Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

EXPERIENCIAS SOBRE UNA METODOLOGIA DE PROGRAMACION

Rafael O. Fontao
Laboratorio de Sistemas Digitales
Departamento de Ingeniería
Universidad Nacional del Sur
Bahía Blanca - ARGENTINA

Juan A. Codagnone
Departamento de Sistemas
ATEC S.A. de Asesoramiento Técnico
Buenos Aires - ARGENTINA

INTRODUCCION

En este trabajo se expresan algunas de las ideas y experiencias en el uso de una metodología de programación. Esta técnica permite hacer frente a la crisis del ambiente informático, caracterizada especialmente por una indisciplina generalizada en el diseño e implementación de programas y sistemas.

Al igual que las normas de escritura de la programación estructurada clásica, esta metodología comparte sus objetivos. Sin embargo, como única estructura de control, y por lo tanto de pensamiento o concepción de programas, ofrece el autómata finito.

La metodología resultante permite unificar el diseño de programas para hacerlo independiente del lenguaje de programación. El trabajo consta de 4 secciones principales, la primera describe brevemente la metodología; las dos secciones siguientes tratan sobre las experiencias académica y profesional adquiridas con esta técnica. Finalmente la 4° sección propone las extensiones y líneas de trabajo a seguir por esta metodología.

DESCRIPCION DE LA METODOLOGIA

En un trabajo previo [2] fue presentada formalmente la metodología SOL. Esta metodología está soportada por un lenguaje cuyas características se incluyen resumidas en el apéndice.

La idea básica de esta metodología consiste en concebir a un programa como un autómata finito y determinista. Luego en sucesivos pasos, denominados refinamientos, se conciben los estados como otros autómatas a niveles inferiores.Este procedimiento concluye cuando todos los estados del autómata así concebido pueden sintetizarse mediante instrucciones en un lenguaje estandard disponible.

Un autómata finito (AF) modela un comportamiento secuencial que puede expresarse por una tabla de transiciones o su grafo equivalente. Partiendo de un estado inicial, el AF permanece en cada instante en un único estado produciendo una salida y ante el estímulo de una entrada, también finita, transita a un nuevo estado (eventualmente el mismo).

Así, se alternan sucesivamente salidas del AF con entradas al mismo hasta que arriba a un estado final y se detiene. La concepción de un AF se realiza en forma descendente expresando en pocos estados la descripción de su comportamiento. Sucesivamente los estados se van refinando en nuevos autómatas hasta alcanzar un nivel de detalle que puede implementarse con los recursos desponibles.

Análogamente, un programa o más generalmente la descripción de una tarea, puede concebírsela como un AF, utilizando para ello la noción clave que relaciona ambas disciplinas: interpretar el estado de un AF como tarea elemental de un programa.Por tarea elemental se entiende un paso del programa o tarea con un único punto de entrada y uno o más de salida. Similarmente el estado de un autómata es único a cada instante pero puede transitar a uno entre varios próximos estados posibles.

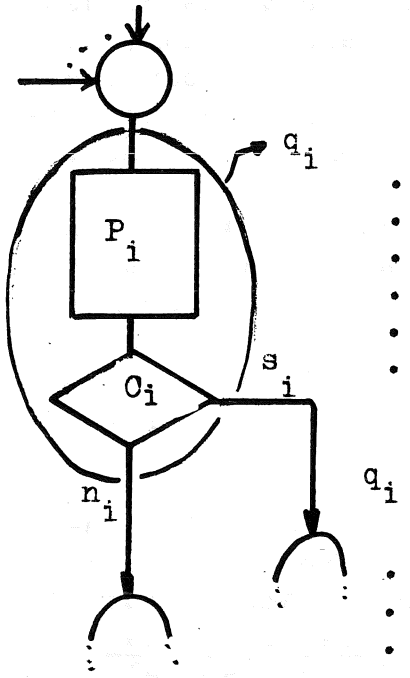
La salida de un estado será considerada como la acción de la tarea (no como el resultado de esa acción). Así por ejemplo, A= B+C como tarea es siempre la misma aún cuando

CEDINFOR

el resultado sobre A sea diferente al ajecutar B+C con otros valores. La acción de cada estado, a su vez, puede descomponerse en dos partes: la ejecución de la tarea y la evaluación de la próxima entrada.

La ejecución de la tarea corresponde a la aplicación de las herramientas de trabajo mientras que la evaluación de la próxima entrada (decisión) corresponde a la aplicación de herramientas de medida.(Las herramientas de trabajo transforman la materia prima:información, y las de medida permiten medir las propiedades de la materia prima) [2].

La Figura 1 muestra genéricamente una tabla de transiciones de un AF que modela un programa. A los efectos de simplificar el modelo sin perder generalidad cada estado tiene a lo sumo dos próximos estados posibles, esto es, basta una condición lógica para determinarlo.



ENTRADA		ACCION	
SI	NO	EJECUCION TAREA	EVALUACION PMA.ENTRADA
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
s_i	n_i	P_i	C_i
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.

Fig. 1

Desde un punto de vista temporal, la evolución del AF es la siguiente:

ENTRADAS:	E^1	$E^2 \ldots E^t$	$E^{t+1} \ldots E^n$
ESTADOS: $q_{inicial}$	q^1	$q^2 \ldots q^t = q_i$	$q^{t+1} \ldots q_{final}$
SALIDAS:	P^1	$P^2 \ldots P^t = P_i$	$P^{t+1} \ldots$ STOP

La evaluación, para cada instante t, de la próxima entrada es,

$$E^{t+1} \begin{cases} SI & : \text{ si } C_i \text{ es verdadera (o vacía)} \\ NO & : \text{ si } C_i \text{ es falsa} \end{cases}$$

por consiguiente el próximo estado, será :

$$q^{t+1} \begin{cases} s_i & : \text{ si } E^{t+1} \text{ es SI} \\ n_i & : \text{ si } E^{t+1} \text{ es NO} \end{cases}$$

La metodología SOL puede aplicarse en sucesivos pasos o refinamientos. A cada paso un módulo o parte del programa se refina explicitando en mayor detalle su comportamiento o programándolo directamente en el lenguaje de programación. Este proceso de refinamiento continúa hasta que todo el programa queda escrito en el lenguaje de programación.

De igual modo, un AF puede diseñarse en sucesivos pasos dando lugar a la siguiente metodología de programación:

Sea LPD el Lenguaje de Programación Disponible con el cual se implementará finalmente el programa.

Paso 0: Elija un lenguaje adecuado L para expresar sus ideas (idealmente un lenguaje natural) y conciba a todo el programa como si fuera un AF de un sólo estado.

Paso 1 a n: Si hay algún estado que no puede escribirse "claramente" en el LPD, entonces conciba a este estado (en L) como un AF. Esto es, un estado del AF es reemplazado por un conjunto de nuevos estados formando entre ellos un AF. (Como regla de claridad conciba cada AF con el menor número de estados posibles y teniendo en cuenta, a su vez, la validez de su concepción).

Paso n+1: (En este paso todos los estados de AF deberían ser "claramente" expresados en el LPD). Escriba en el LPD todos los estados del AF.

En esta metodología de programación, la estructura de control de un programa (Refinamiento, Pasos 1 a n) se escri-

be separadamente del resto de las sentencias (paso n+1). La estructura de control, a su vez, se modela por las transiciones de estado de un AF cuyos estados son el producto de un proceso de refinamientos. Es decir, cada estado cuya acción se describe en lenguaje natural, o se lo escribe en el LPD (paso n+1) si fuera simple o se lo concibe como un nuevo AF (paso entre 1 y n).

Por consiguiente, un programa puede visualizarse como una jerarquía de autómatas donde la relación de dependencia se describe por una estructura de árbol. La raíz de este árbol es un AF de un sólo estado (paso 0) que simboliza a todo el programa. Por cada refinamiento de estado se agregan al nodo del árbol correspondiente tantos descendientes directos como estados tenga el refinamiento. Este proceso continúa hasta que todo nodo terminal del árbol (o estado del AF) pueda modelarse con una acción escrita en el LPD. Ver a continuación.

```
REF
 1 DO...
   THEN ..
 2 DO ...
   THEN ..
 3 DO ...
   THEN ..
END

REF 2
 1 DO ...
   THEN ..
 2 DO ...
   THEN ..
END 2

REF 3
 1 DO ...
   THEN ..
 2 DO ...
   THEN ..
 3 DO ...
   THEN ..
END 3

REF 3.1
 1 DO ...
   THEN ..
 2 DO ...
   THEN ..
END 3.1
FIN
```

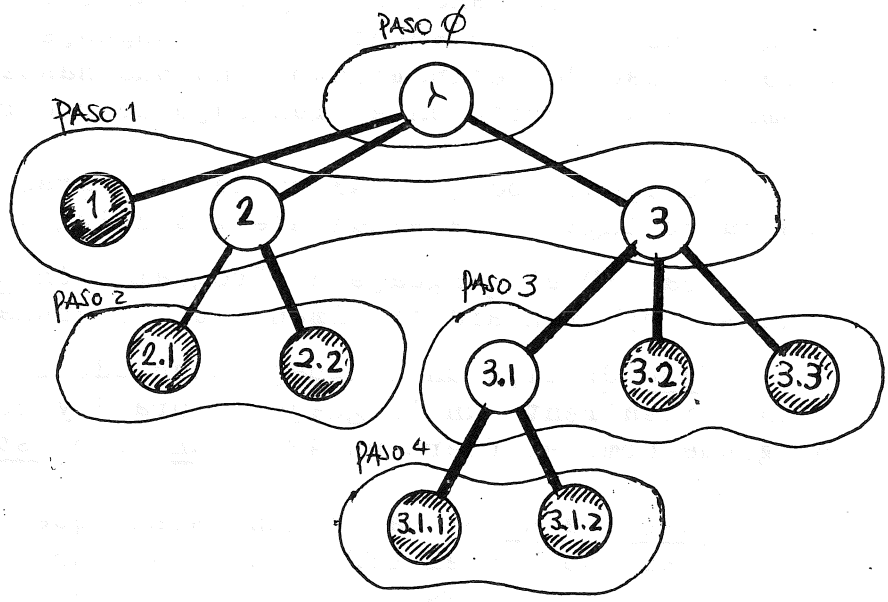


Fig. 2

A partir de aqui se escriben en LPD las hojas u nodos terminales del árbol que constituyen las acciones (nodos en sombra).

EXPERIENCIA ACADEMICA

La experiencia académica en programación suele caracterizarse por su influencia en la enseñanza y en el desarrollo de programas, sin sufrir necesariamente presión de factores externos, como pueden afectar a la experiencia profesional.

Esta particularidad es atractiva en cuanto el docente puede ensayar las metodologías de programación que con libertad académica juzque convenientes. Sin embargo, es en cierta medida la experiencia profesional, la que confirma la aplicabilidad de toda metodología.

En este sentido, la experiencia académica comenzó con el diseño y construcción de un precompilador SOL-BASIC [1] implementado para una minicomputadora PDP/8e de nuestra Universidad.

Poco tiempo después, razones accidentales motivaron dejar de usar el equipo mencionado con la configuración original. No obstante, la metodología resultante de programas en SOL, continuó su influencia en el diseño de programas y en la enseñanza misma de programación.

Cuando no se dispone del precompilador la aplicación de SOL consiste en expresar los refinamientos como si fueran comentarios del lenguaje disponible. Las acciones, por su parte, se escriben teniendo en cuenta la estructura de control establecido en los refinamientos, y en tal sentido las instrucciones de salto incondicional, cuando realmente son necesarias, se limitan específicamente a transferir el control dentro del mismo refinamiento. Las transferencias de control fuera del refinamiento se efectúan siguiendo estrictamente las estructuras definidas por los refinamientos.

En otras palabras, un programa escrito siguiendo esta metodología, contiene una primera parte formada únicamente por un gran "comentario estructurado" donde se especifican todas las relaciones entre los refinamientos. Luego en la segunda parte, se explicitan las acciones comentando solo los límites de cada una.

De este modo la documentación del programa se encuentra concentrada en un único lugar, al principio, lo que per-

mite tratarla como archivo, independiente del resto del programa. Sin lugar a dudas que un programa escrito en SOL sin la documentación respectiva, sería suficiente para producir un colapso entre los estructuralistas, ya que al nivel del lenguaje no es necesario más que concatenación y salto condicional como estructuras básicas.

Sin embargo, es precisamente esta documentación de los refinamientos la que nos da mayor riqueza de información; más aún, podríamos prescindir de las acciones y ser capaces de reconstruir el programa; la inversa, en cambio, sería impracticable la mayoría de las veces.

Con respecto a la enseñanza de programación, la metodología SOL ha tenido una influencia importante en la propia metodología de la enseñanza. La presentación de conceptos, en sí misma, encierra toda una estructura de refinamientos que disciplina el estudio tanto a educandos como educadores. No hemos realizado experimentos didácticos y estadísticos porque la población estudiantil y la disponibilidad de equipos no es lo suficientemente amplia como para asegurar la validez de alguna conclusión inferida estadísticamente. Si, en cambio podemos afirmar que en el diseño de programas por parte de docentes, la metodología SOL ha influído notoriamente en la calidad de los mismos.

EXPERIENCIA ATEC

En los párrafos siguientes se indica la experiencia obtenida durante el desarrollo de un sistema computerizado para un Municipio de la Provincia de Buenos Aires, donde si bien no se dispuso de un precompilador, se aplicó para el desarrollo de los programas la metodología SOL.

El proyecto comprendió, luego del análisis y dictado de normas para los procedimientos administrativos que regulan los sistemas, el desarrollo de 50 programas escritos en lenguaje COBOL, con más de 30000 líneas de código en total.

El objetivo principal consistió en encontrar para los programas estructuras de control simples que permitieran una rápida visualización del flujo de información y control. De este modo se lograría que la tarea de implementación, documentación y puesta a punto de los mismos se viera facilitada. Con ese criterio se adoptó la metodología SOL, aplicada en la forma que se describe a continuación:

Para la implementación de cada programa, a partir del 'nivel enunciado' se procedió a definir cada una de sus componentes, especificadas a través de sucesivos refinamientos, como se indica en la figura3 .

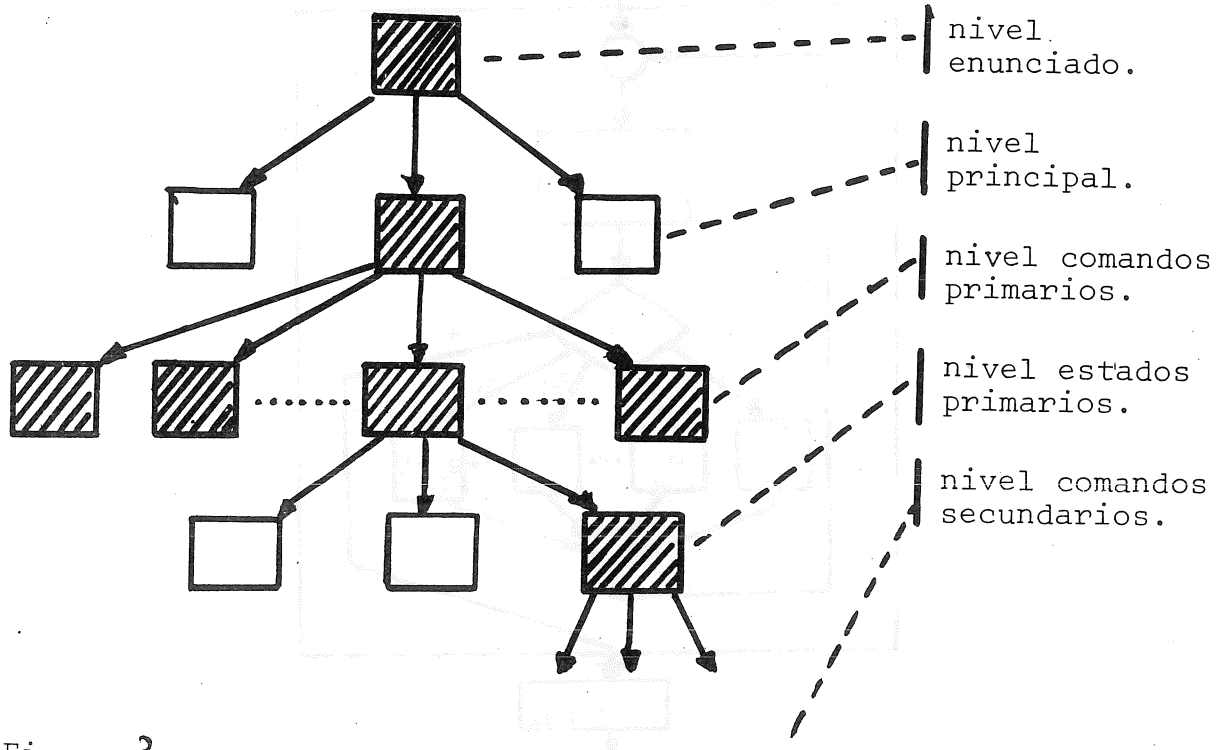


Figura 3

La primera etapa consiste en definir el 'nivel principal' del programa, mediante tres estados o módulos consecutivos, con la estructura indicada en la figura 4 y que en la sintaxis de SOL puede expresarse como se indica a continuación:

```
REF
  1  DO  inicializaciones, apertura de archivos y validaciones necesarias para la correcta ejecución del programa.

     THEN 2

  2  DO  desarrollo de todas las opciones que ejecuta el programa

     THEN 3

  3  DO  cerrar archivos, emitir estadísticas y terminar

     THEN STOP

END
```

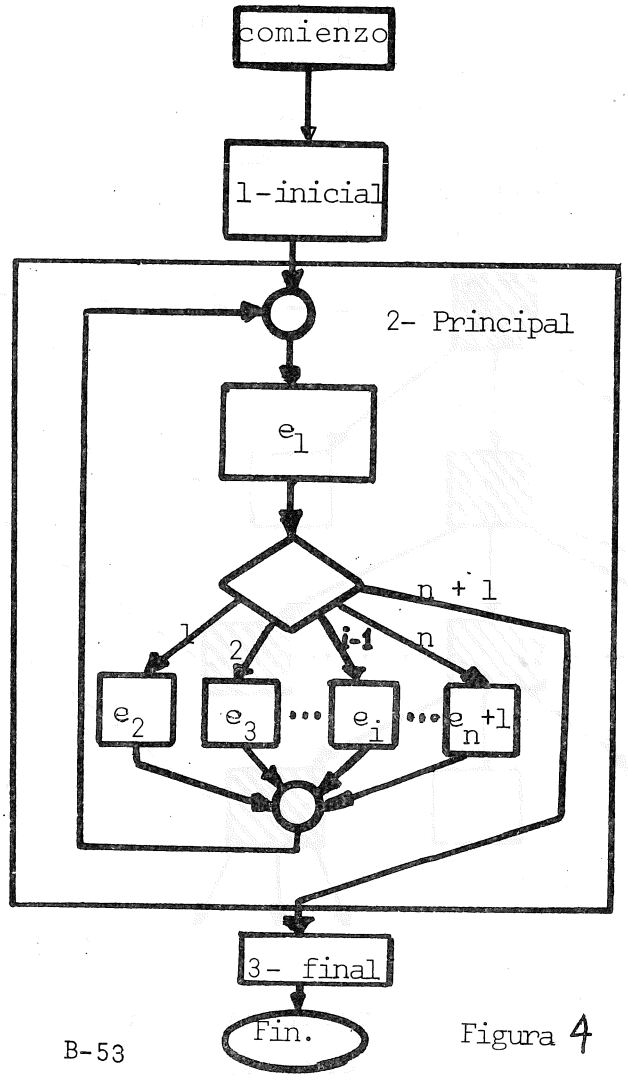


Figura 4

En la segunda etapa se refina el est. 2, módulo principal del programa, como un conjunto de estados, ejecutables de acuerdo a la entrada seleccionada por el operador o de acuerdo con condiciones internas del programa, dando origen al 'nivel comandos primarios'. En él, cada uno de los procesos que se llevan a cabo se los identifica con un estado denominado ek con k= 1,2,....,n,n+1. Se ejecuta inicialmente el estado e_1 y de acuerdo a la opción seleccionada por el operador o las condiciones internas detectadas por el programa, se ejecuta el estado e_i, con $2 \leq i \leq n+1$.

La entrada n+1 (que correspondería al est. e_{n+2}), queda reservada para la salida de este módulo, Según la sintaxis de SOL esta estructura puede definirse como se indica a continuación:

```
REF  2
  1   DO  seleccionar opción(1 a n para ejecutar estados 2
          a n+1 y n+1 para salir)
      THEN 2 OR 3 OR..... OR  i  OR...OR n+1 OR  EXIT1
  2   DO  ejecutar proceso correspondiente a opción 1
      THEN 1
  3   DO  proceso correspondiente a opción 2
      THEN 1
  .
  .
  .
  .
  i   DO  proceso correspondiente a la opción i-1
      THEN 1
  .
  .
  .
  .
  .
 n+1  DO  proceso correspondiente a opción n
      THEN 1
 END 2
```

En la tercera etapa cada uno de los estados componentes del 'nivel comandos primarios' se lo refina obteniéndose el

'nivel estados primarios'. Cada módulo es definido con la estructura que se indica en la figura 5 (el estado 3 se repite hasta que una condición de salida se produzca y retorna al nivel superior), y que en la sintaxis de SCL pueda expresarse, para i = 2,3,....,n_{+1}, como:

```
REF  2.i

  1      DO  estado inicial
         THEN  2

  2      DO se sigue procesando? Afirmativo, estado 3. Negativo, estado 4.
         THEN 3 OR 4

  3      DO  realizar proceso correspondiente a este módulo
         THEN 2
  4      DO  finalizar y volver a refinamiento superior para
         nueva opción o terminar

         THEN EXIT1

END 2.i
```

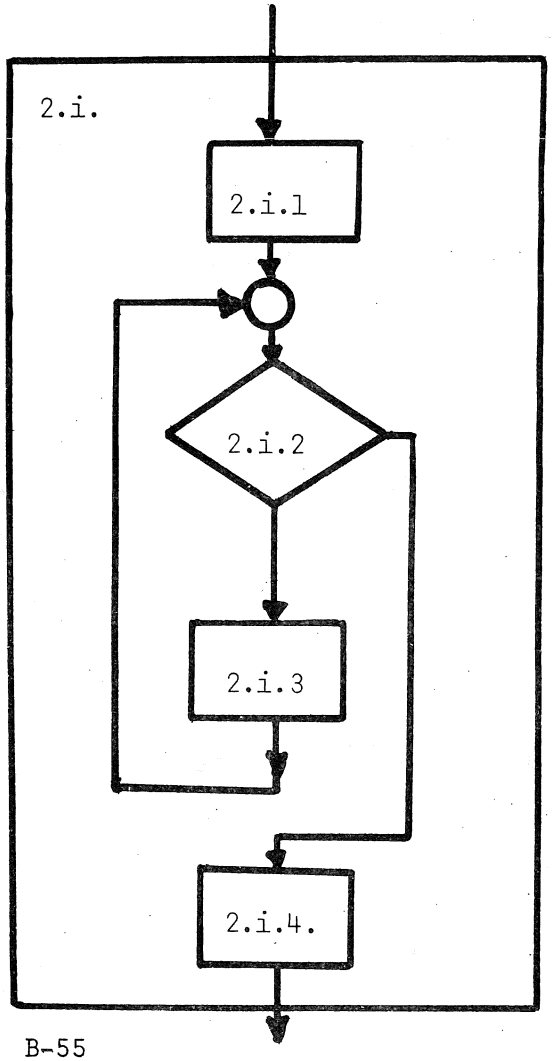


Figura 5

El próximo nivel refina al estado 2.i.3, que depende de cada caso en particular, En muchos casos, en este nivel hay estados implementados por rutinas que pueden ser utilizadas por más de un estado del mismo nivel, o estados similares de otros programas que pueden ser incluídos en el mismo en forma directa.

Al momento de la codificación, cada estado se concibió, dado que el lenguaje de programación era COBOL, como un módulo consistente en una SECCION invocado en el refinamiento superior por una sentencia PERFORM o un subprograma invocado mediante la sentencia CALL, desde el nivel superior.

Si bien se puede argumentar que una SECCION de programa invocada por un PERFORM no cumple todas las condiciones que exige la definición módulo independiente dado que no es capaz de compilarse independientemente, que puede interactuar libremente con otros sectores del programa y que no necesariamente tiene una definición de datos independientes, en el desarrollo que se describe, se prohibió que secciones o estados independientes interactuaran entre sí en otra forma que no fuera la especificada en los refinamientos. A cada SECCION se le asignó un grupo de datos de características locales, además de los globales pertenecientes al programa.

Otra norma adoptada fue la de prohibir que los GO TO's utilizados dentro de una SECCION se refirieran a marcas o rótulos externos a ella, con lo cual se logró independencia entre secciones.

Como corolario de esta experiencia se indica que en la medida que se respetaron las técnicas descriptas en el desarrollo de los distintos subsistémas, además del alto nivel de productividad de los programadores, se logró que la tarea deadaptación y mantenimiento de los programas no resultara una carga importante de trabajo. La estructura modular permitió que las modificaciones introducidas solo involucraran a un módulo en particular, sin extenderse hacia otros sectores, con la consiguiente generación de código confuso y poco claro. Por otra parte, dada la estructura común para casi todos los programas también lo fue la operación, redundando en una pronta adaptación del personal para el manejo de los sistemas.

EXTENSIONES PROPUESTAS

Actualmente parte del grupo de trabajo está implementando un sistema de programación interactiva utilizando la técnica SOL.

La manipulación de un programa parcialmente desarrollado, permitirá bajo ciertas condiciones, la posibilidad de ejecución simbólica de los refinamientos y la simulación de acciones aún no escritas.

Esta particularidad es atractiva, por cuanto el programador interactivo tendrá la posibilidad de ejecutar programas parcialmente desarrollados.

Una línea de trabajo propuesta, constituye la aplicación inversa de la metodología SOL a programas ya escritos. Este aspecto, denominado decompilación, permitiría rescatar en forma de refinamientos la estructura de control de un programa.

La aplicación de SOL al desarrollo de programas para Microprocesadores es parte también del objetivo de investigación.

En este sentido la relación entre programa y autómata se va estrechando para confundirse definitivamente en su implementación práctica.

CONCLUSIONES

Las experiencias logradas mediante la aplicación de la metodología SOL en diseño, desarrollo y mantenimiento de programas, permiten inferir mayor alcance tanto en el medio académico como profesional.

Las líneas de trabajo originadas por esta metodología suponen una intensificación del uso de la propia computadora para el desarrollo de programas. Esta programación interactiva se verá favorecida por la enorme proliferación de procesadores que se estima para la presente década.

Por otra parte el uso de metodologías como la propuesta sirven de marco a un medio unificado de programación, donde los programadores expresen sus ideas por medio de la estructura de refinamientos, independientemente del lenguaje a utilizar.

Este trabajo resume el estudio en una metodología original desarrollada y enriquecida durante los últimos tres años.

REFERENCIAS

1 CODAGNONE, J.A. "Implementación de un Precompilador SOL- BASIC". Informe Proyecto Final Dto. de Ingeniería - U.N.S. 1976.-

2 FONTAO,,R.O., ARDENGHI, J.R. y ARROYO, E.H.: "Sobre una Metodología de Programación". V Panel Computación y Expodata. Valparaíso, Chile (1978).-

APENDICE

LENGUAJE SOL

El lenjuaje SOL está orientado a mostrar explícitamente como se concibe la estructura de un programa mediante refinamientos similares a autómatas finitos.

El lenguaje mismo se diseñó para ser precompilado a un lenguaje práctico disponible. Las características sintácticas de SOL lo hacen especialmante adecuado para ser precompilado en un solo paso.

A continuación se dan resumidas las reglas sintácticas (no dependientes del contexto) que definen al lenguaje.

SINTAXIS

Se dan a continuación las producciones BNF para describir la sintaxis de SOL.

<programa>::=<estructura de control> <descripción de acciones>

<estructura de control>::=<refinamiento> FIN | <refinamiento> <estructura de control>

<refinamiento>::= REF <identificador><definición de AF> END <identificador>

<identificador>::= λ | <identificador propio>

<identificador propio>::=<entero positivo>|<entero positivo>• <identificador propio>

<entero positivo>::=entero positivo sin signo

<definición de AF>::=<descripción de estado>|<descripción de estado><definición de AF>

<descripción de estado>::=<entero positivo> DO <descripción - tarea elemental> THEN <lista de próximos estados>

<descripción-tarea elemental>::=descripción en el lenguaje natural L de la tarea que se realiza en ese estado

<lista de próximos estados>::=<identificador próximo estado>| <identificador próximo estado> OR <identificador próximo estado>

<identificador próximo estado>::=<entero positivo>|EXIT <entero positivo>|STOP

<descripción de acciones>::=<acción> FIN|<acción><descripción de acciones>

<acción>::=ACT <identificador><salida de estado><condición del estado>

<salida del estado>::=programa en el LPD

<condición del estado>::=NEXT|NEXT (<condición lógica>)

<condición lógica>::=condición lógica permitida en el LPD

SEMANTICA DE SOL

En lenguaje SOL, la estructura de un programa se define separadamente del resto de las acciones.

La estructura de control se compone,a su vez, de una sucesión de refinamientos (pasos 1 a n) concebidos como AFs. Los símbolos REF y END se usan para delimitar la definición de un refinamiento. El identificador se forma concatenando por medio de un punto (.) el identificador y el número del estado que será refinado. Así por ejemplo, si el estado 3 del refinamiento 3.2.4 debe ser refinado, entonces su identificador será 3.2.4.3. En particular, el primer refinamiento (paso 1) tendrá identificador nulo (Λ); es decir, el identificador del refinamiento de un estado del primer refinamiento será simplemente un entero (el número del estado que se refina).

Por consiguiente, la profundidad de un refinamiento se representa explícitamente en SOL por medio de una lista ordenada de enteros.

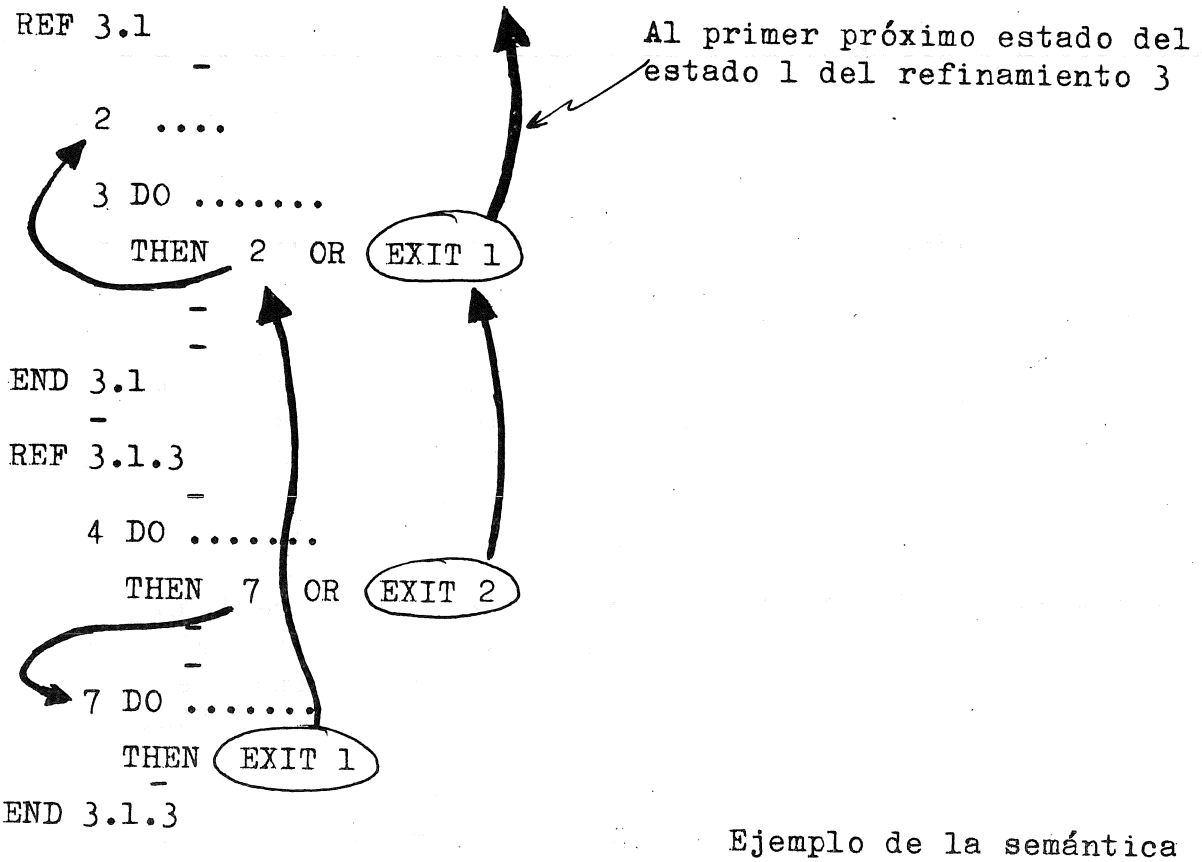
El estado inicial de un refinamiento será 1 y consecutivamente se numerarán los demás estados. Entre las palabras paréntesis DO y THEN se describirá el trabajo elemental a realizarse en cada estado. Esta descripción corresponderá a la acción (salida y condición) de la tabla de transiciones definida en la figura 1. El primer estado en la lista de próximos estados corresponderá a la entrada SI y el segundo (si existe) corresponderá a la entrada NO (figura 1).

El identificador del próximo estado puede ser de los siguientes tipos:

a) Entero positivo: identificará a un estado del mismo AF o refinamiento.

b) EXIT i: significa que el próximo estado será el indicado por el estado i-ésimo de la lista de próximos estados del refinamiento generador. (En el presente modelo i puede tomar solo dos valores 1 y 2).

c) STOP: significa que se ha llegado al estado final del AF.



Ejemplo de la semántica del próximo estado.

Con la palabra FIN termina la definición de los refinamientos. Estos forman un árbol cuyos nodos terminales representan los programas elementales que serán implementados a continuación.

Entre los símbolos ACT y NEXT se escribirán, las acciones correspondientes a las acciones del AF que modelará el programa. Sólo ACT lleva el identificador de la acción. Si NEXT no es seguida por una condición lógica entre paréntesis ello significa que hay un sólo próximo estado posible, de

otro modo el próximo estado estará indicado por el valor lógico de la condición: SI para el primero y NO para el segundo de los próximos estados.
