Saltos no Locales: Una extensiòn a Turbo-PASCAL

Rodrigo Lòpez B.
Universidad de los Andes
Facultad de Ingenieria
Departamento de Sistemas y Computaciòn
Apartado Aèreo 4976
Bogotà - Colombia

RESUMEN

En los ambientes de programaciòn interactivos son tipicos los mòdulos que detectan y se recuperan de los errores; en consecuencia, el flujo normal del control dentro de los programas se altera de manera impredecible al detectar esas situaciones anormales. La justificaciòn clàsica de la presencia de los saltos incondicionales (GOTO) en Pascal ha estado relacionada con una soluciòn "estructurada" para el tratamiento de errores; sin embargo, en Turbo-PASCAL el alcance de los saltos incondicionales es local al procedimiento en donde se definen las etiquetas (label) lo que impide efectuar saltos desde el interior de un procedimiento hacia el final de un programa (por ejemplo).

En este articulo se muestra la implementaciòn de una estrategia de soluciòn para el tratamiento de errores en Turbo_PASCAL, similar a las construcciones ERRSET-ERROR o CATCH-THROW de Lisp. La soluciòn es muy eficiente pues consta de unas pocas lineas en lenguaje ensamblador y es muy sòlida pues las verificaciones que atañen al flujo del control se realizan ùnicamente en dos puntos del programa.

I INTRODUCCION

A pesar de las múltiples críticas que ha recibido la primitiva de control GOTO, a través de la historia de la programación, parece haber consenso (aún entre sus detractores) acerca de su utilidad en situaciones muy precisas, en donde son válidos tanto los argumentos de eficiencia en la ejecución como los de claridad en la programación. De hecho, en lenguajes "estructurados" como PASCAL, se incluyó el GOTO en la versión original aun cuando se hacen advertencias con respecto a su uso [JEN75].

En la versión más popular de PASCAL para microcomputadores de 8 y 16 bits, Turbo-PASCAL [BOR84], existe el GOTO pero su alcance es local. En este artículo se discute una extensión a Turbo-PASCAL (en su versión para sistema operacional MS-DOS) por medio de la cual se pueden efectuar saltos no-locales dentro de un programa. Se ilustra la necesidad de tal facilidad con situaciones tomadas de la implementación de un ambiente de programación real que se utiliza actualmente en el primer curso de programación en la Universidad de los Andes [LOP87a], [LOP87b].

II EL SISTEMA Turbo-KAREL.

El sistema Turbo-KAREL es un ambiente de programación autocontenido que se utiliza como introducción a la programación en un lenguaje imperativo (típicamente PASCAL). El lenguaje de programación utilizado es KAREL y la implementación realizada en la universidad de los Andes se ajusta a la definición hecha por Richard Pattis. KAREL es un robot que se desplaza en un "mundo" cartesiano en donde puede haber obstáculos (muros) y un tipo particular de señales (pitos); para más aclaraciones se puede consultar [PAT81]. Este proyecto se adelantó con financiación del Fondo Nacional de Proyectos de Desarrollo del gobierno colombiano, funciona en microcomputadores del tipo PC-IBM o compatibles y es de distribución gratuita dentro de Colombia.

La implementación del sistema se realizó enteramente en Turbo-PASCAL y consta, además de un "compilador" y un intérprete para los programas de karel, de un editor de programas y de un editor gráfico e interactivo para los mundos . Editar un mundo consiste en definir los muros, los pitos y la posición inicial del robot. Un programa "compilado" se "interpreta" sobre un mundo seleccionado.

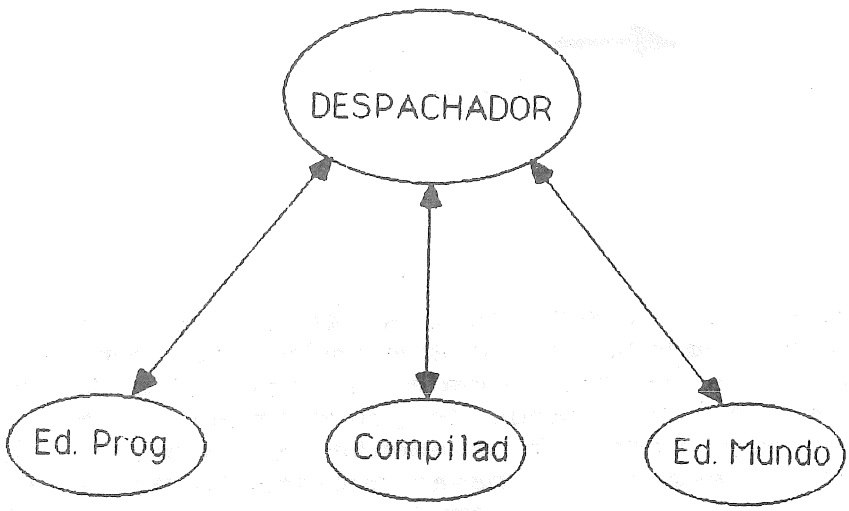
III. FLUJO NORMAL DEL CONTROL EN Turbo-KAREL

La estructura de control de Turbo-KAREL es similar a la de muchos sistemas interactivos en los que un despachador de comandos distribuye el control a diferentes módulos servidores y éstos lo

retornan, a su turno, al despachador de comandos [TOR87]. Los mòdulos servidores en Turbo-KAREL son:

- Editor de Programas
- Editor de Mundos
- Compilador
- Intèrprete.
- Otros: Conexiones con las facilidades del sistema operacional residente.

En el modo normal de operaciòn el control fluye ùnicamente en las direcciones especificadas por las flechas del siguiente diagrama.

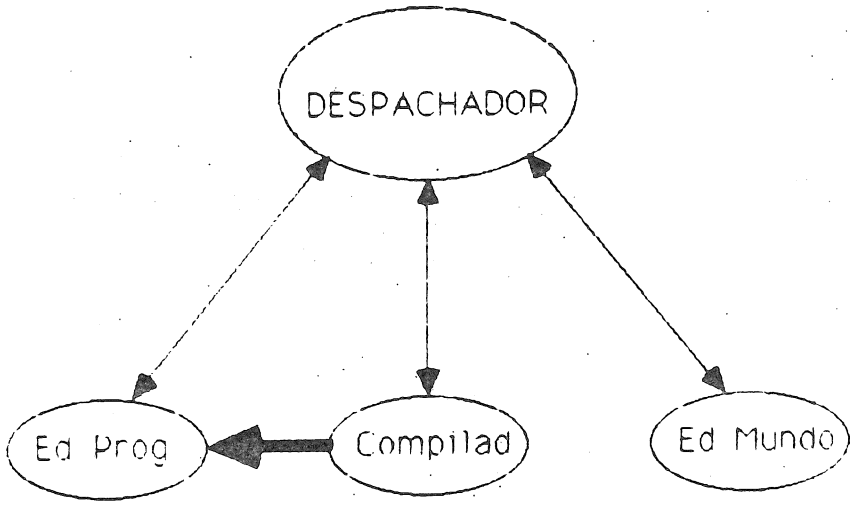


IV FLUJO ANORMAL DEL CONTROL: Recuperaciòn de Errores.

En Turbo-KAREL se manejan los errores tipicos que comete el usuario de un ambiente de programaciòn:

- Errores de sintaxis, manejados por el compilador.
- Errores de ejecuciòn, manejados por el intèrprete.

En los dos casos se suspende la ejecuciòn del servicio solicitado (compilaciòn o interpretaciòn) y el usuario puede elegir, retornar al despachador de comandos (flujo normal del control) o pasar directamente a editar el programa en el sitio en donde sobrevino el problema (flujo anormal del control).



En un sistema de las características anteriores, programado en PASCAL, presumiblemente el despachador de comandos es un procedimiento y la manera de ceder el control a los módulos de servicio es la clásica llamada de procedimiento. No es posible con este esquema pasar bruscamente el control entre dos servicios: es imprescindible pasar nuevamente por el despachador de comandos. En el caso, por ejemplo, de un error de compilación, la dificultad no estriba en "saber cuál fué la línea del texto en donde apareció el error" sino en lograr que desde un sitio "totalmente arbitrario" en la ejecución, pueda cedérsele el control a un procedimiento específico, "olvidándose de casi todo lo que había ocurrido".

Ilustremos con un esquema de programa en Pascal, cuál es la situación. Consideremos el siguiente programa:

```
Program Ejemplo;
  Procedure P;
    Procedure Q;
      Procedure R;
        Procedure S;
         Begin
         End;
        Begin
        End (* de R *);
      Procedure T;
        Begin
        End (* de T *);
      Begin
      End (* de Q *);
```

```
    Procedure U;
       Begin
       End (* de U *);
    Begin
    End (* de P *);
Begin
End. (* de Ejemplo *)
```

Según las reglas de alcance de Pascal, la siguiente sucesión de llamadas es válida:

Ejemplo llama a P llama a Q llama a Q llama a Q llama a T llama a T llama a R llama a S llama a S llama a S

Supongamos que en este instante de la ejecución se quiera ceder el control al procedimiento U. Dadas las restricciones del alcance, la solución sería:

- Retornar bruscamente el control a P (con todo lo que ello implica para los ambientes de variables locales y parámetros de los procedimientos activos entre P y S).

- Que P le ceda el control a U.

Recordemos que en TurboPascal, todos los GOTOs son locales al procedimiento en donde se definen, lo que hace imposible forzar un retorno directo a P con el fin de que él se encargue de la redistribución del control. Una solución sencilla, pero extremadamente frágil, en TurboPascal, sería utilizar una variable global que es colocada en True cuando aparece la situación que requiere de ese retorno especial. Entonces, el valor de esa variable debería ser consultado por TODOS LOS PROCEDIMIENTOS POTENCIALMENTE ACTIVOS para que ellos retornen el control "a quien los invocó" y de esta manera, en algún momento se llega a P quien toma las acciones apropiadas. Es evidente que esta solución, aun cuando puede funcionar, exige demasiado código adicional y no es sencillo certificar su validez. Por otro lado es muy ineficiente pues requiere que todos los procedimientos involucrados, verifiquen explícitamente la ocurrencia o ausencia de la situación anormal para tomar las providencias adecuadas.

V UNA SOLUCION CLASICA IMPLEMENTADA EN Turbo-PASCAL

En la situación descrita anteriormente, cuál es el problema de fondo?. Es necesario hacer retornar todos los procedimientos activos ya que los ambientes de datos de cada uno de ellos (Parámetros y variables locales) se convertirían en objetos carentes de sentido si el control fuera bruscamente a P. Dado que estos ambientes se manejan en una pila, una solución alterna

sería:

- Marcar el ambiente activo en el sitio que va a retomar el control (Dentro del procedimiento P).
- Marcar la dirección de retorno en ese mismo punto.

Estas dos marcas se conocen como el "contexto de ejecución" en un instante dado [PRA81], ya que ellas definen la próxima instrucción a ejecutarse junto con los datos que tienen sentido en ese momento. Finalmente, el último paso sería:

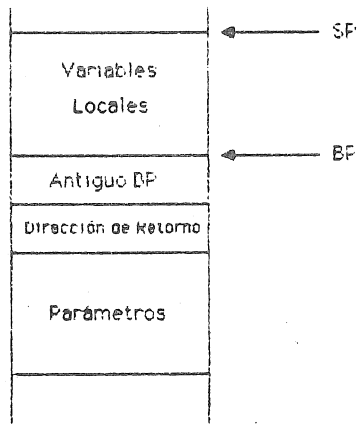
- Utilizar las marcas (recuperar el contexto), en el sitio en donde se quiera forzar un retorno brusco.

En nuestro ejemplo, el contexto se marca en dos sitios diferentes:

- Dentro de P se marca el nivel de la pila de ambientes.
- Dentro de Q se marca la dirección de retorno a P!!!. Esto es importante pues es el procedimiento Q quien conoce el punto de continuación del control dentro de P, ya sea que Q retorne normalmente o que se produzca un retorno brusco a partir de algún procedimiento invocado directa o indirectamente por Q.

y se recupera en S cuando así se requiera.

Para lograr todo lo anterior en turbo pascal, es preciso manipular a bajo nivel la pila de ambientes puesto que (afortunadamente) no es posible hacerlo directamente con instrucciones de Pascal. Según se explica en el manual de TurboPascal, un ambiente local está definido por los valores de los registros BP y SP del procesador 8088 (ver figura).



Las variables locales se encuentran encerradas entre estos dos registros, cuyos valores son direcciones tales que

ContenidoDe(BP) >= ContenidoDe(SP).

Por otro lado, desde la direcciòn apuntada por BP, hacia las direcciones superiores en la memoria, el contenido es:

- Antiguo valor de BP.
- Direcciòn de retorno del procedimiento activo. Note que este valor corresponde a la direcciòn de una de las instrucciones del procedimiento que invocò al que està activo en este momento.
- Parâmetros del procedimiento activo.

Con esta informaciòn, unas pocas lineas en assembler resuelven el problema. Es recomendable utilizar variables enteras y globales para almacenar los valores de los registros. El còdigo de màquina se inserta por medio de la instrucciòn InLine de TurboPascal en los sitios adecuados, con la ventaja de poder mencionar "por su nombre" las variables globales requeridas. Concretamente, supongamos las siguientes declaraciones globales:

```
Var
   Base:Integer;       (* Valor de BP *)
   Tope:Integer;       (* Valor de SP *)
   Retorno: Integer;   (* Direcciòn de Retorno *)
```

Los trozos de còdigo serian:

- Marcar el ambiente de datos:

```
Mov ax,bp
Mov Base,ax
Mov ax,sp
Mov Tope,ax
```

- Marcar la direcciòn de retorno:

```
Mov ax,[bp+2]  (* Pues el antiguo BP ocupa 2 bytes *)
Mov Retorno,ax
```

- Recuperar el contexto:

```
Mov ax,Tope
Mov sp,ax
Mov ax,Base
Mov bp,ax
Mov ax,Retorno  (* Hasta aquí se recuperò todo el
                   contexto *)
```

```
Push ax        (* Coloque la dir de retorno en la pila *)
Ret            (* Retorne *)
```

En nuestro sistema Turbo-KAREL se adoptò esta soluciòn, la que por otro lado es muy similar a las construcciones de alto nivel ERRSET-ERROR o CATCH-THROW de algunos dialectos de Lisp. Concretamente, en el manejo de errores de compilaciòn se requiere:

- El despachador de comandos marca el ambiente de datos antes de cederle el control al compilador.
- La llamada al compilador se hace a travès de un procedimiento "fantasma" cuya ùnica funciòn es marcar el punto de retorno en el despachador e invocar al verdadero compilador.
- El anàlisis sintàctico se realiza con el mètodo del descenso recursivo lo que lleva a situaciones como las descritas anteriormente. Si ocurre un error, quien lo detecte, llama al procedimiento ERROR quien es el encargado de recuperar el contexto para el retorno brusco al despachador de comandos.
- Al recuperar el control, el despachador de comandos verifica el valor de una variable que le indica si el retorno fue normal o anormal. Esta variable es manipulada ùnicamente por el procedimiento ERROR.

Como se ve, la soluciòn es muy sòlida pues solamente se tiene conciencia de la situaciòn anormal en dos puntos "extremos" del programa: en el despachador de comandos y en el procedimiento de ERROR. Los demàs procedimientos involucrados directa o indirectamente, reciben, ceden y retornan el control sin ocuparse de los problemas potenciales. Finalmente, es muy eficiente pues el còdigo requerido se reduce estrictamente a las 13 instrucciones en lenguaje ensamblador (mostradas anteriormente) màs la escasa manipulaciòn de la variable de "comunicaciòn" conocida por el despachador de comandos y el procedimiento de ERROR.

Por otro lado, trae consecuencias en cuanto al "estilo de programaciòn" en los sitios en donde se detectan los errores, en donde aparece claramente que el control interrumpe su flujo normal. El siguiente es un trozo de programa, tomado de uno de los procedimientos del analizador sintàctico, que ilustran este estilo inhabitual (pero estructurado) en Pascal:

```
Case SiguienteToken of
   TokenIterate:
      Begin
        p:= ConsNi(SiguienteToken,Nil,Nil,Linea);
        DemeSiguienteToken;
        If (SiguienteToken <> Numerico) then ERROR(19);
        If (ValorToken > 255) then ERROR(21);
        q:=ConsNi(ValorToken,Nil,Nil,Linea);
        p^.Cuerpo := q;
        If (SiguienteToken <> TokenTime) then ERROR(20);
        DemeSiguienteToken;
        q^.Cuerpo:= AnaliceInstruccion;
      End;
  .......       etc    .......
```

Obsèrvese que, por la forma del programa, es evidente que el procedimiento ERROR no retornarà el control a la siguiente instrucciòn.

VI CONCLUSIONES

Con este ejercicio de programaciòn hemos corroborado que Turbo-Pascal se adapta a casi cualquier necesidad de programaciòn, de manera clara y eficiente. La situaciòn esbozada en el articulo es tipica de los sistemas que detectan y se recuperan de los errores; nuestra soluciòn aparece como un "esquema de control" que puede adoptarse fàcilmente en Turbo-PASCAL, es muy eficiente, confiable e introduce en PASCAL un estilo estructurado para saltos no-locales, similar a las construcciones ERRSET-ERROR o CATCH-THROW de algunas versiones de Lisp.

REFERENCIAS

[BOR86] Borland Inc.
"Turbo-PASCAL: Turbo Tutor"
Borland International Inc, Scotts Valley, U.S.A., 1984.

[JEN75] Jensen Kathleen, Wirth Niklaus.
"PASCAL User Manual and Report".
Springer-Verlag, New York, U.S.A., 1975.

[LOP87a] Lòpez Rodrigo, Toro Victor.
"Turbo-KAREL: Manual del Sistema"
Publicaciòn CIFI-Uniandes, Bogotà, Colombia, 1987.

[LOP87b] Lòpez Rodrigo, Toro Victor.
"Turbo-KAREL: Libro del Proyecto"

Publicación CIFI-Uniandes, Bogotá, Colombia, 1987.

[PAT81] Pattis Richard.
"Karel the Robot: A Gentle Introduction to the Art of Programming"
Prentice-Hall Int, Englewood Cliffs, U.S.A., 1981.

[PRA81] Pratt Terrence.
"Programming Languages: Design and Implementation"
Prentice-Hall, Eglewood Cliffs, U.S.A., 1984.

[TOR87] Toro Victor.
"Diseño, Especificación y Prototipificación de Sistemas Interactivos".
En este volumen.