lógico al componente del programa que refleja un elemento de control.

El **Área** deberá *mostrar* al **Usuario** una **Superficie** *delimitada* donde *construirá* el **Diagrama de la Carta ASM (D. Carta ASM)** *ensamblando* entre sí **EDE** e *indicando su secuencia*. Deberá también *mostrar* una **Lista con las Operaciones (Lista Oper.)** que *administrarán* al **EGS**, al Contenido del **Espacio Disponible** para la **Edición** y facilitará el *guardar* y *recuperar* el **Trabajo Hecho**.

OBJETO	ESPACIO	OPERACIÓN	ATRIBUTO
Área$_4$	Problema	Mostrar	
Usuario$_3$	Problema	Construir, Ensamblar,Indicar Sec.	
Superficie$_8$	Solución	Delimitar	Esp. Disp. Trab. Hecho
D. Carta ASM$_2$	Solución	Ensamblar	
EDE$_5$	Problema		
Lista Oper$_9$	Solución	Administrar, grabar, recuperar	
EGS$_1$	Problema	Presentar	

De este enunciado se desprende que el **Área** consta de una **superficie**, una **lista de operaciones** y un **espacio disponible**, que contendrá al **D. Carta ASM** compuesto por los **EDE** de la carta seleccionados de algún lugar dentro del **Área**. Llamémosle a este ambiente **Área de Trabajo**.

Área y **Usuario** quedan descartados de la tabla por pertenecer al problema, mientras que **Trabajo Hecho** también deberá descartarse por redundar con el contenido del **espacio disponible**. Por otra parte **Edición**, que puede confundirse con otro objeto puede deslindarse al responder a ¿Qué es lo que se edita? (La **Carta ASM**) y ¿Quién la edita? (El **Usuario**), por lo tanto es una operación de un objeto del problema, específicamente del **Usuario**. El **Espacio Disponible** es un sinónimo de superficie que contiene al **Trabajo Hecho.**

La **Superficie$_8$** aparentará *extenderse* más allá de la **Pantalla** de forma que pueda obtenerse una mayor amplitud siempre que se requiera. Debe también *conservar* y *mostrar* consistentemente su **Contenido**.

OBJETO	ESPACIO	OPERACIÓN	ATRIBUTO
Superficie$_8$	Problema	Extender, Contener, Mostrar, Conservar Contenido	Obtiene mayor Amplitud
Pantalla$_{12}$	Problema	Mostrar	

A primera vista en este enunciado no existen objetos de solución y esto nos indica que en realidad estamos describiendo un atributo de objeto o su sinónimo. Si **superficie** es un sinónimo de **Área**, entonces debemos suponer la existencia de las operaciones adecuadas para manejarla, como por ejemplo, *abrir*, *cerrar*, *ampliar* o *definir*, todas ellas serán atributos u operaciones de **Área de trabajo**.

Una **Carta ASM$_2$** es un **Diagrama** que *organiza* la *ejecución* de un **Algoritmo** a lo largo del **Flujo** representado por los **Elementos de Cartas ASM (Elementos ASM)** de su Método. Es una **Entidad** que puede *imprimirse* en cualquier momento, *recuperarse* y *almacenarse* de y en disco.

OBJETO	ESPACIO	OPERACIÓN	ATRIBUTO
Carta ASM$_2$	Problema	Organizar, ejecutar, imprimir, Recuperar, Almacenar	Diagrama, Entidad
Algoritmo$_{16}$	Solución	Ejecutar Flujo	
Elementos ASM$_{10}$	Solución	Representar	

Ya que **Diagrama** profundiza en la descripción de lo que es una **Carta ASM** se considera un atributo, al igual los términos **algoritmo** e **entidad** hacen mención a **Cartas ASM** de una forma en que se consideran un sinónimo.

Aquí observamos como **Cartas ASM** y **elementos** de **Cartas ASM** mantienen una relación. Esto lo habíamos especificado en la definición a **EDE**.

La **Lista de Operaciones** podrá *mostrarse* por un **Menú Desplegable de Texto (MDT)** o por un **Menú De iconos (MDI)** que *agrupen* actividades acerca de la operación del **EGS** como son la *inicialización* del **Área de trabajo,** la **Función** que permite la vista *ampliada* y/o *reducida* del contenido del **Área de trabajo** y la capacidad de *cambiar* **Parámetros** del sistema.

OBJETO	ESPACIO	OPERACIÓN	ATRIBUTO
Lista de Oper$_9$	Problema	Enlistar, Mostrar, Recup. y Almac. de disco, reducir y ampliar conten. área	Funciones
Área$_4$	Solución	Inicializar	Contenido
Parámetros$_{17}$	Solución	Cambiar	
MDT$_{18}$	Solución	Agrupar, Mostrar	Actividades
MDI$_{19}$	Solución	Agrupar, Mostrar	Actividades, Iconos

El **Menú** es una forma de *mostrar* las Operaciones de los diferentes objetos del sistema y de ponerlas a disposición del usuario, claro que no es la única forma pues las operaciones también pueden mostrarse como iconos específicos que no dejan por eso de clasificarse dentro de un **Menú (Menú de iconos)**.

En este enunciado, la descripción en términos de un objeto ya definido se convierte en un objeto de la solución del problema que nos ayuda a conceptualizar la naturaleza de las funciones que se incluyen en la lista. Es de esperarse que esta lista contenga una serie de conexiones con otros objetos, actuando como una interface entre ellas. Los objetos de este tipo ceden sus operaciones a los objetos "primitivos" u originales, ayudando a definirlos.

Contenido es un atributo inherente a la definición de **Área de Trabajo**. Sin embargo, este atributo nos manifiesta la existencia de una función pública que permitirá reducir y ampliar la apariencia del **Área**. Recuperar y Almacenar en disco son frases verbales que forman dos operaciones de la **lista**. El sustantivo contenido del **Área** denota una relación entre la **lista** y el **Área de trabajo**.

La **Superficie$_8$** aparentará *extenderse* más allá de la **Pantalla** de forma que pueda obtenerse una mayor amplitud siempre que se requiera. Debe también *conservar* y *mostrar* consistentemente su **Contenido**.

OBJETO	ESPACIO	OPERACIÓN	ATRIBUTO
Superficie$_8$	Problema	Extender, Contener, Conservar, Mostrar Contenido	Obtiene Mayor Amplitud
Pantalla$_{12}$	Problema	Mostrar	

A primera vista en este enunciado no existen objetos de solución y esto nos indica que en realidad estamos describiendo un atributo de objeto o su sinónimo. Si **superficie** es un sinónimo de **Área**, entonces debemos suponer la existencia de las operaciones adecuadas para manejarla, como por ejemplo, *abrir*, *cerrar*, *ampliar* o *definir*, todas ellas serán atributos u operaciones de **Área de trabajo**.

Una **Carta ASM$_2$** es un **Diagrama** que *organiza* la *ejecución* de un **Algoritmo** a lo largo del **Flujo** representado por los **Elementos de Cartas ASM (Elementos ASM)** de su Método. Es una **Entidad** que puede *imprimirse* en cualquier momento, *recuperarse* y *almacenarse* de y en disco.

OBJETO	ESPACIO	OPERACIÓN	ATRIBUTO
Carta ASM$_2$	Problema	Organizar ejecutar, Imprimir, Recuperar, Almacenar	Diagrama, Entidad
Algoritmo$_{16}$	Problema	Ejecutar	Flujo
Elementos ASM$_{10}$	Solución	Representar	

Ya que **Diagrama** profundiza en la descripción de lo que es una **Carta ASM** se considera un atributo, al igual los términos **algoritmo** e **entidad** hacen mención a **Cartas ASM** de una forma en que se consideran un sinónimo.

Aquí observamos como **Cartas ASM** y **elementos** de **Cartas ASM** mantienen una relación. Esto lo habíamos especificado en la definición a **EDE**.

Estado, **Salida Condicional (Sal Cond)** y **Decisión** son todos **Elementos ASM** que conforman el **Flujo Estructural** que describe a una **Carta ASM**. Puede ser: *presentado*, *seleccionado* para *editarse*, *movido* a través del **Área** y *modificado* su **Contenido**.

OBJETO	ESPACIO	OPERACIÓN	ATRIBUTO
Estado$_{20}$	Solución	Presentar, Seleccionar, Mover, Modificar	Contenido
Sal Cond.$_{21}$	Solución	Presentar, Seleccionar, Mover, Modificar	Contenido
Decisión$_{22}$	Solución	Presentar, Seleccionar, Mover, Modificar	Contenido
Elem ASM$_{10}$	Problema		
Carta ASM$_2$	Problema	Describir	Flujo Estructural
Área$_4$	Problema		

En este enunciado nos auxiliamos de una descripción que pertenece a un nivel anterior. Las operaciones Presentar, Seleccionar, Mover y Modificar y el atributo de contenido debieron obtenerse en aquel nivel y ahora debe manejarse como operaciones comunes a todos los elementos.

Existe relación con **Área** y **EDE**.

Flujo Estructural describe a la **Carta ASM** por lo que es un atributo.

Estado, **Salida Condicional (Sal Condicional)** y **Decisión** son todos **Elementos ASM** que conforman el **Flujo Estructural** que describe a una **Carta ASM**. Puede ser: *presentado*, *seleccionado* para *editarse*, *movido* a través del **Área** y *modificado* su **Contenido**.

Objeto	Espacio	Operación	Atributo
Estado$_{20}$	Solución	Presentar, Seleccionar Mover, Modificar,	Contenido
Sal Condicional$_{21}$	Solución	Presentar, Seleccionar, Mover, Modificar	Contenido
Decisión$_{22}$	Solución	Presentar, Seleccionar, Mover, Modificar	
Elementos ASM$_{10}$	Problema		Contenido
Carta ASM$_2$	Problema	Describir	
Área$_4$	Problema		Flujo Estructural

En este enunciado nos auxiliamos de una descripción que pertenece aun nivel anterior. Las operaciones Presentar, Seleccionar, Mover y Modificar y el atributo de **contenido** debieron obtenerse en aquel nivel y ahora debe manejarse como operaciones comunes a todos los elementos.

Existe relación con **Área** y **EDE**.

Flujo Estructural describe a la **Carta ASM** por lo que es un atributo.

Estado$_{20}$ es una entidad compuesta por un rectángulo, una etiqueta de nombre, una etiqueta de código y una lista de variables de salida, todos ellos *editables*. Es el destino de un flujo y la fuente de otro.

Objeto	Espacio	Operación	Atributo
Estado$_{20}$	Problema	Editar	Rectángulo, Código, Etiq. Nombre, Etiq. Código, Lista de Var., Entidad

Se detiene la abstracción para los atributos de **estado**. Haciendo de este objeto terminal, sin embargo, existe la posibilidad de definir un nuevo objeto "Objeto".

Entidad podemos considerarlo como un sinónimo de Estado.

Salida Condicional$_{21}$ es una entidad compuesta por un Rectángulo de esquinas redondeadas y una lista de variables de salida también *editables*. Es el destino de un flujo y la fuente de otro.

Objeto	Espacio	Operación	Atributo
Sal Condicional$_{21}$	Problema	Editar	Rectángulo Redondeado, Lista de Var., Entidad

De acuerdo a la teoría de Cartas ASM, el flujo fuente debe provenir necesariamente de una **decisión**. Este es un objeto del desarrollo del método.

Decisión$_{22}$ es una entidad compuesta por una figura rómbica y una variable de prueba *editable*. Tiene dos flujos de salida, uno etiquetado con un "1" y el otro con un "0" dirigidos ambos hacia otros elementos.	**OBJETO**	**ESPACIO**	**OPERACIÓN**	**ATRIBUTO**	La teoría de Cartas ASM indica a éste como el único con más de un flujo de salida. Es también un objeto terminal del desarrollo de análisis y diseño.
	Decisión$_{22}$	Problema	Editar	Fig. Rómbica, Var. Prueba, Entidad	

Estado$_{20}$ es una entidad compuesta por un rectángulo, una etiqueta de nombre, una etiqueta de código y una lista de variables de salida, todos ellos *editables*. es el destino de un flujo y la fuente de otro.

Objeto	**Espacio**	**Operación**	**Atributo**
Estado$_{20}$	Problema	Editar	rectángulo, Código, Etiq. Nombre, Etiq. Código, Lista de Var., Entidad

Se detiene la abstracción para los atributos de **estado**. Haciendo de este objeto terminal, sin embargo, existe la posibilidad de definir un nuevo objeto "Objeto".

Entidad podemos considerarlo como un sinónimo de Estado.

Salida Condicional$_{21}$ es una entidad compuesta por un Rectángulo de esquinas redondeadas y una lista de variables de salida también *editables*. Es el destino de un flujo y la fuente de otro.

Objeto	**Espacio**	**Operación**	**Atributo**
Sal Condicional$_{21}$	Problema	Editar	Rectángulo redondeado, Lista de Var., Entidad

De acuerdo a la teoría de Cartas ASM, el flujo fuente debe provenir necesariamente de una **decisión**. Éste es un objeto del desarrollo del método.

Decisión$_{22}$ es una entidad compuesta por una figura rómbica y una variable de prueba *editable*. Tiene dos flujos de salida, uno etiquetado con un "1" y el otro con un "0" dirigidos ambos hacia otros elementos.

Objeto	**Espacio**	**Operación**	**Atributo**
Decisión$_{22}$	Problema	Editar	Fig. Rómbica,Var. Prueba, Entidad

La teoría de Cartas ASM indica a éste como el único con más de un flujo de salida. Es también un objeto terminal del desarrollo de análisis y diseño.

La **Lista de Operaciones (Lista Oper)** podrá *mostrarse* por un **Menú Desplegable de Texto (MDT)** o por un **Menú De iconos (MDI)** que *agrupen* actividades acerca de la operación del **EGS** como son la *inicialización* del **Área de trabajo,** la **Función** que permite la vista *ampliada* y/o *reducida* del contenido del **Área de trabajo** y la capacidad de *cambiar* **Parámetros** del sistema.

Objeto	**Espacio**	**Operación**	**Atributo**
Lista Oper$_9$	Problema	Enlistar, Mostrar, Recup. y Almac. de disco, reducir y ampliar conten. área	Funciones
Área$_4$	Solución	Inicializar	Contenido
Parámetros$_{17}$	Solución	Cambiar	
MDT$_{18}$	Solución	Agrupar, Mostrar	Actividades
MDI$_{19}$	Solución	Agrupar, Mostrar	Actividades, Iconos

El **Menú** es una forma de *mostrar* las Operaciones de los diferentes objetos del sistema y de ponerlas a disposición del usuario, claro que no es la única forma pues las operaciones también pueden mostrarse como iconos específicos que no dejan por eso de clasificarse dentro de un **Menú (Menú de iconos)**.

En este enunciado, la descripción en términos de un objeto ya definido se convierte en una objeto de la solución del problema que nos ayuda a conceptualizar la naturaleza de las funciones que se incluyen en la lista. Es de esperarse que esta lista contenga una serie de conexiones con otros objetos, actuando como una interface entre ellas. Los objetos de este tipo ceden sus operaciones a los objetos "primitivos" u originales, ayudando a definirlos.

Contenido es un atributo inherente a la definición de **Área de Trabajo**. Sin embargo, este atributo nos manifiesta la existencia de una función pública que permitirá reducir y ampliar la apariencia del **Área**. Recuperar y Almacenar en disco son frases verbales que forman dos operaciones de la **lista**. El sustantivo contenido del **Área** denota una relación entre la lista y el **Área de trabajo**.

Los **Parámetros$_{17}$** son un grupo de **Valores** modificables que *determinan características* del **Software**.

Objeto	**Espacio**	**Operación**	**Atributo**
Parámetros$_{17}$	Problema		
Valores$_{23}$	Solución	Determinar Características	Modificables
Software$_{24}$	Problema		

Al indicar que los **valores** hacen a un grupo de **Parámetros**, estamos mencionando un sinónimo, con la salvedad de que son objetos de un distinto nivel. **Parámetros** agrupa a los **valores** y cada valor por si mismo puede ser un tipo de atributo.

El objeto **Parámetros** lo podemos considerar opcional para el desarrollo de la aplicación, debido que ofrece características deseables pero no definitivas para la solución propuesta.

Aquí aparece un posible objeto llamado **Software**, pero como este es en realidad el problema que se desea resolver, por lo que es 100 % parte del problema.

El **Menú Desplegable de Texto (MDT)** será una barra horizontal que *enlistará* nombres de **Submenús,** que al ser *Seleccionados* por **Teclado** o **Apuntador**, *mostrarán* **Ventanas** *enlistando* **Opciones** *seleccionables* compuestas por el **Nombre de la operación** y su **Tecla rápida** para accesar más rápidamente, o **Ventanas de Diálogo** para *capturar información.*

Objeto	Espacio	Operación	Atributo
MDT_{18}	Problema	Enlistar Submenús	Barra Horiz
$Submenús_{25}$	Solución	Mostrar, Seleccionar por teclado o apuntador	Ventanas Elegibles
$Ventana_{13}$	Solución	Enlistar opciones	
Ventana $Dial._{26}$	Solución	Capturar Info	Nombre de la Tecla rápida
$Opciones_{27}$	Solución	Seleccionar Operación,	

Un **menú desplegable** es un objeto familiar en las interfaces de usuario que se maneja tanto con el **teclado** como con el **Ratón**. La lista horizontal Inicial contiene nombres de **Submenús** o en algunos casos, nombres de acciones concretas que se efectúan en cuanto se seleccionan.

Este es un ejemplo de combinación de niveles de abstracción, pues se define **Submenús**, que pertenece a un nivel inferior de abstracción o de mayor detalle.

El **Menú De Iconos$_{19}$ (MDI)** *mostrará* un **Área** con **Botones** *seleccionables* identificados por **Iconos** representando la **Operación** que *ejecutan.*

Objeto	Espacio	Operación	Atributo
MDI_{19}	Problema	Mostrar Área	
$Botones_{28}$	Solución	Seleccionar	Icono
$Operación_{29}$	Solución	Ejecutar	

El **Menú de iconos** agrupa **botones** con un atributo especial, sobre un área que puede llamarse área de menú de iconos definida sobre el **Área de trabajo**. Cada **botón** tiene un pequeño dibujo distintivo de la acción que efectúa, (Icono). La pertenencia del Icono es un atributo de **Botones** distinguido por la palabra identificados.

Los **Parámetros$_{17}$** son un grupo de **Valores** modificables que *determinan* las características del **Software**.

Objeto	Espacio	Operación	Atributo.
Parámetros17	Problema		
Valores23	Solución	Determinar Características	Modificables
Software24	Problema		

Al indicar que los **valores** hacen a un grupo de **Parámetros**, estamos mencionando un sinónimo, con la salvedad de que son objetos de un distinto nivel. **Parámetros** agrupa a los **valores** y cada valor por si mismo puede ser un tipo de atributo.

El objeto **Parámetros** lo podemos considerar opcional para el desarrollo de la aplicación, debido que ofrece características deseables pero no definitivas para la solución propuesta.

Aquí aparece un posible objeto llamado **Software**, pero como éste es en realidad el problema que se desea resolver, por lo que es 100 % parte del problema.

El **Menú Desplegable de Texto (MDT)** será una barra horizontal que *enlistará* nombres de **Submenús,** que al ser *Seleccionados* por **Teclado** o **Apuntador**, *mostrarán* **Ventanas** *enlistando* **Opciones** *seleccionables* compuestas por el **Nombre de la operación** y su **Tecla rápida** para accesar mas rápidamente, o **Ventanas de Diálogo** para *capturar* información.

Objeto	Espacio	Operación	Atributo
MDT18	Problema	Enlistar Submenús	Barra Horiz.
Submenús25	Solución	Mostrar, Seleccionar por teclado o apuntador	Ventanas Elegibles
Ventana13	Solución	Enlistar opciones	
Ventana Dial.26	Solución	Capturar Info.	
Opciones27	Solución	Seleccionar Operación	Nombre de la Tecla rápida

Un **menú desplegable** es un objeto familiar en las interfaces de usuario que se maneja tanto con el teclado como con el **Ratón**. La lista horizontal Inicial contiene nombres de **Submenús** o en algunos casos, nombres de acciones concretas que se efectúan en cuanto se seleccionan.

Este es un ejemplo de combinación de niveles de abstracción, pues se define **Submenús**, que pertenece a un nivel inferior de abstracción o de mayor detalle.

Los **Submenús25** son **Ventanas** que *Emergen* verticalmente sobre el **MDT** y *Muestran* su *contenido* seleccionable, que pueden ser: **Opciones**, Ventanas de Información o **Sub-menús.**

Objeto	Espacio	Operación	Atributo
Submenús25	Problema	Emerger verticalmente, Mostrar Contenido información.	Ventanas, Opción, Ventanas Submenús

El **SubMenú** es una característica del **MDT** que puede ser tratado como un objeto o como parte de las funciones propias del **MDT**, razón por la que hemos trasladado los elementos del contenido como atributos que más tarde serán construidos con el LDP.

La **Ventana**$_{13}$ será un área rectangular que intermediará el diálogo sistema-usuario, *mostrando mensajes* y *requiriendo acciones* **Específicas.**	**Objeto**	**Espacio**	**Operación**	**Atributo**	Una **ventana** asoma al usuario a un **mensaje** del sistema y es el medio en que en general ocurre la comunicación de la interface. Rigurosamente hablando podríamos decir que el **Área de Trabajo** es una **ventana**, sin embargo la definición de ellas que se da aquí es que sólo sirven para la presentación de texto o para el requerimiento de acciones. El **Área de trabajo** no requiere nunca de acciones por sí misma y es sólo una vista de la superficie editable.
	Ventana$_{13}$	Problema	Mostrar mensajes, Requerir acciones	Área rectangular	

Las **Ventanas Diálogo**$_{26}$ **(Ventana Diál)** son **Ventanas** *abiertas* para *mostrar* su **Contenido** al **Usuario** y/o *recibir* información, esperando la confirmación de una **Acción** efectuada sobre ella para cerrarse.	**Objeto**	**Espacio**	**Operación**	**Atributo**	Este enunciado nos aclara que una vez abierta, **la Ventana Diálogo** es cerrada con una acción sobre ella. Éste podría ser el nivel máximo de detalle para esta parte del análisis.
	Ventana Diál.$_{26}$	Problema	Abrir, Mostrar, Accionar Dentro, Cerrar, Recibir	Ventana	
	Usuario$_{3}$	Problema			

Las **Opciones**$_{27}$ son **Alternativas de Acción** *enlistadas* en los submenús utilizadas para *ejecutar funciones*.	**Objeto**	**Espacio**	**Operación**	**Atributo**	Las **opciones** pueden ser tratadas como funciones o como objetos, dependiendo del diseño. Si como funciones, entonces pertenecerán al espacio del problema.
	Opciones$_{27}$	Problema	Enlistar, Ejecutar función	Alternativa de acción	

Los **Submenús**$_{25}$ son **Ventanas** que *Emergen verticalmente* sobre el **MDT** y *Muestran su contenido* seleccionable, que pueden ser: **Opciones, Ventanas de Información o Sub-menús.**				
Objeto	**Espacio**	**Operación**	**Atributo**	El **SubMenú** es una característica del **MDT** que puede ser tratado como un objeto o como parte de las funciones propias del **MDT**, razón por la que hemos trasladado los elementos del contenido como atributos que más tarde serán construidos con el LDP.
Submenús$_{25}$	Problema	Emerger verticalmente, Mostrar Contenido información	Ventanas, Ventanas Opción, SubMenús	

La **Ventana**$_{13}$ será un área rectangular que intermediará el diálogo sistema-usuario, *mostrando mensajes* y *requiriendo acciones* **Específicas.**

Objeto	Espacio	Operación	Atributo
Ventana13	Problema	Mostrar Mensajes, Requerir Acciones	Área Rectangular

Una **ventana** asoma al usuario a un **mensaje** del sistema y es el medio en que en general ocurre la comunicación de la interface. Rigurosamente hablando podríamos decir que el **Área de Trabajo** es una **ventana**, sin embargo la definición de ellas que se da aquí es que sólo sirven para la presentación de texto o para el requerimiento de acciones. El **Área de trabajo** no requiere nunca de acciones por sí misma y es sólo una vista de la superficie editable.

Las **Ventanas Diálogo26 (Ventana Diál)** son **Ventanas** abiertas para *mostrar* su **Contenido** al **Usuario** y/o *recibir* información, esperando la confirmación de una **Acción** efectuada sobre ella para cerrarse.

Objeto	Espacio	Operación	Atributo
Ventana Diál.26	Problema	Abrir, Mostrar, Accionar Dentro, Cerrar, Recibir	Ventana
Usuario3	Problema		

Este enunciado nos aclara que una vez abierta, la **Ventana Diálogo** es cerrada con una acción sobre ella. Éste podría ser el nivel máximo de detalle para esta parte del análisis.

Las **Opciones27** son **Alternativas de Acción** *enlistadas* en los submenús utilizadas para *ejecutar* funciones.

Objeto	Espacio	Operación	Atributo
Opciones27	Problema	Enlistar, Ejecutar función	Alternativa de acción

Las **opciones** pueden ser tratadas como funciones o como objetos, dependiendo del diseño. Si como funciones, entonces pertenecerán al espacio del problema.

El **Menú De Iconos19 (MDI)** *mostrará* un **Área** con **Botones** *seleccionables* identificados por **Iconos** representando la **Operación** que *ejecutan*.

Objeto	Espacio	Operación	Atributo
MDI19	Problema	Mostrar Área	
Botones28	Solución	Seleccionar	Icono
Operación29	Solución	Ejecutar	

El **Menú de Iconos** agrupa **botones** con un atributo especial, sobre un área que puede llamarse área de menú de Iconos definida sobre el **Área de trabajo**. Cada **Botón** tiene un pequeño dibujo distintivo de la acción que efectúa, (Icono). La pertenencia del Icono es un atributo de **Botones** distinguido por la palabra identificados.

Los **Botones$_{28}$** son **Áreas Dibujadas** a semejanza de un botón físico que *reaccionan* análogamente tras su *selección*; cada botón estará ligado a la *ejecución* de una *operación* específica.	**Objeto**	**Espacio**	**Operación**	**Atributo**	Este objeto puede guardar relación directa **con Ventana Diálogo** o con cualquier **EGS** que necesite una acción de parte del usuario.
	Botones$_{28}$	Problema	Reaccionar selección, Ejecutar operación	Área dibujada a semejanza de botón Fís.	
Las **Operaciones$_{29}$** son el **Conjunto de Acciones** que pueden *ejecutarse* con y entre los **CEG** y los **EDE**.	**Objeto**	**Espacio**	**Operación**	**Atributo**	Las **operaciones** no son un objeto en sí, si no que son las funciones propias de los **CEG** y **EDE**.
	Operación$_{29}$	Problema	Ejecutar	Conjunto Acciones	

Los **Botones$_{28}$** son **Áreas Dibujadas** a semejanza de un botón físico que *reaccionan* análogamente tras su *selección*; cada botón estará ligado a la *ejecución* de una *operación* específica.

Objeto	**Espacio**	**Operación**	**Atributo**	Este objeto puede guardar relación directa con **Ventana Diálogo** o con cualquier **EGS** que necesite una acción de parte del usuario.
Botones$_{28}$	Problema	Reaccionar Selección, Ejecutar operación	Área dibujada a semejanza de botón Fis.	

Las **Operaciones$_{29}$** son el **Conjunto de Acciones** que pueden *ejecutarse* con y entre los **CEG** y los **EDE**.

Objeto	**Espacio**	**Operación**	**Atributo**	Las **operaciones** no son un objeto en sí, si no que son las funciones propias de los **CEG** y **EDE**.
Operación$_{29}$	Problema	Ejecutar	Conjunto Acciones	

Los **Elementos De Edición (EDE)** son *definidos* por el **Diagrama de Carta ASM (D. Carta ASM)** como **Figuras Geométricas (Figuras Geo)** que representan **Estado, Salida Condicional y Decisión**, que son los **Elementos de Cartas ASM** y deberán estar disponibles para su *selección* y *uso* en la *construcción* del **Diagrama de la Carta ASM**.

Objeto	Espacio	Operación	Atributo
EDE$_5$	Problema	----------	
D. Carta ASM$_2$	Problema	Definir	
Elemento ASM$_{10}$	Solución	Seleccionar, Construir (usar)	Figura Geo., Edo., Salida C, y Decisión

Los **EDE** y **D. Carta ASM** son parte de la descripción del problema, deben ser eliminados.

Estado, **Salida Cond.** y **Decisión** son los elementos definidos en la teoría de **Cartas ASM**.

Figuras geométricas es un objeto vinculado estrechamente con los **Elementos ASM**, de hecho es una extensión que añade la característica geométrica a la definición, es por tanto, un atributo. Lo mismo puede decirse de **Estado**, **Salida** y **Decisión**, que sólo extienden la definición de **Elemento ASM** a este nivel de abstracción.

El **CEG** es un grupo compuesto de **Dispositivos** localizados en la **Pantalla** dedicados a *controlar* la *forma* en que *interactúa* con el **Área** al *modificar* su escala, *desplazándola* para ser observada por el **Usuario** y *mostrando* los **Mensajes De las Operaciones (MDO)** en una **Ventana** *desplegada* dentro del **Área**. Un dispositivo **Apuntador** y un **Cursor de Texto** ayudarán a *introducir* los datos y a *dirigir* las *acciones* del resto.

Objeto	Espacio	Operación	Atributo
CEG$_6$	Problema	Agrupar	Grupo Comp.
Dispositivos$_{11}$	Solución	Controlar forma, Interactuar, Modif. escala, Desplazar	Localizados en Pantalla
Pantalla$_{12}$	Solución	Mostrar	
Área$_4$	Problema	Mostrar	
Usuario$_3$	Problema	Observar	
Ventana$_{13}$	Solución	Mostrar MDO, Desplegar	
Apuntador$_{14}$	Solución	Seleccionar Dispositivos	
Cursor Texto$_{15}$	Solución	Introducir datos, Dirigir acciones	

El enunciado propone la existencia implícita de los objetos que interactúan con el **Área**, y que se pueden identificar también como atributos del objeto **Dispositivos**.

La operación *Muestra* de **Pantalla** se deduce del atributo de **Dispositivos**: Localizados. Por otra parte, si **Dispositivos** designa a elementos que forman parte de la interface y **Pantalla** al elemento físico de vídeo del sistema, podríamos catalogar **Dispositivos** físico al componente de hardware y como **dispositivo** lógico al componente del programa que refleja un elemento de control.

Los **Dispositivos**$_{11}$ son **Elementos de Software** que *complementan* el *manejo* del **Área**, dando *acceso* a las partes que componen el **EGS**	**Objeto**	**Espacio**	**Operación**	**Atributo**	**Dispositivos** es una generalización de todos los elementos del **EGS** proyectado. Los agrupa y define su interacción con el usuario. Este tipo de enunciados denota la existencia de una generalización de la cual se desprenderán los objetos **EDE** y **CEG.** Es importante mencionar que los **Dispositivos** Lógicos comparten características como se denota si observamos que todos los **dispositivos** que se presentan en pantalla tienen que ser dibujados sobre ella, a partir de un solo punto. Todos tienen que estar autodocumentados para ser usados adecuadamente. Una forma de documentación es el empleo de iconos que expliquen la acción a seguir después de seleccionar un **dispositivo** o mediante el desplegado de un mensaje en una **ventana**. Otro objeto como por ejemplo, una **barra de desplazamiento**, necesita otro tipo de elementos, como lo es mostrar un porcentaje de avance con un cursor y unos **botones** con flechas indicadoras de la dirección.
	Dispositivos$_{11}$	Problema	Complementar manejo	Elementos de Software	
	Área$_4$	Problema	Accesar a los EGS		
La **Pantalla**$_{12}$ es el **Dispositivo Físico** de Despliegue Gráfico que *Mostrará* al **Entorno Gráfico de Software (EGS).**	**Objeto**	**Espacio**	**Operación**	**Atributo**	La **pantalla** es un elemento físico que posee ciertas características propias del hardware que afectan directamente al software, proporcionando una plataforma de ejecución que debe configurarse y prepararse para la modalidad gráfica.
	Pantalla$_{12}$	Problema	Mostrar EGS	D. Físico de despliegue gráfico	
	EGS$_1$	Problema			

El **Apuntador**$_{14}$ es un **Dispositivo** que permite el *señalamiento* y *selección* de cualquier **Punto** en la **Pantalla** que coincida con un **EDE** o **CEG**, su *ejecución* o -si es posible- su *transporte*	**Objeto** / **Espacio** / **Operación** / **Atributo** Apuntador$_{14}$ / Problema / Señalar, Seleccionar puntos, Transportar, Ejecutar / Dispositivo Punto$_{30}$ / Solución / Seleccionar Pantalla$_{12}$ / Problema EDE$_5$ / Problema CEG$_6$ / Problema	El **Apuntador** es otro elemento de software que modela a un **dispositivo** físico reconocido directamente por la computadora. Una pluma electrónica, una palanca de juego (JoyStick) o un **Ratón** (Mouse) son los más conocidos, sin embargo el sistema sólo se diseñará para funcionar con un Mouse Microsoft (Tres botones), que es el más popular actualmente. Este aparato requiere -para su reconocimiento- de programación especializada que maneje las interrupciones definidas para su manejador (Driver).
El **Cursor de Texto**$_{15}$ permitirá *reconocer* el **Lugar Actual** en que será *introducido* el texto para cualquier dispositivo que lo permita, *señalando* una posición mediante un guión de las dimensiones de un carácter.	**Objeto** / **Espacio** / **Operación** / **Atributo** Cursor de Texto$_{15}$ / Problema / Reconocer, Señalar, Seleccionar puntos, Introducir texto / Dispositivo, Lugar actual	El **Cursor de Texto** es un auxiliar del usuario para la *introducción* y *selección* del texto que compone ciertas partes de los **EDE**. A pesar de ser considerado como un objeto su instrumentación puede reemplazarse en la forma de una función de captura de texto.

Los **Dispositivos**$_{11}$ son **Elementos de Software** que *complementan* el *manejo* del **Área**, dando *acceso* a las partes que componen el **EGS**.

Objeto	Espacio	Operación	Atributo
Dispositivos$_{11}$	Problema	Complementar el manejo	Elementos de Software
Área$_4$	Problema	Accesar a los EGS	

Dispositivos es una generalización de todos los elementos del **EGS** proyectado. Los agrupa y define su interacción con el usuario. Este tipo de enunciados denota la existencia de una generalización de la cual se desprenderán los Objetos **EDE** y **CEG.** Es importante mencionar que los **Dispositivos** Lógicos comparten características como se denota si observamos que todos los **dispositivos** que se presentan en pantalla tienen que ser dibujados sobre ella, a partir de un solo punto. Todos tienen que estar autodocumentados para ser usados adecuadamente. Una forma de documentación es el empleo de iconos que expliquen la acción a seguir después de seleccionar un **dispositivo** o mediante el desplegado de un mensaje en una **ventana**. Otro objeto como por ejemplo, una **barra de desplazamiento**, necesita otro tipo de elementos, como lo es mostrar un porcentaje de avance con un cursor y unos **botones** con flechas indicadoras de la dirección.

La **Pantalla$_{12}$** es el **Dispositivo Físico** de Despliegue Gráfico que *Mostrará* al **Entorno Gráfico de Software (EGS).**

Objeto	Espacio	Operación	Atributo
Pantalla$_{12}$	Problema	Mostrar EGS	D. Físico de despliegue gráfico
EGS$_1$	Problema		

La **pantalla** es un elemento físico que posee ciertas características propias del hardware que afectan directamente al software, proporcionando una plataforma de ejecución que debe configurarse y prepararse para la modalidad gráfica.

El **Apuntador$_{14}$** es un **Dispositivo** que permite el *señalamiento* y *selección* de cualquier **Punto** en la **Pantalla** que coincida con un **EDE** o **CEG**, su *ejecución* o -si es posible- su *transporte*.

Objeto	Espacio	Operación	Atributo
Apuntador$_{14}$	Problema	Señalar, Seleccionar puntos, Transportar, Ejecutar	Dispositivo
Punto$_{30}$	Solución	Seleccionar	
Pantalla$_{12}$	Problema		
EDE$_5$	Problema		
CEG$_6$	Problema		

El **Apuntador** es otro elemento de software que modela a un **dispositivo** físico reconocido directamente por la computadora. Una pluma electrónica, una palanca de juegos (JoyStick) o un **Ratón** (Mouse) son los más conocidos, sin embargo el sistema sólo se diseñará para funcionar con un Mouse Microsoft (Tres botones), que es el más popular actualmente. Este aparato requiere -para su reconocimiento- de programación especializada que maneje las interrupciones definidas para su manejador (Driver).

Un **Punto$_{30}$** es una coordenada en pantalla que puede ser *establecido*, *dibujado*, *borrado* y *movido*.

Objeto	Espacio	Operación	Atributo
Punto$_{30}$	Problema	Establecer, Dibujar, Borrar, Mover	Coordenada

Como se mencionó para **dispositivo**, esta es la base para todos los elementos que forman el **EGS**.

Un **Punto$_{30}$** es una coordenada en pantalla que puede ser *establecido*, *dibujado*, *borrado* y *movido*.

Objeto	Espacio	Operación	Atributo
Punto$_{30}$	Problema	Establecer, Dibujar, Borrar, Mover	Coordenada

Como se mencionó para **dispositivo**, esta es la base para todos los elementos que forman el **EGS**.

El **Cursor de Texto$_{15}$** permitirá *reconocer* el **Lugar Actual** en que será *introducido* el texto para cualquier dispositivo que lo permita, *señalando* una posición mediante un guión de las dimensiones de un carácter.

Objeto	Espacio	Operación	Atributo
Cursor de Texto$_{15}$	Problema	Reconocer, Señalar, Seleccionar puntos, Introducir texto	Dispositivo, Lugar Actual

El **Cursor de Texto** es un auxiliar del usuario para la introducción y selección del texto que compone ciertas partes de los **EDE**.

A pesar de ser considerado como un objeto su instrumentación puede reemplazarse en la forma de una función de captura de texto.

"*Diseño* y *Realización* de los **Elementos Complementarios del Software de Edición (ECSE)** para *efectuar la Síntesis* de una **Carta ASM**".

Objeto	Espacio	Operación	Atributo
----------	Problema	Diseñar, Realizar	
$ECSE_1$	Solución	Efectuar la Síntesis	
Carta ASM_2	Problema		

Para el caso de ----------, *Diseñar y Realizar* se consideran operaciones miembro del problema puesto que es una declaración del problema mismo y no hacen referencia a un objeto del presente enunciado.

ECSE es una frase nominal que se considera indivisible y *efectuar síntesis* es su verbo que actúa directamente sobre **Cartas ASM**.

Cartas ASM aquí se refiere al elemento perteneciente al sistema de edición, que será aprovechado para el presente desarrollo.

Los $ECSE_1$ serán construidos como funciones, objetos y estructuras de datos, que *aprovechen* las *facilidades* otorgadas por el **Sistema de Edición (SE)**, para así obtener un **Controlador con Arquitectura MICA-I (CAMICAI)** completo que *solucione el problema*.

Objeto	Espacio	Operación	Atributo
$ECSE_1$	Problema	Aprovecha facilidades	Funciones, Objetos y Estructuras de datos
SE_3	Problema	Otorga Facilidades Solucione el P.	Completo
$CAMICAI_4$	Solución		

Los **ECSE** son considerados parte del espacio del problema puesto que es el objeto del enunciado.

La frase "*serán construidos como ...*" nos indica una analogía que puede ser considerada como sinónimo que formará parte de los atributos del objeto **ECSE**.

El **SE** es un conjunto de entidades ya existentes, de las que es menester conocer las funciones disponibles y los objetos que pueden ayudarnos en el diseño.

Los $ECSE_1$ serán construidos como funciones, objetos y estructuras de datos que *aprovechen* las *facilidades otorgadas* por el **Sistema de Edición (SE)** para así obtener un **Controlador con Arquitectura MICA-I (CAMICAI)** completo que *solucione el problema*.

Objeto	Espacio	Operación	Atributo
$ECSE_1$	Problema	Aprovecha Facilidades	Funciones, Objetos y Estructuras de datos
SE_3	Problema	Otorga Facilidades Solucionar el Problema	Completo
$CAMICAI_4$	Solución		

Las facilidades del **SE** deberán ser añadidas por los métodos necesarios para alcanzar el objetivo.

El **CAMICAI**, siendo el elemento esencial de la arquitectura **MICA I** es el objeto hacia el cual va enfocada la solución.

El **CAMICAI** es un **controlador** formado por un **Conjunto de Dispositivos Digitales (CDD)** interconectados de una manera especial para *obedecer* a las *instrucciones contenidas* en un **Elemento de Memoria Externo(EME)**, *generadas* por un **Sintetizador de la Carta ASM (SCASM)**. El **CAMICAI** *demuestra* físicamente la *solución*.

OBJETO	ESPACIO	OPERACIÓN	ATRIBUTO
$CAMICAI_4$	Problema	Demuestra Sol	Controlador
CDD_5	Solución	Obedece Instr.	Interconectar
EME_6	Solución	Contiene Instr.	
$SCASM_7$	Solución	Genera Instr.	

El posible objeto "**controlador**" es considerado un sinónimo de **CAMICAI**, por lo que pasa a ser un atributo.

El **CAMICAI** es un **controlador** formado por un **Conjunto de Dispositivos Digitales (CDD)** interconectados de una manera especial para *obedecer* a las *instrucciones contenidas* en un **Elemento de Memoria Externo(EME)**, *generadas* por un **Sintetizador de la Carta ASM (SCASM)**. El **CAMICAI** *demuestra* físicamente la *solución*.

OBJETO	ESPACIO	OPERACIÓN	ATRIBUTO
$CAMICAI_1$	Problema	Demuestra la solución	Controlador
CDD_5	Solución	Obedece Instrucciones	Interconectar
EME_6	Solución	Contiene Instrucciones	
$SCASM_7$	Solución	Genera Instrucciones	

La frase verbal "*obedecer a las instrucciones*" comparte su objeto directo con "*instrucciones contenidas*" e indirectamente con "*... generadas*". *Instrucciones* no puede considerarse un objeto ya que en el enunciado "*instrucciones contenidas ...*" son enlistadas como operación de un posible objeto relacionado con la memoria física.

El **CDD** es un módulo electrónico físico *disponible* para su uso en la construcción de módulos más complejos.

OBJETO	ESPACIO	OPERACIÓN	ATRIBUTO
CDD_5	Problema	Disponible para uso y construcción	

De la frase verbal "*disponible para uso y construcción*" podría derivarse otro objeto "**módulos más complejos**" o podría tomarse como una frase verbal íntegra perteneciente al **CDD**.

Puesto que el **CDD** son elementos del mundo real que en este caso particular no necesitan concretarse (los dispositivos digitales se pueden considerar cajas negras) dejaremos hasta aquí el método para este objeto.

El **EME** es un elemento perteneciente al CDD encargado de *almacenar información*.

OBJETO	ESPACIO	OPERACIÓN	ATRIBUTO
EME_6	Problema	Almacena información	Pertenece al CDD_5.

El **EME** se halla contenido dentro del **CDD** que, como ya se dijo son elementos físicos que se comportan como unidades (cajas negras).

El **$SCASM_7$** es una **entidad** encargada de la *validación* de la estructura *de la Carta ASM*, la *interpretación de la misma* y la *generación del microprograma* de acuerdo a la **metodología de diseño MICA I**.

OBJETO	ESPACIO	OPERACIÓN	ATRIBUTO
$SCASM_7$	Problema	Valida e Interpreta la carta, genera microprograma	Entidad

La metodología de diseño MICA-I ha sido ya descrita en capítulos anteriores y puede ser considerada como un atributo del **SCASM**, que es el posible objeto donde se refleja todo el proceso de síntesis. Nótese como la metodología es mencionada como punto de referencia para las operaciones del **SCASM**. La metodología es un conjunto de pasos y técnicas que, vistos como una lista de funciones pueden ser insertados como parte de las operaciones del **SCASM**.

El **CDD** es un módulo electrónico físico *disponible* para su uso en la construcción de módulos más complejos.

OBJETO	ESPACIO	OPERACIÓN	ATRIBUTO
CDD_5	Problema	Disponible para uso y construcción	

De la frase verbal "*disponible para uso y construcción*" podría derivarse otro objeto "**módulos más complejos**" o podría tomarse como una frase verbal íntegra perteneciente al **CDD**.

Puesto que el **CDD** son elementos del mundo real que en este caso particular no necesitan concretarse (los dispositivos digitales se pueden considerar cajas negras) dejaremos hasta aquí el método para este objeto.

El **EME** es un elemento perteneciente al CDD encargado de *almacenar información*.

OBJETO	ESPACIO	OPERACIÓN	ATRIBUTO
EME_6	Problema	Almacena información	Pertenece al CDD_5.

El **EME** se halla contenido dentro del **CDD** que, como ya se dijo son elementos físicos que se comportan como unidades (cajas negras).

El **SCASM**$_7$ es una **entidad** encargada de la *validación* de la estructura *de la Carta ASM*, la *interpretación de la misma* y la *generación del microprograma* de acuerdo a la **metodología de diseño MICA I**.

OBJETO	ESPACIO	OPERACIÓN	ATRIBUTO
$SCASM_7$	Problema	Valida e Interpreta la carta, genera microprograma	Entidad

La metodología de diseño MICA I ha sido ya descrita en capítulos anteriores y puede ser considerada como un atributo del **SCASM**, que es el posible objeto donde se refleja todo el proceso de síntesis. Nótese como la metodología es mencionada como punto de referencia para las operaciones del **SCASM**. La metodología es un conjunto de pasos y técnicas que, vistos como una lista de funciones pueden ser insertados como parte de las operaciones del **SCASM**.

IV.8 Comentarios al desarrollo del LDP

OBJETO - OPERACIÓN	COMENTARIO
Objeto: EGS_1	**** Considerado como sinónimo del sistema solución, relaciona al CEG con el Hardware. Se construirá un objeto denominado **Unidad Gráfica** encargado de Inicializar el Ambiente Gráfico en la computadora y de conmutar al hardware a ese modo.
Operación: Permite Edición	**** Operación trasladada a **Área de Trabajo** y a **Carta ASM**.
Objeto: **Carta ASM**$_2$	**** Construido como un objeto independiente de **Dispositivo**, pues no pertenece al EGS.
Operación: Permite Edición	**** Operación trasladada a **Área de trabajo**.
Operación: Ensambla	**** Construida en forma de funciones para el manejo de una lista.
Objeto: $Usuario_3$	**** Abstracción que no necesita ser construida, pero podría ser representada por el Apuntador y el teclado alfanumérico.
Operación: Maneja Elemento	**** Operación trasladada a **Área de Trabajo**.
Operación: Construir	**** Operación trasladada a **Área de Trabajo**.
Operación: Ensamblar	**** Operación trasladada a **Cartas ASM**.
Operación: Indicar Secuencia	**** Operación trasladada a **Área de Trabajo**.
Objeto: **Área de Trabajo**$_4$	**** Construido como objeto dependiente de **Dispositivo**.
Operación: Visualizar **Elementos**	**** Construido como una operación inherente de **Área de Trabajo**.
Operación: Mostrar	**** Sinónimo de la operación anterior.
Objeto: EDE_5	**** Conceptualiza un grupo de elementos exclusivos para la edición, sin construir.
Operación: Edición	**** Operación identificada dentro de **Área de Trabajo**.
Objeto: CEG_6	**** Conceptualiza al total de los elementos exclusivos del Entorno Gráfico (**Dispositivo**).
Operación: Permite Manipular Organizar	**** Construida independientemente por cada objeto derivado de **Dispositivo** e Interpretar
Objeto: $Superficie_8$	**** Se considera un atributo de **Área de Trabajo**.

Operación: Delimitar	**** Pasa a ser un límite intrínseco a una función de **Área de Trabajo**.
Objeto: Lista Oper9	**** Agrupa objetos que administran al sistema (**Área de Trabajo**, MDT, etc.), Construido como funciones independientes en distintos objetos.
Operación: Administrar, Grabar, Recupera	**** Operaciones Trasladadas a sus respectivos objetos.
Objeto: **Elemento** ASM10	**** Construido como objeto derivado de dispositivo que concentra las características comunes a los distintos elementos.
Operación: Seleccionar, Construir	**** Cada **Elemento** ASM se representa a sí mismo, la selección la opera el objeto **Carta ASM** y la Construcción es dirigida **por Área de Trabajo** y almacenada por **Carta ASM**.
Objeto: **Dispositivos**11	**** Objeto fundamental para concentrar características de los objetos que componen al EGS.
Operación: Controla Forma	**** Atributo de cada uno de los dispositivos, no construida.
Operación: Interactuar	**** Atributo de cada uno de los dispositivos, no construida.
Operación: Modificar Escala	**** Construida como una función ligada al botón de Zoom.
Operación: Desplazar	**** Construida como un nuevo objeto de **Barra de Desplazamiento**.
Objeto: Pantalla12	**** Se considera sinónimo de **Área de Trabajo** y su operación se transfiere a éste.
Operación: Mostrar	**** Atributo asignado a **Área de Trabajo**.
Objeto: **Ventana**13	**** Objeto que agrupa a otros tipos de ventana hereda de **Dispositivo**.
Operación: Mostrar MDO	**** Construida.
Operación: Desplegar	**** Construida.
Objeto: Apuntador14	**** Objeto complementario de EGS que se construye con nombre "**Ratón**" tanto para el modo texto como para el modo gráfico.
Operación: Selecciona **Dispositivos**	**** Función construida como parte del flujo principal del programa (main) basado en una operación de **Ratón** que obtiene la posición actual.
Objeto: Cursor Texto15	**** En el modo gráfico el cursor texto no existe, pero puede ser manejado por funciones.

Operación: Introducir Datos	**** Construida como una función que coloca un cursor y espera información por teclado.
Operación: Dirigir Acciones	**** Considerada atributo.
Objeto: Algoritmo[16]	**** Una abstracción de una función o de un procedimiento sin necesidad de implantarse.
Operación: Ejecutar	**** Atributo de cualquier función, eliminada.
Objeto: Parámetros[17]	**** Pueden incorporarse al sistema como parte del objeto de menú o como objeto en sí.
Operación: Cambiar	**** Construida como proceso implícito del menú.
Objeto: Menú Desplegable de Texto[18]	**** No necesariamente un objeto, si se organiza de manera adecuada puede ser una extensión funcional del **Área de Trabajo**.
Operación: Agrupar	**** Considerado atributo, es una consecuencia de utilizar un menú.
Operación: Mostrar	**** Construida como una función que hace uso de **Ventana**.
Objeto: Menú de Iconos (MDI)[19]	**** Dado que los iconos pueden crearse con **Botones**, el menú es la designación a la función que hace aparecer a dichos objetos.
Operación: Agrupar	**** Considerada atributo.
Operación: Mostrar	**** Operación trasladada al flujo principal del programa por su escaso uso posterior.
Objeto: **Estado**[20]	**** Objeto derivado de **Elemento** ASM.
Operación: Presentar	**** Construida localmente como "Dibuja".
Operación: Seleccionar	**** Construida en **Elemento** ASM.
Operación: Mover	**** Construida originalmente en **Elemento** ASM, fue trasladada a **Dispositivo**.
Operación: Modificar	**** Construida.
Objeto: **Salida Condicional** [21]	**** Objeto derivado de **Elemento** ASM.
Operación: Presentar	**** Construida locamente como "Dibuja".
Operación: Seleccionar	**** Construida en **Elemento** ASM en Coordinación de **Área de Trabajo**.

Operación: Mover	**** Construida originalmente en **Elemento** ASM pero movida a **Dispositivo** como algo más general aplicable a otros objetos.
Operación: Modificar	**** Construida.
Objeto: **Decisión**[22]	**** Objeto derivado de **Elemento** ASM.
Operación: Presentar	**** Construida.
Operación: Seleccionar	**** Construida en **Elemento** ASM en coordinación de **Área de Trabajo**.
Operación: Mover	**** Construida en **Dispositivo**.
Operación: Modificar	**** Construida.
Objeto: Valores[23]	**** Objeto omitido por hacer referencia a valores que por su cantidad reducida serán mantenidos implícitamente en el sistema.
Operación: Determina Características	**** Considerada atributo de los valores mantenidos dentro del sistema.
Objeto: Submenús[25]	**** Es un **Elemento** constituyente del MDT que puede desarrollarse como funciones.
Operación: Mostrar **Ventana**	**** Construida.
Operación: Seleccionar	**** Construida.
Operación: Emerger Verticalmente	**** Operación distinta a las de **Ventana**, puesto que sobre ella se elegirá un Texto.
Operación: Mostrar Contenido	**** Operación común a todas las **Ventanas** "Implantada".
Objeto: **Ventana Dialogo** [26]	**** Como **Ventana**, pero con **Botones** de interacción (OK, Cancelar, etc.).
Operación: Capturar Información	**** Operación que podría hacer uso del Cursor de Texto.
Objeto: Opciones[27]	**** Elementos de los Submenús, pueden ser conjunto de apuntadores a funciones.
Operación: Seleccionar	**** Operación instrumentada como procedimiento en el MDT.
Objeto: **Botones**[28]	**** Objeto dependiente de **Punto** para dibujarse que "responde" a su selección.

Operación: Seleccionar **** Operación que aparentará el presionado visual del **Botón**.

Objeto: Operación[29] **** Otro sinónimo de una función o procedimiento, esta vez asignado a un **Botón**.

Operación: Ejecutar **** Atributo del objeto **Botón**.

Objeto: **Punto**[30] **** La Base de cualquier elemento gráfico, puesto que todas las imágenes están formadas por puntos, es la clase inicial para el dibujado en pantalla.

Operación: Seleccionar **** Construida.

Objeto **ECSE**[1] **** Abstracción de un objeto más concreto

Operación: Aprovecha Facilidades

Operaciones derivadas:

Obtiene un controlador completo con arquitectura MICA I

- Soluciona el problema

Objeto **Carta ASM**[2] **** Objeto ya construido dentro del sistema de edición

Objeto **SE**[3] **** Referencía a un objeto parte de la solución, el sistema editor de **Cartas ASM**

Operación: Otorga Facilidades

Objeto: **CAMICAI**[4] **** Se construye como un objeto que mostrará todos los elementos del objeto real, renombrado como objeto **Síntesis**

Operación: Demuestra fís. Solución

Operaciones derivadas:

Muestra el diagrama de la arquitectura MICA I*

- Obtiene información del diseño de la carta.
- Dibuja el diagrama actualizado de la arquitectura.
- Muestra la tabla de funcionamiento del contador*

Objeto **CDD**[5] **** Una abstracción del CAMICAI, es casi un sinónimo, y por lo tanto no es construido.

Objeto **EME**$_6$ **** Es una abstracción de un elemento del CAMICAI, no construido.

Objeto: **SCASM**$_7$ **** Construido como objeto independiente instanciado en Área de Trabajo. Que unificado con CAMICAI forma el objeto **Síntesis.**

Operación: Valida la **Carta ASM**. **** Verifica la estructura de la carta contra las reglas de diseño de la arquitectura Mica I.

Operación: Interpretación tabular de la **Carta ASM**

Operaciones derivadas:

Muestra tabla del contador*

- Obtiene información del diseño del contador y construye la tabla en memoria.

Muestra tabla de entradas y salidas*

- Obtiene información del Objeto **Carta ASM** para llenar la tabla con la información de entradas/salidas.

Muestra tabla de estados de la **Carta ASM***

- Obtiene información del diseño de la carta y construye la tabla de estados en memoria.

Muestra las operaciones por estado*.

- Obtiene información del diseño de la carta, Llena la tabla de operaciones por estado.

Muestra tabla de instrucciones de MICA I*.

- Genera la tabla de instrucciones a partir de información previamente almacenada en forma de texto.

Operación: Generación del microprograma

Operaciones derivadas:

Muestra la tabla de microprograma en código binario*.

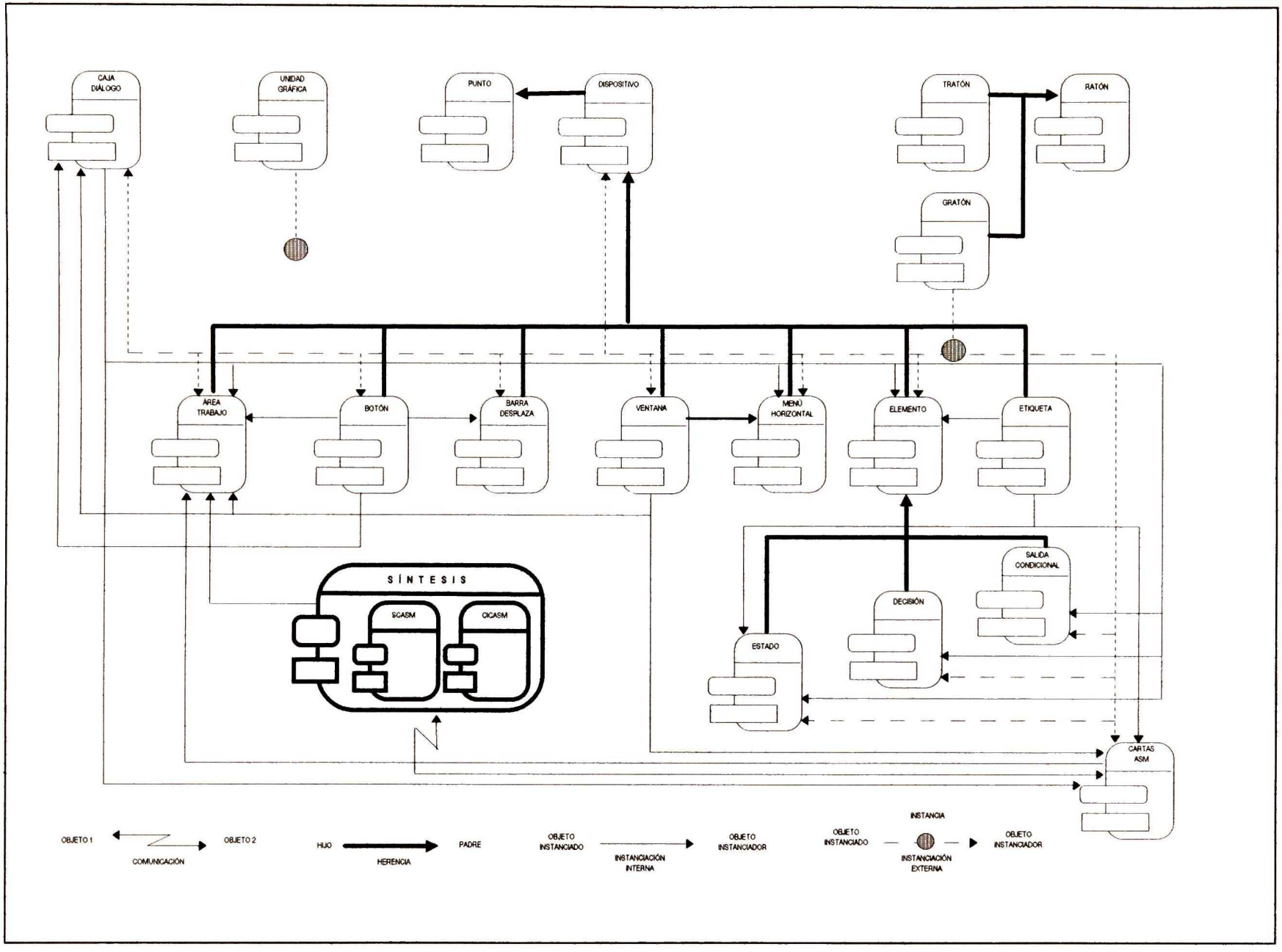
- Obtiene la información de las tablas y la relaciona para generar el microprograma.

- Genera el microprograma y llena una tabla con la información en código binario.

Muestra tabla de microprograma en código hexadecimal*.

- Toma la tabla del microprograma y la traduce a código cexadecimal.

* Las Operaciones que muestran imágenes presentan también una operación equivalente para la impresión en papel.


CAJA DIÁLOGO
UNIDAD GRÁFICA
PUNTO
DISPOSITIVO
TRATÓN
RATÓN
GRATÓN
ÁREA TRABAJO
BOTÓN
BARRA DESPLAZA
VENTANA
MENÚ HORIZONTAL
ELEMENTO
ETIQUETA
SÍNTESIS
SCASM
CICASM
SALIDA CONDICIONAL
DECISIÓN
ESTADO
CARTAS ASM
OBJETO 1
COMUNICACIÓN
OBJETO 2
HIJO
HERENCIA
PADRE
OBJETO INSTANCIADO
INSTANCIACIÓN INTERNA
OBJETO INSTANCIADOR
INSTANCIA
OBJETO INSTANCIADO
INSTANCIACIÓN EXTERNA
OBJETO INSTANCIADOR

CONCLUSIONES

Conclusiones

El aprovechamiento de la tecnología en la construcción de herramientas que a su vez ayudan al desarrollo de nuevas tecnologías, es una característica especialmente notable en el software de computadoras y que se demuestra con este trabajo, donde se incorpora a un sistema de edición gráfica la capacidad de la síntesis del diseño electrónico digital.

De hecho, el cumplimiento del objetivo principal referente a crear una herramienta como la mencionada, se ha logrado reutilizando el diseño y el código originado en el desarrollo del editor gráfico perteneciente al mismo proyecto. La disponibilidad de un entorno gráfico afín a las expectativas del proyecto, permite plantearlo como una extensión más que como una modificación.

La reutilización de un sistema cualquiera reducirá el tiempo y esfuerzo de desarrollo en la medida en que el sistema original sea estructurado de una forma en la que dicha tarea se facilite. En este caso, la orientación a objetos conformó un diseño fácilmente extendible y que además aisló al nuevo desarrollo del sistema original, con lo que se cumplió también con el subobjetivo 2. Adicionalmente, la continuidad del proyecto exigía el uso de los métodos de orientación a objetos que, aplicados al diseño previo del editor, producirían un desarrollo homogéneo metodológicamente hablando, con lo que se cumpliría así con el subobjetivo 5 y de paso facilitaría la tarea de documentación y diseño de los nuevos elementos.

Para alcanzar el subobjetivo 3, y lograr la síntesis mediante MICA I, se procedió al análisis e interpretación de los principios del diseño con Cartas ASM y a conjugarlos con la generalidad de las metodologías de diseño de microcontroladores. Fué así como primeramente se vió cumplido el subobjetivo 1, formando un marco estructural en la aplicación que permitiera posteriormente la particularización del diseño sobre el método llamado MICA I. Es importante señalar que la estructura obtenida por el diseño, permite el acoplamiento de nuevos módulos que tendrán como único requisito obedecer a las especificaciones del sistema, enunciadas por la definición de clases del Apéndice E.

El sistema desarrollado, permite la posterior incorporación de módulos que lleven a cabo la síntesis de los controladores utilizando nuevas arquitecturas de diseño microprogramado, como podrían ser: el diseño con MICA II, con bit slice o con dispositivos PLD´s; esto le da una amplia potencialidad en crecimiento.

La síntesis computarizada de controladores digitales, como todo desarrollo encaminado a reducir el tiempo-hombre dedicado a tareas repetitivas, debe mantener una comunicación constante con el usuario con el fin de determinar si lo que se pensó originalmente es aquello que le está siendo descrito al sistema, esto es, retroalimentar al usuario; pero también se requiere comunicar el final de un requerimiento, de mostrar ese resultado y, eventualmente, el proceso que conduce al mismo. Para tal efecto se incorporaron al desarrollo los elementos necesarios para una visualización e impresión del resultado del proceso de síntesis.

Cabe señalar que la impresión de resultados se llevó a cabo utilizando el estándar Epson, pero que existen otros más que podrían utilizarse y que en un momento dado se harían necesarios, ya que le darían al sistema una mayor versatilidad al poder imprimir en dispositivos que no manejan este estándar. Al sistema se le pueden añadir otros manejadores de impresión y presentarlos en un menú del cual el usuario puede elegir para llevar a cabo una correcta impresión, en dispositivos que no manejen el estándar Epson.

Como se puede ver al final del trabajo, la arquitectura con que se desarrolla la síntesis, utiliza una memoria para el almacenamiento del programa que gobierna al controlador; como el sistema genera este programa, sería posible hacer una conexión con un dispositivo programador de memorias y hacer una grabación más directa de la información, quitando así la necesidad de introducir manualmente el programa dentro del dispositivo grabador, proceso que ocasionalmente podría generar errores.

Una forma de enriquecer el sistema realizado, puede surgir de la creación de una interface para poder aprovechar librerias de circuitos ya existentes como los de OrCad, si podemos manipular sus librerias tendriamos una gran diversidad de circuitos tanto digitales como analógicos que nos darian gran versatilidad para manipular no sólo circuitos digitales.

En el diseño de controladores digitales, después de realizado éste, se hace necesario construir un prototipo físico para verificar y demostrar que el diseño es correcto y que el controlador funciona como lo planeamos; si después de hacer esto nos damos cuenta de que no todo salió como se planeó, habremos perdido tiempo, y muy probablemente recursos; este problema puede ser resuelto si tenemos la posibilidad de hacer las pruebas sin tener que construir físicamente el controlador. Considerando que dentro del sistema se tienen todas las características tanto físicas como de funcionamiento del controlador, se pueden construir módulos que aprovechen esta información y realicen la simulación del funcionamiento del circuito, y así cuando estemos totalmente convencidos del funcionamiento del controlador, procedemos a construirlo físicamente.

En la definición de las arquitecturas de controladores digitales, se tienen definidas operaciones que derivan de las características de los circuito empleados, estas operaciones disponibles rigen las posibilidades de diseño de las Cartas ASM que describen el funcionamiento del controlador; teniendo bien definidas y presentes estas operaciones, se antoja factible la definición del funcionamiento del controlador sin tener que construir una Carta ASM, se tendría con esto un lenguaje para el desarrollo de controladores digitales cuyas instrucciones tendrían su origen directamente en las características de funcionamiento de los circuitos digitales con que se desarrolla.

Si consideramos este trabajo como una demostración de la extensión de un sistema para la consecución del objetivo general, podríamos definir al conjunto de facilidades de desarrollo otorgadas por el entorno gráfico, como una plataforma sobre la cual es posible diseñar y estructurar sistemas específicos que involucren la edición de objetos. Este tipo de sistemas permite al usuario construir rápidamente especializaciones concretas a partir de una base general, por lo que es altamente recomendable su uso en el desarrollo acelerado de aplicaciones.

De acuerdo a la situación actual del mercado de desarrollo de sistemas para computadoras personales, podemos decir que el presente trabajo encaja en la tendencia prevalente de generación de aplicaciones que incorporan a su diseño los nuevos métodos disponibles que apuntan hacia un mejor aprovechamiento de los recursos humanos y de cómputo. Sería deseable la planeación de nuevos proyectos que extiendan el presente desarrollo explotando la potencialidad que representa.

El sistema esta desarrollado bajo el popular ambiente DOS usando el compilador de C++, pero como una adaptación seria deseable hacer un desarrollo para su manejo en otras plataformas tanto de software como de hardware, como pueden ser:

Hardware:	Software:
HP-9000	Windows
Silicon Graphics	Unix
Pentium	NextStep
Next	Mac
etc.	etc.

Esperamos que el presente trabajo contribuya al aprendizaje y desarrollo tanto de sistemas digitales como computacionales, el primero dentro del área de controladores digitales y el segundo con la filosofía de la OO. Se proporciona tanto el producto en sí como los fuentes de la aplicación. Esperando que sea de utilidad.

APÉNDICE A

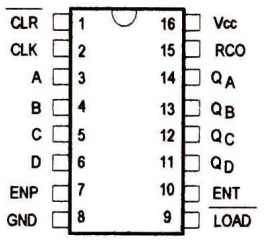
DIAGRAMA DE ALGUNOS CIRCUITOS TTL

APÉNDICE A

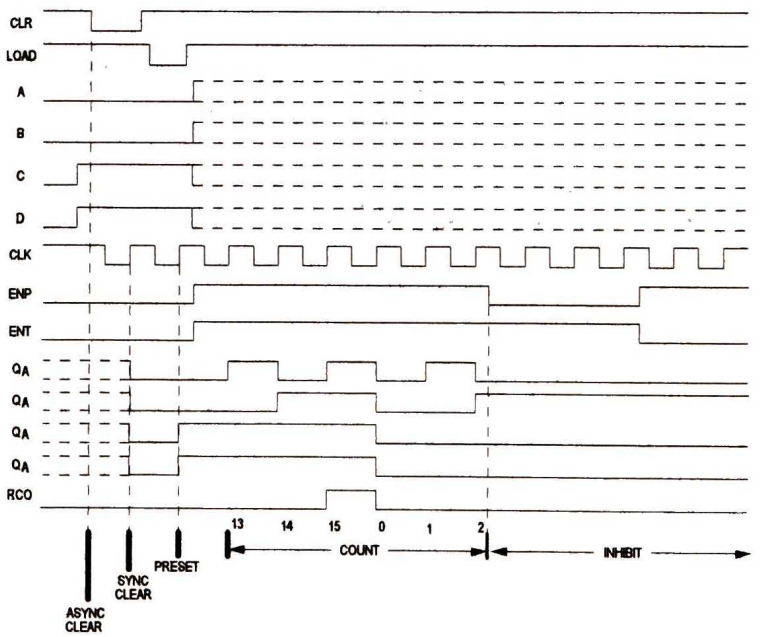
DIAGRAMA DE ALGUNOS CIRCUITOS TTL

En este apéndice se muestra información básica de los circuitos TTL que pueden ser empleados para implementar los modulos mostrados en el Capítulo de "Arquitecturas de Diseño". Si se requiere más información a cerca de éstos, consulte el manual correspondiente.

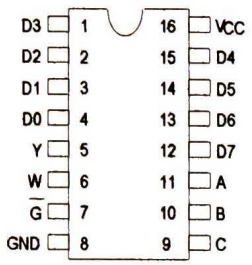
•Circuito SN74F161. Contador binario de 4 bits.



ENP	ENT	LOAD	
1	1	1	COUNT
0	0	0	PRESET
0	1	1	INHIBIT
1	0	1	INHIBIT

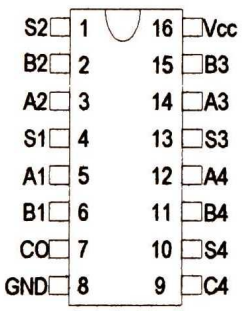


•Circuito SN74151. Multiplexor 8:1.



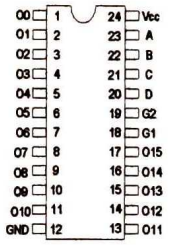
INPUTS				OUTPUTS	
SELECT			STROBE		
C	B	A	$\overline{G}$	Y	W
X	X	X	H	L	H
L	L	L	L	D0	$\overline{D0}$
L	L	H	L	D1	$\overline{D1}$
L	H	L	L	D2	$\overline{D2}$
L	H	H	L	D3	$\overline{D3}$
H	L	L	L	D4	$\overline{D4}$
H	L	H	L	D5	$\overline{D5}$
H	H	L	L	D6	$\overline{D6}$
H	H	H	L	D7	$\overline{D7}$

•Circuito 74283. Sumador binario de 4 bits.



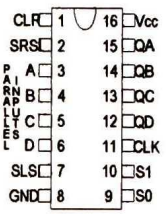
INPUTS				OUTPUTS					
				WHEN C0=L / WHEN C2=L			WHEN C0=H / WHEN C2=H		
A1/A3	B1/B3	A2/A4	B2/B4	S1/S3	S2/S4	C2/C4	S1/S3	S2/S4	C2/C4
L	L	L	L	L	L	L	H	L	L
H	L	L	L	H	L	L	L	H	L
L	H	L	L	H	L	L	L	H	L
H	H	L	L	L	H	L	H	H	L
L	L	H	L	L	H	L	H	H	L
H	L	H	L	H	H	L	L	L	H
L	H	H	L	H	H	L	L	L	H
H	H	H	L	L	L	H	H	L	H
L	L	L	H	L	H	L	H	H	L
H	L	L	H	H	H	L	L	L	H
L	H	L	H	H	H	L	L	L	H
H	H	L	H	L	L	H	H	L	H
L	L	H	H	L	L	H	H	L	H
H	L	H	H	H	L	H	L	H	H
L	H	H	H	H	L	H	L	H	H
H	H	H	H	L	H	H	H	H	H

•Circuito 74154. Decodificador 1 de 16.



INPUTS						OUTPUTS															
G1	G2	D	C	B	A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
L	L	L	L	L	L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	H	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	H	L	L	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H
L	L	L	H	L	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H
L	L	L	H	H	L	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H
L	L	L	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H
L	L	H	L	L	L	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H	H
L	L	H	L	L	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H
L	L	H	L	H	L	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H
L	L	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H
L	L	H	H	L	L	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H
L	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H
L	L	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H
L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L
L	H	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
H	L	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
H	H	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H

•Circuito 74194. Registro de corrimiento universal bidireccional de 4 bits.



INPUTS										OUTPUTS			
	MODE			SERIAL		PARALLEL							
CLR	S1	S0	CLK	LEFT	RIGHT	A	B	C	D	QA	QB	QC	QD
L	*	*	*	*	*	*	*	*	*	L	L	L	L
H	*	*	L	*	*	*	*	*	*	Q_{A0}	Q_{B0}	Q_{C0}	Q_{D0}
H	H	H	↑	*	*	a	b	c	d	a	b	c	d
H	L	H	↑	*	H	*	*	*	*	H	Q_{An}	Q_{Bn}	Q_{Cn}
H	L	H	↑	*	L	*	*	*	*	L	Q_{An}	Q_{Bn}	Q_{Cn}
H	H	L	↑	H	*	*	*	*	*	Q_{Bn}	Q_{Cn}	Q_{Dn}	H
H	H	L	↑	L	*	*	*	*	*	Q_{Bn}	Q_{Cn}	Q_{Dn}	L
H	L	L	*	*	*	*	*	*	*	Q_{A0}	Q_{B0}	Q_{C0}	Q_{D0}

APÉNDICE B

MANUAL DE USUARIO

APÉNDICE B

MANUAL DE USUARIO

Bienvenida

Bienvenido al **Diseñador Gráfico de Microcontroladores AuXiliar o Sistema de Apoyo al Diseño Digital con Arquitectura Microprogramada (DGMAX/SADDAM)**, el primer sistema de procesamiento de Cartas ASM por computadora realizado en México. El DGMAX/SADDAM incluye muchas características del estándar para interfaces gráficas de usuario (GUI's por sus siglas en inglés), que transformará el trabajo de construir el diagrama de la Carta ASM junto con su síntesis en papel, a archivos de computadora para su posterior procesamiento. Desarrollado con la tecnología de la Orientación a Objetos, el DGMAX/SADDAM liberará al diseñador de controladores digitales -que usa la metodología de Cartas ASM-, de la laboriosa tarea de sintetizar el problema en una serie de tablas, microcódigo y diagrama electrónico del circuito -posterior a alambrarse-, a una pequeña serie de "tikis" (palabra que se emplea en sustitución del tecnicismo inglés "click") del dispositivo apuntador denominado ratón, con la finalidad de obtener los mismos resultados aunque en forma más eficiente.

¿A quiénes está dirigido DGMAX/SADDAM?. DGMAX/SADDAM está dirigido a todas aquellas personas que alguna vez diseñen controladores digitales empleando la metodología de Cartas ASM y síntesis de la misma con la arquitectura MICA I, además a todas aquellas interesadas en el desarrollo de sistemas de computadoras en ambiente gráfico y con la metodología de la Orientación a Objetos.

Para ambos perfiles de usuarios está disponible el sistema operando en forma correcta, para el segundo perfil están disponibles además, los fuentes de la aplicación, así como el diseño del sistema en el capítulo correspondiente a su desarrollo. Se hace del conocimiento del usuario que se cuenta con la libertad de desarrollar otras aplicaciones usando la mayoría de las componentes del sistema, como es la interface gráfica de usuario, teniendo la libertad inclusive de modificar alguna de dichas componentes. Bueno, como lo que realmente importa es saber como podemos trabajar con DGMAX/SADDAM, como entrar y como salir de la aplicación, las tres secciones siguientes están dedicadas a ello, así que ¡manos a la obra!.

Instalación del sistema

En primer lugar, Usted debe contar con el DGMAX/SADDAM, proporcionado en un disquete en formato de 3½" de alta densidad (1.44 MB) como material del presente trabajo. El cual contiene los siguientes archivos:

- **Manual.txt**. Contiene -en código ASCII- la información del presente documento relativo a la instalación y manejo de la aplicación.
- **Fuentes.exe**. Desempaqueta los archivos fuente de la aplicación.
- **Sistema.exe**. Desempaqueta los archivos del sistema de la aplicación.
- **Instalar.bat**. Permite instalar los archivos de sistema y/o los archivos fuentes -los archivos fuentes no son necesarios para el proceso de sesión de trabajo con el sistema- de la aplicación en el disco duro.

Continuando, se debe determinar la unidad de disco en el cual se instalará el sistema. Una vez elegida la unidad de instalación, se está en posibilidad de ejecutar el archivo de instalación **instalar.bat**. En el proceso de instalación sólo se desempaquetan los archivos del sistema, los correspondientes a los archivos fuentes de la aplicación se dejan empaquetado (**Fuentes.exe**), esto con la finalidad de ahorrar espacio en disco, aunque se cuenta con la opción de desempaquetarlo también. En principio de cuenta veremos como instalar los archivos de sistema:

Pasos de la instalación:

1. Ejecutar el archivo **instalar.bat** con la siguiente sintaxis:

   instalar UDisco [Enter]

   donde:

   UDisco. Es la unidad de disco seleccionada (c, d, ...).

Comentario: La unidad de disco sólo se representa con la letra indicada para ella y seguida de : (dos puntos). Lo que está encerrado en corchetes indica que se debe presionar la tecla etiquetada con la palabra "Enter"

El programa instalar.bat le preguntará si desea desempaquetar el código del sistema, si rechaza esta opción, el código fuente del sistema se grabará en UDisco, en el directorio "\sistema\fuentes", empaquetado como fuentes.exe.

Al terminar de ejecutarse el programa de instalación, Ud. tendrá grabado en el disco de la unidad seleccionada, la siguiente estructura de directorios con sus respectivos archivos:

```
UDisco:\
└─sistema
   ├─ fuentes
   └─ datos
```

Si ningún error ocurrió, estará en posibilidad de manejar la aplicación, por lo que puede ir directamente a las secciones de "¿Cómo entrar al sistema?" y "¿Cómo salir del sistema?". En caso de error, realice de nuevo el proceso de instalación como se indicó.

En el directorio sistema, estarán disponibles los siguiente archivos: asm_mica.exe (archivo ejecutable del sistema), archivos con extensión chr y bgi. Inicialmente el subdirectorio **datos** se encontrará vacío, este subdirectorio está reservado para almacenar los diferentes archivos generados en el proceso de edición de una Carta ASM, es decir, cada vez que usted cree una nueva Carta ASM o modifique una existente, al salvarla, estos archivos se almacenarán ahí. En el subdirectorio fuentes estará un archivo llamado Fuentes.exe, el cual puede ser ejecutado para desempaquetar los fuentes de la aplicación, esto es útil, siempre y cuando se desee conocer el código de alguno o de todos los archivos fuentes desarrollados -recomendado para los programadores-.

2. El procedimiento para desempaquetar los archivos fuentes es el siguiente (opcional):

- ✓ Ubicarse en el subdirectorio **UDisco:>\Sistema\Fuentes** y teclear
- ✓ fuentes [Enter]

Al final de este procedimiento veremos en el directorio "Fuentes", archivos con extensión i, h y cpp, con lo que termina el procedimiento adicional de instalación.

¿Cómo entrar al sistema?

Una vez realizada la fase de instalación, Ud. está listo para trabajar con el sistema, la forma de hacerlo es la siguiente:

Si está ubicado en un directorio diferente al directorio llamado **sistema**, debe teclear lo siguiente:

```
cd\ [Enter]
cd sistema [Enter]
asm_mica [Enter]
```

En caso de que halla estado en el directorio sistema, simplemente teclee la última instrucción.

Una vez hecho lo anterior, si no hubo error en el proceso, estará listo para trabajar con el sistema, esto lo confirmamos porque se presentará una interface como la que se muestra en la Fig. B.1. En caso de error simplemente corregirlo volviendo a teclear todo a la parte donde se cometió el error.

Sugerencia: Puede crear un archivo .bat ("punto bat") incluyendo las instrucciones anteriores para poder ahorrarse el escribirlas cada vez que desee ejecutar el sistema, además de poder iniciar desde cualquier directorio. Para mayor explicación de los archivos .bat refiérase a su manual de usuario de DOS.

¿Cómo salir del sistema?

A reserva de explicar con mayor profundidad los elementos de la interface gráfica del editor, mencionaremos que para salir del sistema, simplemente se debe apuntar con el ratón el botón  (botón con icono de una puerta abierta, una flecha y una etiqueta de salir) y presionar el botón izquierdo del ratón.

Convenciones:

- Todas las operaciones realizadas con el ratón serán por medio del botón izquierdo de éste, a menos que se especifique otro.
- Cuando se indique, "seleccionar un botón" o una "orden de la interface", significa apuntar con el ratón y proporcionar un "tiki" con el botón izquierdo de éste.

Descripción de la interface gráfica del sistema

Las Figs. B.1 a, b y c muestran los elementos más relevantes de la interface gráfica del sistema.

Elementos permanentes de la interface gráfica del sistema

A continuación presentaremos los elementos de la interface gráfica del sistema, los cuales se encuentran siempre disponibles. La Fig. B.1.a es la pantalla principal de la aplicación, estos elementos son permanentes, es decir, se conservan desde el inicio hasta el final de una sesión de trabajo.

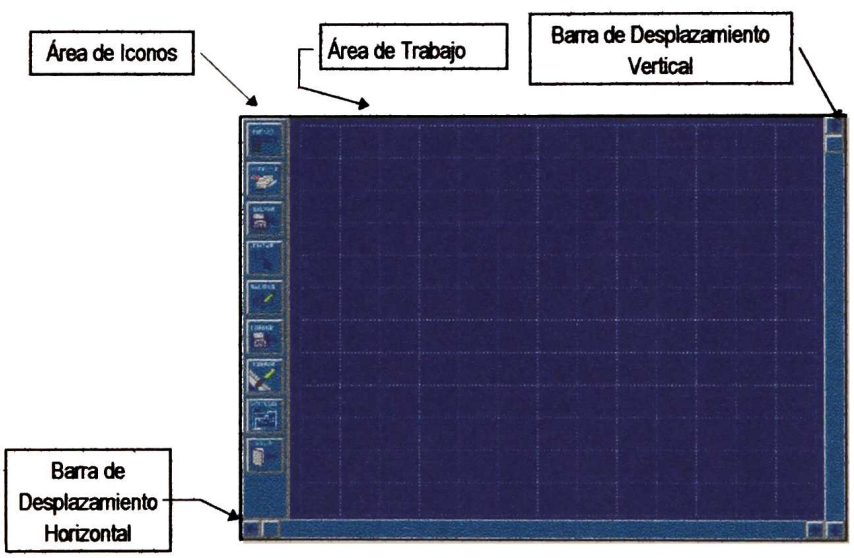


Fig. B.1.a. Descripción de los elementos permanentes de la Interface Gráfica del Sistema.

En la Fig. B.1.a observamos que la interface está dividida en varias secciones:

Área de iconos: La cual está reservada para establecer los botones con un icono, representando las operaciones más comunes del editor. Las operaciones disponibles en esta área son accesadas mediante la sencilla operación de apuntar al botón correspondiente y proporcionar un "tiki" con el botón izquierdo del ratón. Posteriormente ampliaremos la explicación de esta área.

Área de trabajo: Ésta está destinada para contener los elementos de la Carta ASM que se desee editar. Como observamos en dicha área, ésta presenta una cuadricula para la ubicación de los diversos elementos que contendrá la Carta ASM, además de presentar en ella los resultados de la síntesis de la carta en edición.

Adicional al Área de Trabajo se cuenta con un Área Virtual. El área virtual adicional es proporcionada al hacer uso de las Barra de Desplazamiento ya sea vertical u horizontal.

Dos Barras de Desplazamiento (Vertical y Horizontal): Las funciones que realizarán dichas barras, como ya se mencionó anteriormente, es la de navegar en el área virtual de trabajo, es decir, si la Carta ASM en edición se hiciese demasiado extensa para ubicarla en la porción del Área de Trabajo disponible, ésta se desplazará hacia arriba, hacia abajo, a la izquierda o la derecha dependiendo del desplazamiento deseado.

Menú de elementos de Cartas ASM

En la Fig. B.1.b se presenta otro elemento de la interface el cual es el **Menú de elementos** de Cartas ASM.

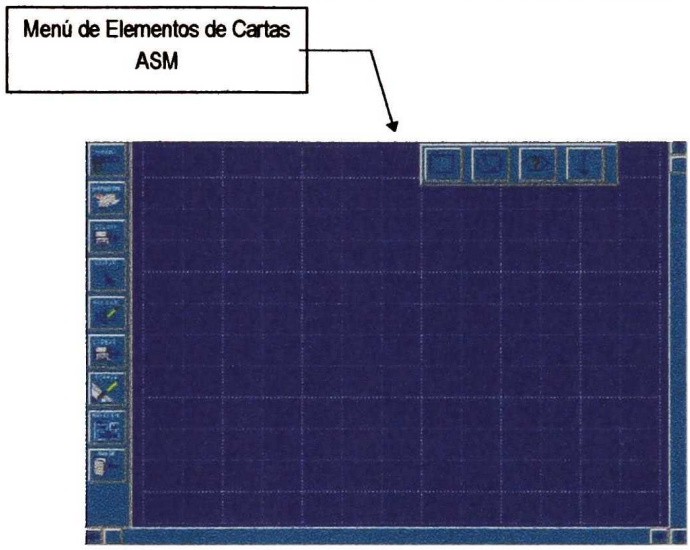


Fig. B.1.b. Descripción del Menú de Elementos de la Carta ASM.

Menú de elementos de Cartas ASM: Este menú contiene botones con iconos que representan los elementos (estado, decisión, salida condicional y unión de elementos) para construir una Carta ASM. Es invocado con un "tiki" del botón derecho del ratón, independiente de la ubicación de éste. El elemento deseado de la Carta ASM es seleccionado por su botón correspondiente (recuerde: apuntándolo y proporcionando un "tiki" al botón izquierdo del ratón). Posteriormente ampliaremos la explicación de este menú.

Descripción de los iconos del área de iconos

Icono Barra de Menú. Al seleccionar este botón, aparece en la parte superior de la pantalla la Barra de Menú para poder realiza operaciones de edición por medio de teclado o ratón. Reservado para ampliación del sistema.

Icono Impresión. Al presionar este botón, aparece una caja de diálogo para configurar las opciones disponibles de impresión de resultados tablas generadas de la metodología, microprograma, diagrama del circuito electrónico, etc. La caja de diálogo con sus respectivas opcione se presentarán más adelante.

Icono Salvar Carta ASM. Al presionar este botón, aparece una caja de diálogo preguntado si se desea salvar la carta con el mismo nombre o con uno diferente o cancelar la operación Salvar carta. La carta salvada es almacenada en el directorio **datos**. Si es una carta inicial, el sistema preguntará por el nombre a asignarle.

Icono Edición. Este botón proporciona el mecanismo para cambiar los nombres, los códigos de los estados y los nombres de las variables de entrada de la Carta ASM.

Icono Salidas. El presionar este botón, nos posibilita el introducir variables de salidas de la Carta ASM mediante una ventana de edición y su posterior ubicación en los diversos estados en los que aparezca.

Icono Cargar Carta ASM. Al presionar este botón aparece, una caja de diálogo solicitando el nombre de la Carta ASM deseada para su posterior manejo.

Icono Borrar. El presionar este botón, nos posibilita la forma de borrar un elemento de la Carta ASM, borrar la unión entre dos elementos, borrar una salida de un estado o de todos los estados o borrar la carta en su totalidad. Cuando solicitamos esta operación, el sistema presenta una serie de ventanas indicando las operaciones disponibles.

Icono Síntesis de Carta ASM. El presionar este botón, presenta los resultados siguientes en pantalla (si la Carta ASM es válida): Tabla del Conjunto de Instrucciones, Tabla del Contador Programable, Tabla de Estados, Tabla de Salidas, Microprograma en formato binario y en hexadecimal, y el diagrama electrónico representado la solución del problema.

Icono Salida. El presionar este botón nos permite salir del sistema de edición de Cartas ASM. Al hacerlo restablece las condiciones iniciales de la computadora. No verifica si las modificaciones de la carta han sido salvadas.

Nota: La operación presionar botón la realizamos haciendo un "tiki" sobre el botón izquierdo del ratón.

Descripción de las Barras de Desplazamiento

Como se observa en la Fig. B.2, cada barra de desplazamiento consta de una barra que representa la longitud disponible de edición, en los extremos de cada barra se dispone de dos botones con un icono de flecha, los cuales nos sirven para desplazar la información ubicada en el área de trabajo, ya sea hacia arriba, hacia abajo, a la izquierda o a la derecha.

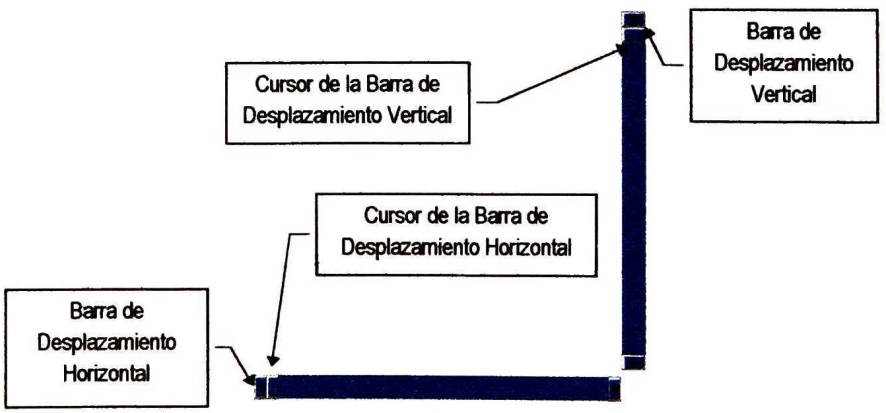


Fig. B.2. Barras de Desplazamiento: Vertical y Horizontal.

Además, observamos que consta de otro botón, el cual es el cursor de la barra y se desplazará según la indicación que le demos por medio de los botones extremos.

Descripción de los botones del menú de elementos

El Menú de elementos  está conformado por cuatro elementos con los que se representan las Cartas ASM, y que son: estado, representado por un botón con la figura de un rectángulo; salida condicional, representado por un botón con la figura de un rectángulo con esquinas redondeadas; diamante de decisión, representado por un botón con la figura de un rombo estilizado y por último la unión, representada por un botón con la figura de una flecha para la unión de los elementos. Para mayor información acerca de las características de cada elemento refiérase al Capítulo de Cartas ASM.

Icono Elemento Estado. El presionar este botón crea un nuevo Estado, con un nombre y un código por omisión. La reubicación al lugar deseado, cambio de nombre y/o código de estado y unirlo con otro elemento de la Carta ASM se realiza posteriormente.

Icono Elemento Salida Condicional. El presionar este botón crea un nuevo elemento Salida Condicional, al igual que al Estado sólo nos restaría reubicarlo en la posición adecuada, establecerle la(s) salida(s) y unirlo a los elementos deseados.

Icono Elemento Decisión. El presionar este botón crea una nueva Decisión para la Carta ASM, le asigna una variable de prueba por omisión, a la que le podemos cambiar el nombre. Al igual que a los elementos anteriores, deberemos reubicarlo y unirlo a los elementos correspondientes.

Icono Unión de Elementos. Técnicamente hablando éste no es un elemento de una Carta ASM, sin embargo aquí lo consideraremos así. El presionar este botón nos permite unir dos elemento de la Carta ASM. Más tarde ampliaremos el procedimiento que tenemos que seguir para realizar la unión.

Operación arrastrar elementos

Para arrastrar un elemento, ya sea un estado, una decisión, una salida condicional o alguna Variable de salida del sistema, se procede de la siguiente manera:

- ✓ Se apunta al elemento deseado.
- ✓ Se presiona el botón izquierdo del ratón, sin soltarlo.
- ✓ Se arrastra el elemento seleccionado hasta el lugar deseado.
- ✓ Se libera el botón del ratón.

Con esto queda completo el proceso de arrastre y el elemento arrastrado se ubica en su nueva posición.

Nota.- Si el elemento arrastrado fue una variable, ésta es ubicada en una posición, siempre y cuando sea dentro de un estado.

Operación imprimir

A continuación detallaremos las opciones de la operación de impresión. Supongamos que ya hemos presionado el botón  (botón con el icono de impresora), a lo que el sistema respondió con la siguiente caja de dialogo:

Fig. B.3. Opciones de Impresión.

Por omisión, todas las opciones presentes en la anterior caja de diálogo están activas, es decir, todas ellas se imprimirán si seleccionamos la opción de imprimir. Lo anterior lo sabemos porque se presentan los botones a la derecha de la opción con el símbolo "paloma" (√). En la caja de diálogo podemos seleccionar/deseleccionar las opciones que nos interesa imprimir, esto lo realizamos simplemente apuntando y proporcionando un "tiki" al botón correspondiente. Las opciones son las siguientes:

Tabla de MICA I. Con esta opción activa se imprime el conjunto de instrucciones correspondiente a la arquitectura MICA I.

Tablas de Resultados. Si esta opción está activa se imprimen las Tablas correspondiente al funcionamiento del contador programable, la combinación de las Tablas del Conjunto de Instrucciones con la del contador, la Tabla de estados de la Carta ASM, la Tabla de salidas, la Tabla de Entradas.

Tabla de Microprograma. Con esta opción activa se imprime el microprograma resultante en formato binario y hexadecimal.

Diagrama. Cuando esta opción está activa, se incluye en la impresión, el diagrama del circuito controlador.

Carta. Con esta opción activa se imprime el diagrama de la Carta ASM.

Nota: Antes de proceder a imprimir, asegúrese que su impresora esté configurada como Epson.

Operación edición del nombre y/o código de estado y/o variables de prueba de los elementos de decisión

Como se mencionó anteriormente, este botón  (botón con icono de cursor de ratón) proporciona el mecanismo para cambiar los nombres, los códigos de los estados y los nombres de las variables de entrada de la Carta ASM.

Esto se realiza de la siguiente manera:

I. Inmediatamente después de presionar este botón, apuntamos al nombre, al código del estado o a la variable de prueba dando un "tiki" con el botón izquierdo del ratón.
II. A continuación se presenta una pequeña ventana de edición, en la cual podemos teclear el nuevo nombre, al terminar de teclearlo se actualiza la etiqueta y eso es todo. El término de la edición puede ser mediante un [Enter] o porque el nombre de la variable excede de cuatro caracteres.

Operación edición de variables de salidas y ubicación de ellas en los estados correspondientes

Para proporcionar las salidas de cada uno de los estados de la Carta ASM, presionamos el botón  (botón con icono de lápiz), a lo que el sistema responde con la siguiente ventana de edición:

Fig. B.4. Ventana de edición de Variables.

En un inicio, esta ventana aparece vacía de variables, con una etiqueta de "NuevaVar", lo que indica que el programa está listo para recibir las salidas del sistema. En caso de que previamente se hallan proporcionado variables de salida, ésta o éstas aparecerán en la ventana de edición anterior. Los pasos para proporcionar salidas al sistema son los siguientes:

I. Con la ventana de edición abierta y posicionado en la etiqueta "NuevaVar", presionar la tecla [Enter], a continuación se presenta un pequeño cursor dentro de la ventana, en la posición de la etiqueta "NuevaVar", con lo que nos indica que podemos empezar a teclear el nombre de una salida, el término de ésta es mediante otro [Enter] o porque el nombre de la variable excede de cuatro caracteres, y así sucesivamente.
II. Una vez tecleada(s) la(s) salida(s) y dada una carta en pantalla, simplemente seleccionamos la variable correspondiente (apuntarla, presionar el botón izquierdo sin liberarlo) y la arrastramos dentro de la pantalla hasta ubicarla en el estado deseado.
III. Una vez ahí liberamos el botón del ratón y eso es todo, la variable queda relacionada con el estado elegido. Para salir de este modo apuntamos y damos un "tiki" del ratón fuera de la ventana.

Observaciones:

- ✗ Si por alguna razón seleccionamos una salida y no deseamos ubicarla en algún estado, simplemente la colocamos en cualquier parte diferente de un estado con lo que cancelamos esta operación.
- ✗ Si deseáramos eliminar una variable de salida de algún estado, ver **"Operación borrado de elementos del sistema"**.

Operación cargar carta de disco

Esta operación y la de "Salvar carta a Disco" son similares pero con sus características propias. Después de presionar el botón  (botón con el icono de un disquete y flecha apuntado hacia afuera) el sistema responde con la siguiente caja de dialogo:

Fig. B.5 Ventana de lectura de archivo.

Aquí tenemos que proporcionar el nombre de una Carta ASM previamente grabada, dicho nombre obedece a las reglas de todo archivo de DOS, es decir, el primer carácter debe ser una letra, se permiten dígitos después del primer carácter, el nombre completo no debe ser mayor de ocho caracteres, etc.

En el caso de que la carta esté bien grabada y no exista ningún error, se cargará en el Área de Trabajo para su posterior manejo, en caso contrario, nos proporcionará un mensaje de error indicando que el archivo no existe o que no se puede cargar.

Operación borrado de elementos del sistema

Después de presionar el botón  (botón con icono de goma de borrar), el sistema responde con las siguientes ventanas:

Fig. B.6. Ventanas de Borrado de Elementos.

Estas ventanas nos indican los diversos elementos que podemos borrar, de las cuales podemos deducir que podemos eliminar:

- Uniones entre elementos. Para esto el sistema presenta una barra en la parte inferior de la pantalla solicitando el origen de la unión, esto se lo proporcionamos con un "tiki" al elemento deseado, Si el elemento donde inicia la unión es un diamante de decisión la unión a borrar dependerá de si el elemento fue seleccionado en su extremo derecho o izquierdo. Esto es, si se seleccionó el extremo derecho se borrará la unión a la derecha del diamante, en caso contrario la unión izquierda será la eliminada. Para el estado o la salida condicional no importa el extremo seleccionado
- La carta completa. Simplemente presionando el botón que indica "carta".
- Una Variable de un estado. Esto lo realizamos apuntando y proporcionando un "tiki" a la variable en la ventana de variables de salida, a lo que el sistema responderá con otra ventana donde se listarán los estados en donde aparece: si queremos borrarla de alguno, seleccionamos de cual, y si queremos borrarla de todos, elegimos la opción "todos".

Adicionalmente podemos eliminar un elemento de la carta, simplemente selecionandolo.

Operación unión de dos elementos de la Carta ASM

Después de presionar el botón  (botón con el icono de una flecha), el sistema responde con lo siguiente:

Se presenta una barra en la parte inferior de la pantalla solicitando el elemento fuente de la unión, a lo cual le respondemos seleccionando alguno.

A continuación se presenta la misma barra solicitando el final del primer segmento de la unión -que puede ser el final de la unión si es que seleccionamos otro elemento-. Este procedimiento continua hasta que se selecciona el elemento destino.

En caso de que el elemento inicial haya sido una decisión:

a) Para seleccionar el elemento que va unido a la rama etiquetada con "0" (cero), estableceremos una línea vertical imaginaria dividiendo el Diamante de decisión en dos. seleccionamos la parte izquierda de este elemento (recuerde, apuntar y presionar el botón izquierdo del ratón).
b) Para seleccionar el elemento que va unido a la rama etiquetada con "1" (uno), realizamos lo mismo que el paso a) excepto que seleccionamos la parte derecha del diamante de decisión.
c) Completamos la operación siguiendo los pasos descritos para los otros elementos.

Ejemplo de edición de una Carta ASM

A continuación desarrollaremos un pequeño ejemplo demostrativo del editor, se contempla introducir estados, Decisiones, Variables de salida, cambiar nombre a los estados, a las Variables de prueba, etc. La carta que obtendremos será como la siguiente:

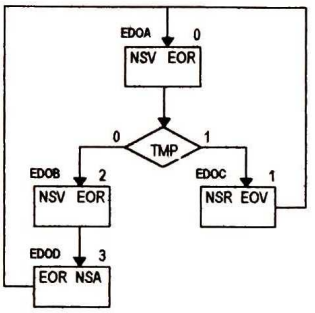


Fig. B.7.a. Carta ASM a editar.

Los pasos que realizaremos son:

1) Generar los estados, esto lo realizamos de la siguiente manera:

a) Presionar el botón derecho del ratón, a continuación aparece el menú de elementos. De este menú presionar el botón de estado. Una vez concluido lo anterior el sistema nos proporciona un estado ubicado en un recuadro vacío de la primera columna del Área de Trabajo. Observe que este estado tiene un nombre etiquetado con la letra "a", su código es el "0" (cero).
b) Hacer lo anterior para los estados restantes. Observe que cada vez que generamos un nuevo estado, éste se ubica en el cuadro inferior del anterior, además, observemos que los siguientes estados tiene como nombre la siguiente letra del alfabeto y su código es el número consecutivo del anterior.

Una vez realizado lo anterior, deben estar presentes los estados: "a", "b", "c" y "d".

2) Generar la decisión. Esto lo realizamos presionando el botón derecho del ratón, una vez que aparece el menú de elementos, presionar el botón de decisión. La decisión que se genera aparece en el recuadro inferior del último elemento creado.

Nota: El elemento que se genera, aparece en el primer recuadro libre de la primera columna, si ésta está llena, entonces aparece en el primer recuadro libre de la siguiente columna y así sucesivamente.

3) A continuación reubicaremos los elementos de tal manera que queden de la siguiente forma:

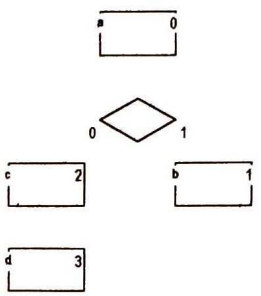


Fig. B.7.b. Carta ASM a editar.

Para lo anterior, realizaremos los siguientes pasos:

a) Apuntar al elemento que deseamos mover.
b) Presionar el botón izquierdo del ratón y sin liberarlo, ubicarse en un recuadro apropiado (sugerencia: ubíquelo en el centro del Área de Trabajo); ahora si, libere el botón del ratón.
c) Haga los dos pasos anteriores para los elementos restantes, de tal manera que queden como en la figura anterior.

4) Ahora procederemos a unir cada uno de los elementos, los pasos necesarios son:

Pasos para uniones en donde el primer elemento no es una decisión.

a) Presione el botón derecho del ratón.
b) Del menú de elementos, presione el botón que representa una flecha; en la parte inferior de la pantalla verá una barra solicitando que seleccione el elemento fuente -seleccione el estado etiquetado con la letra "a" como elemento fuente-.
c) A continuación verá la misma barra solicitándole el final del primer segmento de la unión, seleccione la decisión como elemento destino -final del segmento-, si no existió error verá una línea uniendo estos dos elementos.

Pasos para unir el elemento diamante de decisión.

a) Presione el botón derecho del ratón.
b) Del menú de elementos, presione el botón que representa una flecha, en la parte inferior de la pantalla verá una barra solicitando que seleccione el elemento inicial. Si desea unir la rama etiquetada con "0" (cero) seleccione el diamante por su extremo izquierdo, de lo contrario hágalo del extremo derecho.
c) A continuación verá la barra inferior solicitándole el final del primer segmento de la unión, seleccione la casilla ubicada a la izquierda de la decisión para definir el primer segmento de la unión izquierda -verá una línea como primer segmento-. La barra inferior solicitará el final del nuevo segmento, seleccione el estado con el código 2. Realizado lo anterior queda definida la unión de la rama izquierda. Para definir la unión derecha el procedimiento es similar. Nota. Los segmento deben ser siempre ortogonales.

Ahora proceda para cada uno de los elemento presentes uniendo:

a) El estado "a" con la decisión,
b) la decisión unida con el estado "c" por su rama "0" o izquierda,
c) la misma decisión con el estado "b" por su rama "1" o derecha,
d) el estado "b" con el estado "a",
e) el estado "c" con el estado "d" y
f) el estado "d" con el estado "a".

La carta debe quedar conformada de la siguiente manera:

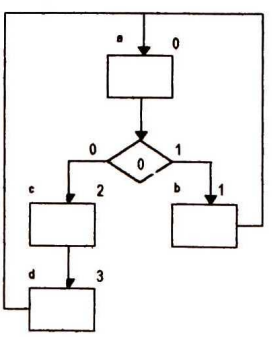


Fig. B.7.c. Carta ASM a editar.

5) Bueno, ahora es momento de establecer las variables de salida de la Carta ASM, para esto presionamos el botón  (botón con icono de lápiz), aparece una pequeña ventana de edición en la parte superior derecha del Área de Trabajo, como no se ha definido ninguna variable, ésta sólo se presenta con un mensaje que indica "NuevaVar" encerrada en un rectángulo.

Para proceder a teclear una variable presionamos la tecla [Enter] y empezamos a introducir el nombre de la variable deseada, en este caso NSV; para finalizar presionamos otra vez la tecla [Enter].

Lo anterior hay que realizarlo para cada una de las variables deseadas, a continuación presentamos la lista completa de variables que deben ser introducidas:

- ✓ NSV
- ✓ EOR
- ✓ NSA
- ✓ NSR
- ✓ EOV
- ✓ EOA

Observe que cada vez que realizamos lo anterior, el mensaje de "NuevaVar" se desplaza hacia abajo y las variables quedan en la parte superior.

6) Continuando con el proceso, relacionaremos las variables de salida con los estados correspondientes, es decir, ubicaremos cada variable de salida con el estado que le corresponda. Realicémoslo así:

De la ventana de edición de variables de salida seleccionamos por ejemplo a NSV (NSV aparece en los estados "a" y "c"), es decir, apuntamos a esta variable y presionamos el botón izquierdo del ratón y sin liberarlo, la arrastramos hacia uno de estos estados, digamos el "a", si ya estamos ubicados en este estado liberamos el botón del ratón.

Observe que inmediatamente se estable la variable en ese estado, de manera similar se procede para establecer la variable en el estado "c" y así para cada una de las variables. Para finalizar con el proceso de introducir variables, realícelo en el siguiente orden:

- ✓ NSV debe establecerse en los estados "a" y "c",
- ✓ EOR debe establecerse en los estados "a", "c" y "d",
- ✓ NSA debe establecerse en el estado "d",
- ✓ NSR debe establecerse en el estado "b" y
- ✓ EOV también debe establecerse en el estado "b".

La carta debe quedar conformada de la siguiente manera:

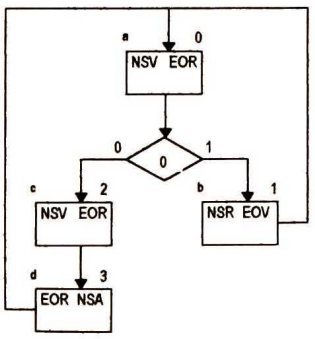


Fig. B.7.d. Carta ASM a editar.

7) Finalizaremos este ejemplo de edición con el cambio de nombre de los estados y de la Variable de prueba "0" del elemento decisión (no confundir con la rama "0" de dicha decisión). Para realizar el cambio de nombre presionamos el botón  (botón con icono de flecha de apuntador de ratón), a continuación seleccionamos el nombre del estado, en este caso "a", con lo que aparece una ventana solicitando el nuevo nombre -no debe ser mayor de cuatro caracteres-, en este punto teclee EDOA.

   Proceda de manera similar para los siguientes estados:

   - ✓ al estado "b" renombrelo con la etiqueta EDOC (note que no lo estamos renombrando como EDOB),
   - ✓ al estado "c" renombrelo con la etiqueta EDOB (también observe que no lo renombramos como EDOC),
   - ✓ al estado "d" renombrelo con la etiqueta EDOD y,
   - ✓ de manera similar a la variable de prueba "0" renombrela con la etiqueta TMP.

   Con lo que la Carta ASM queda como la de la Fig. B.7.a.

8) A continuación, obtendremos los resultados de la Carta ASM anterior, para realizar esto simplemente presionamos el botón  (botón con icono de diagrama de un circuito electrónico), para sintetizar la carta, al realizar esto observaremos las primeras tablas que genera el sistema, observará una barra de mensajes en la parte inferior de la pantalla, indicando las teclas que debe presionar para continuar observando los resultados, para regresarse a ver los resultados anteriores o para salir de este modo. Las teclas que se proporcionan para realizar lo anterior son:

   - ✓ [Pg Up] para observar resultados siguientes,
   - ✓ [Pg Dn] para regresarse a ver resultados anteriores,
   - ✓ [Esc] para salir del modo síntesis.

9) Si desea imprimir los resultados anteriores sólo tiene que presionar el botón de impresión  (botón con icono de impresora) y configurarlo para obtener los resultados deseados, recuerde que debe configurar la impresora en modo Epson.

   Con los pasos anteriores se cuentan con los elementos necesarios para hacer uso del sistema DGMAX/SADDAM.

   Esperamos que la información contenida en este manual sea de utilidad para simplificar su proceso de diseño.

APÉNDICE C

BIBLIOGRAFÍA

APÉNDICE C

BIBLIOGRAFÍA

NO.	AUTOR	TÍTULO	PAÍS	AÑO
1		Microsoft MS-DOS Guía del Usuario V. 3.3	Microsoft Corporation, México	1987
2		Funciones de DOS y BIOS	Editorial Addison-Wesley, México	1990
3		Manual del Usuario Impresora FX-850/1050	Epson	1990
4		User's Manual, Citizen 200GX, Dot Matrix Printer	Citizen America Corporation, Santa Monica, California	1990
5		User´s Manual NX-1001 multi-font dot matrix printer	Star Micronics Co., Ltd., USA	1990
6		Easy guide to graphics file and video standards	Compute's getting started with graphics standars, Compute publications international ltd., pp. 18	1991
7		Laser Printer HL-6/6T User's Guide	Brother Industries, Ltd, Japan	1993
8		Turbo C++, User's Guide	Borland International, Inc., USA	1992
9	A. Wiatrowski, Claude & H. House, Charles	Circuitos Lógicos y Sistemas de Microcomputadoras	Limusa, 1a. Edición, México	1987
10	Abrash, Michael	Graphics Programming	Dr. Dobb's Journal, Vol. 16, Núm. 8, pp. 165	1991
11	Abrash, Michael	Graphics Programming	Dr. Dobb's Journal, Vol. 16, Núm. 10, pp. 155	1991
12	Abrash, Michael	Graphics Programming	Dr. Dobb's Journal, Vol. 16, Núm. 11, pp. 123	1991
13	Betsira, Mark	VGA Palette Mapping Using BSP Trees	Dr. Dobb's Journal, Vol. 18, Núm. 7, pp. 28	1993
14	Betz, Mark	Interoperable objects	Dr. Dobb's Journal, Vol. 19, Núm. 11, pp. 18	1994
15	C. Lee, Samuel	Digital circuits & logical design	Prentice Hall, EUA	1976
16	Campbell, Tom	How to get started with graphics cards and monitors	Compute's getting started with graphics standars, Compute publications international ltd., pp. 4	1991
17	Cannavino, James A.	Object Insider: The future of object technology	Object Magazine, Vol. 4, Núm. 7, pp. 12	1994
18	Ceballos, Francisco Javier	Curso de Programación con "C"	Macrobit Editores, México	1990
19	English, David	How to with graphics standards	Compute's getting started with graphics standars, Compute publications international ltd., pp. 2	1991
20	Ezzell, Ben	Graphics Programming in Turbo C++	Addison-Wesley Publishing Company, Inc., Massachusetts, EUA	1990
21	Faison, Ted	Borland C++ 3.1 Programación Orientada a Objetos	Prentice Hall, 2a. Edición, México	1993
22	Fernández Villarroel, David	Diccionario de dudas e irregularidades de la lengua española	Editorial Teide, S.A., Barcelona, España	1991
23	Floyd, Michael	The object d'art	Dr. Dobb's, Journal, Vol. 16, Núm. 10, pp. 52	1991
24	Floyd, Michael	Comparing Object Oriented languaje	Dr. Dobb's Journal, Vol. 18, Núm. 11, pp. 104	1993

NO.	AUTOR	TÍTULO	PAÍS	AÑO
25	Götze, Ralf & Starrermayr, Gerhard	El gran libro de las HP-DeskJet	Marcombo, Barcelona, España	1994
26	G. Fichman, Robert & F. Kemerer, Chris	Object-Oriented and Conventional Analysis and Design Methodologies	Computer, Vol. 25, Núm. 10, pp. 22, IEEE Computer Society, N.J.	1992
27	Gaona Castillo, Areli	Diseño de interfaces ¿El secreto del éxito?	PC/Tips-Byte, Año 5, Núm. 51, pp. 10	1992
28	Goodwin, Mark	Graphical User Interfaces in C++ & Object-Oriented Programming	MIS: PRESS, Portland, Oregon	1990
29	Gossain, Sanjiv	Modelling Rules in Object-Oriented Analysis	Object Magazine, Vol. 4, Núm. 7, pp. 28	1994
30	Graham, Ian	Migration Strategies: You was my object ... but I done you wrong	Object Magazine, Vol. 4, Núm. 7, pp. 68	1994
31	Green, David	Modern Logic Design	Addison-Wesley, EEUU	1986
32	Harter, Richard	Object-Oriented Software Configuration Management	Dr. Dobb's, Journal, Vol. 16, Núm. 10, pp. 36	1991
33	Hekmatpour, Sharam	C++ guía para programadores en "C"	Prentice Hall Hispanoamericana, S. A. México, D. F.	1992
34	Henderson-Sellers, B.	A Book of Object-Oriented Knowledge	Prentice Hall, Australia	1991
35	I. Fletcher, William	An Engineering Approach to Digital Design	Prentice All, EEUU	1980
36	IBM	Object Technology, The Developer's & manager's Guide to the Future	IBM, International Business Machines Corporation	1994
37	IBM	On course for the future, How you can plan now to exploit the benefits of Object Technology	IBM, International Business Machines Corporation	1994
38	J. Hill, Frederick & R. Peterson, Gerald	Teoría de conmutación y diseño lógico	Editorial Limusa, México, D. F.	1990
39	J. Rochkind, Marc	Advanced C Programming for display	Prentice Hall, EEUU	1988
40	J. Tocci, Ronald	Sistemas digitales, principios y aplicaciones	Prentice Hall, 5a. Edición, México	1993
41	Johnson, Nelson	Advanced Graphics in C: Programming and Techniques	Osborne McGraw-Hill, Berkeley, California	1990
42	Kay Russell	Objects in use	Byte, Vol. 19, Núm. 4, pp. 99	1994
43	Ladd, Jim	Developing a medical diagnostic system wit OOA/RD	Object Magazine, Vol. 4, Núm. 7, pp. 48	1994
44	Leblanc, G.	Turbo C para IBM-PC y Compatibles	Editorial Gustavo Gili, S. A., Barcelona	1988
45	Lin Wang, Barbara & Wang Jin	Is a deep class hierarchy considered harmful?	Object Magazine, Vol. 4, Núm. 7, pp. 35	1994
46	M. Schevin-Tejera, Geneviéve & G. Tejera, Héctor	Diccionario Moderno de Informática, Inglés-Español	Grupo Editorial Iberoamérica, México, D. F.	1989
47	Martin, James & J. Odell, James	Object-Oriented Analysis and Design	Prentice Hall, Englewood, New Jersey	1992
48	Meyer, Bertrand	Applying "Design by Contract"	Computer, Vol. 25, Núm. 10, pp. 40, IEEE Computer Society, N.J.	1992
49	Minasi, Mark	How to upgrade your graphics card and monitor	Compute's getting started with graphics standars, Compute publications international ltd., pp. 10	1991
50	Minassi, Mark	How to choose a VGA System	Compute, Vol. 13, Núm. 11, pp. 52	1991

NO.	AUTOR	TÍTULO	PAÍS	AÑO
51	Mondragón Ballesteros, Jorge	Programe su propia interface gráfica	PC/Tips - Byte, Año 5, Núm. 51, pp. 77	1992
52	Morris Mano, M.	Lógica digital	Prentice Hall, Colombia	1982
53	O'Brien Stephen	Turbo Pascal 6, manual de referencia	McGraw-Hill/Interamericana, España	1991
54	Oros, Juan Carlos & Montes, Antonio	Impresoras Matriciales, Chorro de Tinta y Laser	Editorial Paraninfo, Madrid, España	1991
55	P. Deschamps, Jean & Angulo José Maria	Diseño de Sistemas Digitales	Editorial Paraninfo, 2a. Edición, España	1992
56	Peñaloza Romero, Ernesto	Fundamentos de Programación	U.N.A.M., México	1994
57	Pfaffenberger, Bryan	Diccionario para usuarios de computadoras	Prentice Hall Hispanoamericana, Naocalpan de Juárez, Edo. de México	1993
58	Poor, Alfred	Monitores de 16 pulgadas	PC Magazine, Vol. 2, Núm. 8, pp. 70	1991
59	Porras, Alejandro y Plácido, Antonio	Autómatas programables	McGraw-Hll, México	1990
60	Rieble, Richard	Generalized Methods in Object-based Languages	Computer Language, Vol. 8, Núm. 10, pp. 79	1991
61	S. Pressman, Roger	Ingeniería del Software, un enfoque práctico	Borland-Osborne/McGraw-Hills, Madrid España	1990
62	S. Weiner, Richard	Applications of object-oriented programming	Addison Wesley, Massachusetts, EUA	1991
63	Schildt, Herbert	Lenguaje C, programación avanzada	McGraw-Hill/Interamericana, México, D. F.	1990
64	Schildt, Herbert	Programación en Turbo C	Borland-Osborne/McGraw-Hill, Madrid España	1990
65	Schwork, Paul	About the paradigm	Object Magazine, Vol. 4, Núm. 7, pp. 60	1994
66	Shelton, Ted	Objects in Business: The challenge of change	Object Magazine, Vol. 4, Núm. 7, pp. 20	1994
67	Soley, Richard	Standards: Defining interfaces	Object Magazine, Vol. 4, Núm. 7, pp. 71	1994
68	Stevens, Al	C Programming	Dr. Dobb's Journal, Vol. 16, Núm. 8, pp. 149	1991
69	Stevens, Al	C Programming	Dr. Dobb's Journal, Vol. 18, Núm. 11, pp. 133	1993
70	Stevens, Al	C Programming	Dr. Dobb's Journal, Vol. 18, Núm. 7, pp. 115	1993
71	Stroustrup, Bjorne	The C++ Programming Languaje	AT&T Bell Laboratories, Murray Hill, New Jersey, Addison-Wesley Publishing Company	1992
72	T. Demel John & J. Miller Michael	Gráficas por computadoras	McGraw-Hill/Interamericana, México, D. F.	1990
73	Ungar, David; B. Smith, Randall; Chambers, Craig & Holzle Urs	Object, Message, and Performance: How They Coexist in Self	Computer, Vol. 25, Núm. 10, pp. 53, IEEE Computer Society, N.J.	1992
74	Voss, Greg & Chui Paul	Turbo C++ DiskTutor	Osborne McGraw-Hill, 2a. edición, United States of America	1991
75	W. Koontz, Richard	Lessons from the experts	Object Magazine, Vol. 4, Núm. 7, pp. 64	1994
76	Wayt Gibbs, W.	Software's Chronic Crisis	Scientific American, Vol. 271, Núm. 3, pp. 72	1994
77	Wegner, Peter	Dimensions of Object-Oriented Modeling	Computer, Vol. 25, Núm. 10, pp. 12, IEEE	1992

			Computer Society, N.J.	

APÉNDICE D

ARCHIVO DE ESTRUCTURA DE LA CARTA Y ARCHIVO DE SALIDAS

Definición de la estructura del archivo que describe a la Carta Asm.

ARCHIVO DE ESTRUCTURA DE LA CARTA	
CAMPO	COMENTARIO
NE	Valor entero que indica el número de elemento, éste es asignado de acuerdo a como se van creando en el sistema.
TE	Valor entero que indica el tipo de elemento, asignados como sigue: 0.- Para los estados. 1.- Para las salidas condicionales. 2.- Para los diamantes de decisión.
PX PY	Coordenadas (x,y) dentro del área de trabajo donde se ubica la esquina superior izquierda del elemento.
NOMBRE	Cadena de 4 cararacteres. Si el elemento es un estado indica el nombre del estado. Si el elemento es una salida condicional no se utiliza. Si el elemento es una decisión indica la variable de entrada asociada.
CÓDIGO	Cadena de cuatro caracteres. Se utiliza sólo si el elemento es un estado e indica el código asignado a éste.
SD	Es un apuntador que indica: Si el elemento es un estado o una salida condicional, el elemento al que está conectado el actual. Si el elemento es un diamante de decisión, el elemento al que está conectada la salida tomada en caso de que la variable de prueba sea cero.
SI	Es un apuntador utilizado solamente en los elementos diamante de decisión, indica el elemento al que está conectada la salida tomada en caso de que la variable de prueba sea uno.

Definición de la estructura del archivo que describe a las Salidas de la Carta Asm.

ARCHIVO DE SALIDAS	
CAMPO	COMENTARIO
NombreVar	Nombre de la variable.
NoEdo	Ocho dígitos enteros que indican en que estados aparece o está asignada la variable. Una variable puede estar asignada hasta en ocho estados distintos.

APÉNDICE E

DEFINICIÓN DE CLASES

```
/*****************************************************************************
PROGRAMA:              DEFINE.I
COMENTARIO:            Codigo fuente Turbo C++ para la definicion de las clasesempleadas en el sistema.
FECHA REVISION:        Enero/98
*****************************************************************************/
#ifndef DEFINE.I
#define DEFINE.I

#include <c:\tesis\programa\cabecera.h>
#include <c:\tesis\programa\grafinic.i>
/*****************************************************************************
Definicion de objetos y funciones
*****************************************************************************/
/*****************************************************************************
Definicion del objeto clase base Raton general
*****************************************************************************/
class Raton
{
    int Rver;          // Bandera de Visualizacion del cursor del raton
protected:
    Raton();           // constructor
    ~Raton();          // destructor
public:
    static raton_evento far* Reventos;  // Apuntador far global para registro
                                        // eventos del raton
    Rresultado *RRestaura();
    void RMuestra(int);
    Restado RPos();
    void RMueveA(int,int);
    Restado RPresionado(int);
    Restado RLiberado(int);
    void RxLimite(int,int);
    void RyLimite(int,int);
    Rmovimiento *RMueve();      // Movimiento neto del cursor
    void RMueve_Razon(int,int);
    void RAreaOculta(int,int,int,int);
    void RRapidez(int);
    void RLibera();         // Restaura el raton dejandolo en la posicion actual.
    void Pone_Cursor(g_cursor );
    void pone_cursor(int, int, unsigned, unsigned);
}; // class Raton

/*****************************************************************************
Definicion del objeto GRaton, hereda de Raton (para modo gráfico)
*****************************************************************************/
class GRaton : public Raton
{
private:      // figura de cursor grafico
    void pone_cursor(int,int,unsigned,unsigned);
public :
    void Pone_Cursor(g_cursor);
    void RPlumaOptica(int);
}; // class GRaton

/*****************************************************************************
Definicion del objeto TRaton, hereda de Raton
*****************************************************************************/
class TRaton : public Raton
{
public:             // figura de cursor texto
    void Pone_Cursor(int,unsigned,unsigned);
    void RPlumaOptica(int);
}; // class TRaton

/*****************************************************************************
Definicion de la Clase Unidad Grafica
*****************************************************************************/
class UnidadGrafica  // Esta clase se usa para inicializar la unidad grafica
{                    // como un objeto manejador del dispositivo grafico
int  TarjetaVideo,         // Tipo de la tarjeta de Video
     ModoGrafico,          // Valor del modo grafico
     MaxColores, // Maximos de colores disponibles
     CodigoError; // Reportes de cualquier error grafico
public:
     UnidadGrafica();               // Realiza la inicializacion de la unidad
     ~UnidadGrafica();              // Cierra la unidad grafica
}; // class UnidadGrafica

/*****************************************************************************
Definicion de la Clase Punto
*****************************************************************************/
class Punto
{
protected:
     int x,y,                  // (x,y) coordenadas del punto a dibujarse
     Color;                    // Color de visualizacion del punto a dibujarse
     viewporttype Pref;        // Salva las caracteristicas iniciales del puerto activo
public:
     Punto();
     virtual void Mueve(int,int);
     virtual void Dibuja();
     virtual void Crea(int,int,int);
     void RestauraPuerto();
     virtual void FijaColor(int);
     virtual void FijaLoc(int,int);
     virtual void Borra();
     int ObtenColor();
     int ObtenX();
     int ObtenY();
}; // class Punto

/*****************************************************************************
Definicion de la Clase Dispositivo (logicos)
*****************************************************************************/
class Dispositivo : public virtual Punto
{
protected:
/*****************************************************************************
**NOTA: Variables Heredadas de Punto.                                        **
**      (x,y): Vertice inicial del dispositivo.                               **
```

```
**    Color: Color del dispositivo.                                          **
**    PRef : Caracteristicas del puerto activo en el instante de creacion    **
*****************************************************************************/
    int  Edo,        // Estado de visualizacion del dispositivo Presionado/NoPresionado.
         Abierta,    // Indica si el dispositivo esta o no dibujado en pantalla.
         TxtTam,     // Indica tamano de la fuente a usar.
         Ancho,      // Dimension en x del dispositivo.
         Alto,       // Dimension en y del dispositivo.
         Rotar;      // Indica si el dispositivo esta en posicion horizontal o vertical.
    void *AreaOrigen;    // Guarda el area original en el cual se dibujara el dispositivo.
public:
    Dispositivo();
    Dispositivo(int PtX,int PtY,int Anch,int Alt,int C);
    ~Dispositivo();
    virtual void Crea(int PtX,int PtY,int Anch,int Alt,int C);
    vertice Vertices();
    int Dentro(int PtX,int PtY);
    void *SalvaImagen();
    void DibujArea();
    int ObtenAncho();
    int ObtenAltura();
    int ObtenEstado();
    int ObtenAbierta();
    int ObtenTxtTam();
    int DispElegido();
    virtual void Borra();
    virtual void Mueve(int PtX,int PtY);
    virtual void Arrastra();
    virtual void FijaColor(int C);
    void AlturaTexto(int TxtTamano);
    void LiberAreaOrig();
    void Puerto();
}; // class Dispositivo
/*****************************************************************************
Definicion de la Clase Etiqueta
*****************************************************************************/
class Etiqueta : public Dispositivo
{
/*****************************************************************************
**NOTA:                                                                      **
**    (x,y): Posicion en la cual se escribira el texto.                      **
**    Color: Color de la fuente a escribir.                                  **
**    PRef : Caracteristicas del puerto activo en el instante de creacion.   **
**    Edo : Define el tipo de la fuente a usar.                              **
**    Abierta: Sin aplicacion.                                               **
**    TxtTam: Tamano de la fuente a usar.                                    **
**    Ancho : Longitud del texto.                                            **
**    Alto : Altura de la fuente usada.                                      **
**    Rotar : Sin aplicacion.                                                **
**    AreaOrigen: Guarda el area original en el cual se escribira el texto.  **
*****************************************************************************/
    char texto[80];    // Almacena el texto de la etiqueta deseada.
public:
    Etiqueta();
    Etiqueta(int,int,char*);
    Etiqueta(int,int,int,int,int,char*);
    virtual void Cambia(int,int,char*);
    virtual void Crea(int,int,int,int,int,char *);
    virtual void FormaColor(int);
    void FuenteTamano(int);
    void Texto(char *);
    void EscribeTexto();
    void EscTexto();
    void EditaTexto(int NoCar = 59);
    char *ObtenTxt();
    void FijaColor(int);
    void FijaTamFte(int T); //Fija el Tamaño de la letra.
}; // class Etiqueta

/*****************************************************************************
Definicion de la Clase Boton (hereda de la clase Dispositivo)
*****************************************************************************/
class Boton : public Dispositivo
{
/*****************************************************************************
**NOTA:                                                                      **
**    (x,y): Posicion del vertice inicial del Boton.                         **
**    Color: Color del Boton a dibujarse                                     **
**    PRef : Caracteristicas del puerto activo en el instante de creacion    **
**    Edo : Estado de visualizacion del Boton Presionado/NoPresionado.       **
**    Abierta: Indica el Boton esta o no dibujado en pantalla.               **
**    TxtTam: Indica tamano de la fuente a usar.                             **
**    Ancho : Dimension en x del Boton                                       **
**    Alto : Dimension en y del Boton.                                       **
**    Rotar : Indica si el Boton es dibujado horizontal o verticalmente.     **
**    AreaOrigen: Almacena el area original en el cual se dibujara el Boton. **
*****************************************************************************/
    int TipoFte;            // Almacena el tipo de la fuente deseada.
    char BtnTxt[40];        // Almacena el mensaje del boton.
public:
    Boton();
    Boton(int PtX,int PtY,int Anch,int Alt,int C,char* Texto,int dib = 0,int borr = 0);
    ~Boton();
    virtual void Crea(int PtX,int PtY,int Anch,int Alt,int C,char* Texto,int dib = 0,int borr = 0);
    virtual void Mueve(int PtX,int PtY,int borr = 1);
    void Etiqueta(char* Texto,int borr = 1);
    virtual void FijaColor(int C,int borr = 1);
    void Estado(int BEdo,int borr = 1);
    void TipoFuente(int TxtFuente);
    void Invierte(int borr = 1);
    virtual void Dibuja(int borr = 1);
    int BotonElegido(int borr = 1);
}; // class Boton


/*****************************************************************************
Definicion de la Clase BarrDesp (hereda de la clase Dispositivo)
*****************************************************************************/
class BarrDesp : public Dispositivo
{
```

```
/*****************************************************************************
**NOTA:                                                                     **
**    (x,y): Vertice inicial de la Barra de Desplazamiento (BD).           **
**    Color: Color de la Barra de Desplazamiento.                          **
**    PRef : Caracteristicas del puerto activo en el instante de creacion. **
**    Edo : Color del contorno de la Barra de Desplazamiento (LineaColor). **
**    Abierta: Posicion del cursor de la Barra de Desplazamiento (BDPos).  **
**    TxtTam : Tamano del paso de desplazamiento de la BD (Pasos           **
**    Ancho: Dimension en x de la Barra de Desplazamiento.                 **
**    Alto : Dimension en y de la Barra de Desplazamiento.                 **
**    Rotar: Indica si la Barra de Desplazamiento es horizontal o vertical.**
**    AreaOrigen: Almacena la imagen del area donde se dibujara el cursor. **
*****************************************************************************/
    int AntPos;       // Posicion anterior del cursor de la Barra de Desplazamiento.
    Boton BtnExt1, BtnCursor, BtnExt2; // Cursores extremos y central de la BD.
public:
    BarrDesp(int PtX,int PtY,int Tamano,int CFr,int CFn,int Dir);
    ~BarrDesp();
    virtual void Crea(int PtX,int Pty,int Tamano,int CFr,int CFn,int Dir);
    virtual void FijaLoc(int PtX,int PtY);
    PresTipo BDPresionada();
    int ObtenPosicion();
    int ObtenDireccion();
    int ObtenPorcentaje();
    void Restaura();
private:
    virtual void Dibuja();
    void Traza();
    void PonFlechas();
    void PonCursor();
}; // class BarrDesp
/*****************************************************************************
Definicion de la Clase Ventana (hereda de la clase Dispositivo)
*****************************************************************************/
class Ventana : public Dispositivo
{
/*****************************************************************************
**NOTA:                                                                     **
**    (x,y): Vertice inicial de la Ventana.                                **
**    Color: Color de la Ventana.                                          **
**    PRef : Caracteristicas del puerto activo en el instante de creacion. **
**    Edo : Sin Aplicacion.                                                **
**    Abierta: Indica la Ventana esta o no dibujado en pantalla.           **
**    TxtTam: Sin Aplicacion.                                              **
**    Ancho: Dimension en x de la Ventana.                                 **
**    Alto : Dimension en y de la Ventana                                  **
**    Rotar: Indica si la Ventana es horizontal o vertical                 **
**    AreaOrigen: Guarda el area original en el cual se dibujara la Ventana.**
*****************************************************************************/
public:
    Ventana();
    Ventana(int r1, int c1, int anch, int alt, int vcolor);
    ~Ventana();
    virtual void Dibuja(int borr = 0);
    void crea(int r1,int c1,int r2,int c2,int w,int b,int s);
    void Limpia(int borr = 0);
    void Cierra();
    void Informa(char *Msg,int espera,int borra,int color);
}; // class Ventana

/*****************************************************************************
Definicion de la Clase MenuHorizontal (hereda de la clase Dispositivo)
*****************************************************************************/
class MenuHorizontal : public Dispositivo
{
/*****************************************************************************
**NOTA:                                                                     **
**    (x,y): Vertice inicial del Menu Horizontal.                          **
**    Color: Color del Menu Horizontal.                                    **
**    Edo : Sin aplicacion.                                                **
**    Abierta: Indica si el Menu Horizontal esta o no dibujado en pantalla.**
**    TxtTam:Color del texto del Menu                                      **
**    Ancho: Dimension en x del Menu Horizontal                            **
**    Alto : Dimension en y del Menu Horizontal                            **
**    Rotar: Color de la Tecla Aceleradora                                 **
**    AreaOrigen: Sin aplicacion.                                          **
*****************************************************************************/
    int NOpciones,              // # de opciones de la barra horizontal.
      Tabs,                     // Tabuladores de las opciones de la barra horizontal.
      OpcActiva,                // Opcion latente sin desplegar su submenu.
      OpcElegida,               // Opcion activa con submenu desplegado.
      ColorLetra,               // Color de la letra aceleradora.
      OpcSubMenu;               // # de opcion del submenu.
    char *TeclesRapidas;        // Cadena de caracteres de las teclas aceleradoras.
    MENU_ENCABEZADO *OpcionesH;     // Contiene las opciones completas de la barra.
    Ventana VMenu,VSubMenu,         // Barra del Menu Horizontal y Ventana del SubMenu.
            VAyudaTxt,VAyudaMsg;    // Barra de mensages de ayuda.
    void (*Menu_Ayuda)(void);
    void EscribeOpcion(int PtX,int PtY,char* Opc,int Pos,int Clr);
    void TeclaRapida(int PtX,int PtY,char* Opc,int Clr);
    void CursorActivo(int,int,int,int,char*,int);
    void OpcSeleccionada(int,int);
    void DeseleccionaOpcion(int);
    void DesmarcaOpcion(int);
    void DespSubMenu();
    void NuevaOpcion(int,int);
public:
    MenuHorizontal(int,int,int,int,int,int,int,int,MENU_ENCABEZADO *,void(*ayuda)(void) = NULL);
    ~MenuHorizontal();
    void Despliega();
    virtual void Borra();
    void Selecciona();
}; // class MenuHorizontal
/*****************************************************************************
Definicion de una Clase para manejar Cajas de Dialogo
*****************************************************************************/
class Dialogo
{
public:
    Dialogo() { };
```

```
    void DialogoConfirmacion(int,int,int,int,int,char*);
    char *DialogoLee(int,int,int,int,int,char*,int LongCadena = 55);
    int DialogoOpciones(int Cx,int Cy ,int Ax,int Ay,int Color,char *Msg,int GpoOpc = 0);
    void DialogoImpresion(int *TabImp);
}; //class Dialogo

/*****************************************************************************************
Definicion de la Clase Elemento (hereda de la clase Dispositivo)
*****************************************************************************************/
class Elemento : public virtual Dispositivo
{
/*****************************************************************************************
**NOTA:                                                                                  **
**    (x,y): Vertice inicial del Elemento.                                              **
**    Color: Color de contorno del Elemento.                                            **
**    PRef . Caracteristicas del puerto activo en el instante de creacion.              **
**    Edo : Probablemente sea color de fondo!!!.                                        **
**    Abierta: Sin Aplicacion.                                                          **
**    TxtTam: Tamano de la fuente a usar para calcular tamano de Elemento.              **
**    Ancho : Dimension en x del Elemento.                                              **
**    Alto : Dimension en y del Elemento.                                               **
**    Rotar: Sin Aplicacion.                                                            **
**    AreaOrigen: Guarda el area original en el cual se dibujara el Elemento.           **
*****************************************************************************************/
protected:
    Etiqueta Nombre;// Nombre del Elemento en cuestion.
    UneManual *UnionDer,*UnionIzq;
public:
    Elemento();
    ~Elemento();
    void LlenaColor();
    virtual void FijaColor(int C) {Color = C;};
    void Modifica(int Tam = 1);
    int EditaNombre();
    char *ObtenNombre();
    UneManual*GeneraSegmento(Coord,int IzqDer = 0, int Inicio_Fin_Union=1); // IzqDer = 1 para la lista Izq.
    int BorraSegmento(int IzqDer = 0);        // Inicio_Fin_Union=1 -> primer segmento
    int BorraUnion(int IzqDer = 0);
    Coord DibujaSeg(Coord, Coord, int Dib = 1,int Inicio_Fin_Union=1);    // Dib = 0 para borrar el Seg.
                                                                          //Inicio_Fin_Union=1 -> primer //segmento
    void DibujaUnion(int IzqDer = 0, int Dib = 1);
    Coord InicioSeg(UneManual*);
    void DspUnVis(int Direccion, int Desplazamiento);
}; // class Elemento

/*****************************************************************************************
Definicion de la Clase Estado (hereda de la clase Elemento)
*****************************************************************************************/
class Estado : public virtual Elemento
{
/*****************************************************************************************
**NOTA:                                                                                  **
**    (x,y): Vertice inicial del Estado.                                                **
**    Color: Color de contorno del Estado.                                              **
**    PRef : Caracteristicas del puerto activo en el instante de creacion.              **
**    Edo : Probablemente sea color de fondo del Estado!!!.                             **
**    Abierta: Sin Aplicacion.                                                          **
**    TxtTam: Tamano de la fuente a usar para calcular tamano del Estado.               **
**    Ancho : Dimension en x del Estado.                                                **
**    Alto : Dimension en y del Estado.                                                 **
**    Rotar: Sin Aplicacion.                                                            **
**    AreaOrigen: Guarda el area original en la cual se dibujara el Estado.             **
**    Nombre : Nombre del Estado en cuestion.                                           **
*****************************************************************************************/
protected:
    Etiqueta Codigo;        // Codigo del Estado en cuestion.
    int Desp;               // Ubicacion del rectangulo del Estado
public:
    Estado();
    virtual void Crea(int Tam,char *Nombr,char *Codig,int PtX = 0,int PtY = 0,int CForma = WHITE,
                int CFondo = getbkcolor());
    virtual void Dibuja();
    int CodigoEntero();
    void EditaCodigo();
    char *ObtenCodigo();
    void NuevaPosNomCod();
    void CambiaPos(Coord);
}; // class Estado

/*****************************************************************************************
Definicion de la Clase Decision (hereda de la clase Elemento)
*****************************************************************************************/
class Decision : public virtual Elemento
{
/*****************************************************************************************
**NOTA:                                                                                  **
**    (x,y): Vertice inicial de la Decision.                                            **
**    Color: Color de contorno de la Decision.                                          **
**    PRef : Caracteristicas del puerto activo en el instante de creacion.              **
**    Edo : Probablemente sea color de fondo de la Desicion!!!                          **
**    Abierta: Sin Aplicacion.                                                          **
**    TxtTam: Tamano de la fuente usada para calcular tamano de la Decision.            **
**    Ancho : Dimension en x de la Decision.                                            **
**    Alto : Dimension en y de la Decision.                                             **
**    Rotar : Sin Aplicacion.                                                           **
**    AreaOrigen: Guarda area original en el cual se dibujara la Decision.              **
**    Nombre : Condicion Booleana de la Decision.                                       **
*****************************************************************************************/
public:
    Decision();
    virtual void Dibuja();
    virtual void Crea(int Tam,char *Nombr,int PtX = 0,int PtY = 0,int CForma = WHITE,
                int CFondo = getbkcolor());
    void EditaCondicion();
    char* ObtenCondBool();
    void NuevaPosCondBool();
    void CambiaPos(Coord);
    virtual void Arrastra();
}; // class Decision
```

```
/*****************************************************************************************
Definicion de la Clase SalCondicional (hereda de la clase Elemento)
*****************************************************************************************/
class SalCondicional : public virtual Elemento
{
/*****************************************************************************************
**NOTA:                                                                                 **
**    (x,y): Vertice inicial de la Salida Condicional.                                 **
**    Color: Color de contorno de la Salida Condicional.                               **
**    PRef : Caracteristicas del puerto activo en el instante de creacion.             **
**    Edo : Probablemente sea color de fondo de la Salida Condicional!!!.              **
**    Abierta: Sin Aplicacion.                                                         **
**    TxtTam: Tamano de la fuente a usar para calcular tamano de la SalCond.           **
**    Ancho : Dimension en x de la Salida Condicional.                                 **
**    Alto : Dimension en y de la Salida Condicional.                                  **
**    Rotar : Bordeado de la Salida Condicional.                                       **
**    AreaOrigen: Guarda area original en el cual se dibujara la Sal_Cond.             **
**    Nombre : Sin Aplicacion.                                                         **
*****************************************************************************************/
public:
    SalCondicional();
    virtual void Dibuja();
    virtual void Crea(int Tam,int PtX = 0,int PtY = 0,int CForma = WHITE,int CFondo = getbkcolor());
    void CambiaPos(Coord);
}; // class SalCondicional

/*****************************************************************************************
Definicion de la estructura para generar una lista de Estados, Decisiones y Salidas Condicionales en CartasAsm
*****************************************************************************************/
struct Elem
{
    int Tipo;              // Tipo del Elemento: Estado/Salida Condicional/Decision.
    Estado *Est;           // Apuntador a un Estado en particular.
    SalCondicional *Sal;   // Apuntador a una Salida Condicional en particular.
    Decision * Dec;        // Apuntador a una Decicion en particular.
    Elem *ElemAnt,         // Apuntador a Elem anterior en la lista doblemente ligada.
         *ElemSig,         // Apuntador a Elem siguiente en la lista doblemente ligada.
         *ElemIzq,         // Apuntador a Elem izquierdo en la Carta ASM para la Decision.
         *ElemDer;         // Apuntador a Elem derecho en la Carta ASM para la Decision
}; // struct Elem

/*****************************************************************************************
Definicion de la Clase CartasAsm.
*****************************************************************************************/
class CartasAsm
{
    int  NoEdos,     // # de Estados totales de la Carta ASM.
         NoSal,      // # de Salidas Condicionales totales de la Carta ASM.
         NoDec,      // # de Decisiones totales de la Carta ASM.
         NoElem,     // # de Elementos totales de la Carta ASM. NOTA: Determinar si es necesaria.
         NoVarSal,   // # de Variables de Salidas de la Carta ASM. NOTA: Determinar si es necesaria.
         TamElem;    // Variable para dimensionar a los Elementos de la Carta ASM.
    Elem *ElemInicial,     // Apuntador al Elemento inicial de la Carta ASM.
         *ElemAsm,         // Apuntador al ultimo Elemento de la lista doblemente ligada.
         *ElemActual;      // Apuntador al ultimo Elemento activo de la Carta ASM.
    VarsEntSal Salidas[16];    // Tabla de Variables de Salida con sus Estados asociados.
    char NombreBase[9];
    void RestauraSalidas();
    void RestauraVariables();
    Estado *GeneraEdos(int PtX,int PtY,char *Nom = NULL,char *Cod = NULL);
    Decision *GeneraDecision(int PtX,int PtY,char *Nom = NULL);
    SalCondicional *GeneraSal(int PtX,int PtY);
    void EscribeSalidas(int,int,int);
    void ColocaSal(Elem*);
    void BorraVariable(int);
    void BorraSalEdo(int,int);
    void BorraElemento(Elem*);
    int VerificaUnion(Elem*,Elem*);
    void BorraUnion();
public:
    void EditaVar();
    void BorraE_S();
    void ColocaSalGral();
    CartasAsm();
    ~CartasAsm();
    void LibMemElem(Elem* Lib);
    void BorraAsm();
    virtual void Dibuja(int);
    Coord DaDimension();
    void PoneTamano(int Tam) {TamElem = Tam;};
    vertice EnlistaElementos(int Tipo,int CX = 0,int CY = 0,char *NOMBRE = NULL,
                             char *CODIGO = NULL);
    void NuevaPos(Elem*,Coord);
    Elem *ObtenElemActual() {return ElemActual;};
    Elem *ChecaElemActual();
    Elem *ChecaSeleccion(Coord Pos);
    Elem *ChecaSeleccion();
    vertice PosElem(Elem*);
    vertice ArrastraElemento(Elem*);
    void LiberaImagen(Elem*);
    void CapturaSalidas();
    void BorraSalida(int);
    int UnElementos(Elem*,Elem*);
    void CambiaPos(int,PresTipo);
    void LiberaImagenTotal();
    void GrabaArchivos(Coord OrigenAT);
    void LeeElementos();
    void GeneraArchASCII(Coord OAreaT,Coord Paso);
    void G_EDO_ASCII(FILE *Archivo,Coord CCasilla,char NombreSal[8][6]);
    Elem *ElementoInicial() {return ElemInicial;};
    int NoSalidas() {return NoVarSal;};
    VarsEntSal *TablaSal() {return Salidas;};
    void FijaNombre(char *NuevoNombre) {strcpy(NombreBase,NuevoNombre);};
    void LeeSecuencia();
    void ImprimeArch(Coord OAreaT,Coord Paso);
    void G_DEC_ASCII(FILE *Archivo,Coord PosEnArch)
    void CambiaCaracteristicas(Coord NuePos,int Tam);
    void CambiaDimPos(int,Coord);
    void CambiaElemActual(Elem* NuevoActual) {ElemActual=NuevoActual;};
    void LeeSalidas();
```

```
    void LeeSal();
    Elem *ElementoAnt(Elem*);
    Coord AsmXY(Elem*);
    int CartasAsm::Vacia() {if(!ElemInicial) return 1; return 0; };
    void FijaTamElem(int TamNuevo){ TamanoElem = TamNuevo; };
    int ObtenTamElem() { return(TamanoElem); };
    void ImprimeCarta(Coord OAreaT, Coord Paso);
    void G_SAL_ASCII(FILE *Archivo,Coord CCasillas); //Genera una salida condicional
                                                     //dentro de un archivo ascii
}; // class CartasAsm

/*****************************************************************************
Definicion de la Clase Sintesis.
*****************************************************************************/
class Sintesis
{
    int NoVarEnt;      // Numero de variables de Entrada (Condiciones Booleanas)
    char TablaProg[16][56];    // Tabla de almacenamiento del Micro Programa.
    VarsEntSal Entradas[16];  // Tabla de almacenamiento de las entradas de la Carta ASM.
public:
    Sintesis();
    int MicaValida(Elem *);
    void DefineOperaciones(Elem*, int, VarsEntSal*),
    void InstruccionesMical(int,int);
    void AccionModoCta(int,int);
    void Tabla_Ep_Ld(int, int);
    void MuestraEstados(int, int);
    void MuestraSalidas(int, int, int, VarsEntSal*);
    void MuestraEntradas(int, int);
    void MuestraOperaciones(int, int, int);
    void MuestraTabla(int, int, int);
    void MuestraHexa(int,int,int);
    void Diagrama(int,int,VarsEntSal*);
}; // class Sintesis

/*****************************************************************************
Definicion de la Clase Area_trabajo (hereda de la clase Dispositivo)
*****************************************************************************/
class Area_trabajo : public Dispositivo
{
/*****************************************************************************
**NOTA:                                                                      **
**    (x,y): Vertice inicial del Area de trabajo en pantalla.               **
**    Color: Color de la cuadricula del Area de Trabajo en pantalla.        **
**    PRef : Caracteristicas del puerto activo en el instante de creacion.  **
**    Abierta: Sin Aplicacion.                                              **
**    TxtTam: Tamano de los elementos de la carta Asm, asi como de la cuadricula **
**          (Antes TamElementos).                                           **
**    Ancho : Dimension en x del Area de Trabajo.                           **
**    Alto : Dimension en y del Area de Trabajo.                            **
**    Rotar: Sin Aplicacion.                                                **
**    AreaOrigen: Sin aplicacion.                                           **
*****************************************************************************/
// Estas constan de dos componentes: Long. de dx,dy de cada elemento de la cuadricula
    Coord   Origen,       // Coordenada de la casilla inicial de Area de Trabajo en pantalla.
            Margen,       // Sangria lateral del Elemento y su casilla de AT.
            OrigAT,       // Origen del Area de trabajo no con respecto a la pantalla.
            Paso,         // Dimension en x e y de las casillas de Area de Trabajo.
            PasElem;      // No se para que se usa. Tal vez este de mas.
    char NomCarta[9];     // Nombre del archivo o carta en edicion.
    CartasAsm Carta;      // instancia de CartasAsm

/*****************************************************************************
*** Estructura dedicada al manejo de las posiciones de Area de trabajo
*** ocupadas por algun tipo de entidad
*****************************************************************************/
    Lista *Ultimo;
    int AdicionALista(int x, int y, char Marca, char Salida);
    int BorraDeLista(int x, int y);
    char ObtenMarca(Coord *);
    char CambiaMarca(int x, int y, char Marca, char Salida);
    void CambiaListaXY(int x, int y, int x1, int y1);
    void CambiaTodaListaXY(int x1, int y1);
    void IniciaLista();
    void ConstruyeLista();

    void Dimensiona(int Tam);
    void PoneOrigen(int a,int b);
    Coord Coordxy(int i,int j);
    Coord Coordij(int x,int y);
    Coord LugarVacio();
    void IncOrigATX(int IncX) {OrigAT.x += IncX;};
    void IncOrigATY(int IncY) {OrigAT.y += IncY;};
    int RutaParcialOK(Coord Fte,Coord Dtn,Coord Fin,int Dir);
    int Direccion(Coord Fte, Coord Dtn, int Dir);
    Coord IncrementaPaso(Coord Fte, int Dir);
    Coord RutaParcial(Coord Fte, Coord Dtn, int Dir);
    char ObtenDir(char Direccion, char Suma);
    void FlechitasUnion(char Dir,Coord Pt,int Dib);
    int InvierteDireccion(int Dir);
public:
    Coord Ajusta(Coord puntoen,char Opcion);
    Coord Ubica(Coord puntoen);
    Coord AlVertice(Coord puntoen);
    Coord AlCentro(Coord puntoen);
    void Cuadricula(int bandera);
    void Zoom();
    Area_trabajo();
    void Agrega(int tipo);
    void Manipula();
    void CapturaVarSal();
    void RestauraCoord();
    void IncOrigenAT(PresTipo);
//========= Funciones miembro para el manejo de las Cartas ASM
    void Editar();
    void Borrar();
    void CartaCambiaPos(PresTipo Dir);
    Elem *VerificaSel();
    int PideArch();
    void LeeCarta();
```

```
    void EscribeCarta();
    void MuestraSintesis(int ConjOpc); // Manipula la presentacion de las tablas
    void GeneraElementos();
//========== funciones miembro para la union visual ================
    Coord Une(Coord * , char , char, Coord *, char Dib);
    Coord CreaUnion(Coord *, Coord *, char Dib);
    void DibujaUniones(char Dib);
    char FijaMarca(Coord *, char , char);
    char FijaMarca(Coord , char);
    char FijaSalida(Coord *, char Dir);
    void Imprimir();
    void Escala(int);
}; // class Area_trabajo

#endif
```

*** NOTA:
Para mayor detalle de la interface de las funciones consulta el código correspondiente a cada una de las funciones incluidos en el disco del sistema.

Los abajo firmantes, integrantes de jurado para el examen de grado que sustentará **Ing. Efren López Martínez,** declaramos que hemos revisado la tesis titulada:

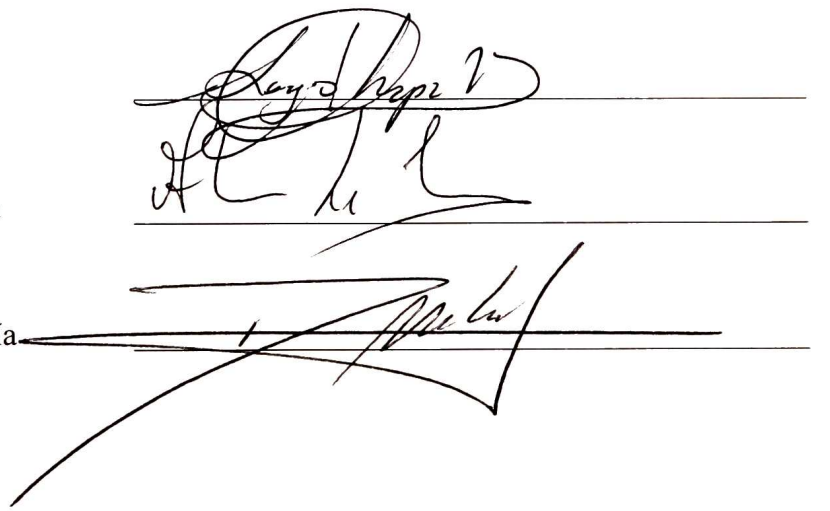
"SINTESIS DE ARQUITECTURAS MEDIANTE CARTAS DE MAQUINAS DE ESTADOS ALGORITMICAS" consideramos que cumple con los requisitos para obtener el Grado de Maestro en Ciencias, con especialidad en Ingeniería Eléctrica.

Atentamente

Dr. Sergio V. Chapa Vergara

Dr. Adriano de Luca Pennnacchia

Dr. Felipe Rolando Mencha García



```
void EscribeCarta();
void MuestraSintesis(int ConjOpc); // Manipula la presentacion de las tablas
void GeneraElementos();
```