

Objetos Natural (II)

Cerrando el tema iniciado el pasado mes, el autor, con el presente artículo, expone lo que para él supone cada uno de los objetos que permiten la modularización en NATURAL, así como el uso que en su modesta opinión se debe hacer de cada uno de ellos.

1. Introducción

El pasado mes, esta sección se dedicó a describir los distintos objetos que pueden usarse bajo la plataforma NATURAL. Dicha descripción se hizo en plan general, tal y como cuentan los manuales proporcionados por la casa diseñadora, **SOFTWARE AG**, para que sirviera de introducción a este segundo artículo, con el cual su autor expone una visión partidista del tema de acuerdo con su experiencia, al tiempo que define cómo se pueden combinar los distintos objetos a la hora de diseñar un proceso. No obstante, tampoco descubre nada que en teoría no se sepa, ya que simplemente se limita a considerar al 'programa' como un elemento transformador de información, e incorpora a su planteamiento el principio básico de que todo programa debe tener un único punto de entrada y un único punto de salida, pues en informática también es aplicable el dicho popular: 'casa con dos puertas¹ mala es de guardar'.

2. Esquema general de un comando

Hasta ahora se ha hablado de objetos Natural de una forma indeterminada, pero en cualquier Centro de Proceso de Datos (CPD) se debe distinguir entre procesos **Online** y procesos **Batch**. Ambos procesos pueden combinar todos los tipos de objetos descritos, pero el planteamiento que se haga de los mismos debe ser distinto, ya que, en el primer caso, se asocia a procesos muy cortos que tratan una única entrada o registro, mientras que los segundos tratan todos los registros de un fichero, y además los procesos que tienen asociados suelen ser más largos y pesados².

El esquema que se presenta a continuación permite dar vida a la regla "un programa un mapa"³ válido para **procesos Online**. Esta regla permite dar claridad al programa y facilitar el mantenimiento, pues obliga a descomponer en funciones elementales problemas complejos. Además, permite clasificar como proceso tipo programa todo aquel módulo que invoca a un mapa, dejando los tipos subrutina y subprograma para aquellos subprocesos que validan o elaboran información elemental. No obstante, se debe puntualizar que no todos los programas de tipo principal tienen que tener mapa, tal es el caso de los programas batch.

La regla "un programa un mapa" da claridad al programa y facilita el mantenimiento y la modularización.

NATURAL, como ya se ha dicho, permite descomponer los farragosos programas COBOL que existían antes de su llegada en el año 85, en programas que escasamente pueden ocupar dos pantallas, ajustándose en cualquier caso al siguiente esquema en el que el orden de los puntos es invariable aunque puede darse el caso de que alguno de ellos se encuentre vacío :

¹En informática es habitual, aunque no académico, el que un programa pueda abandonarse por múltiples puntos. Por ejemplo usando la sentencia **FETCH**.

²Se entiende por tratamiento pesado aquel que tiene un gran número de accesos a bases de datos y/o ficheros.

³Esta regla ha sido llevada a la práctica por el autor de este artículo, y ha demostrado que siempre es posible seguirla, aunque a veces es necesario realizar un buen análisis previo.

- Definición de datos
- Recepción de los parámetros iniciales
- Preparación inicial de datos
- Presentación de un mapa con los datos iniciales, que sirva de interface para el tratamiento de la información.
- Validación de los datos especificados en el paso anterior.
- Tratamiento de los datos.

En **procesos batch** este esquema se puede modificar anteponiendo otro programa, de modo que el programa principal se desdoble en:

- un primer programa encargado de seleccionar los registros a tratar.
- Un segundo programa, invocado normalmente vía **FETCH RETURN** o **CALLNAT**, que será el encargado de procesar un único registro de los seleccionados por el programa inicial.

A modo de ejemplo se va a seguir la utilidad que acompañó el pasado mes al artículo, por ser una aplicación de uso general y además ser completa, sencilla y no necesitar especificaciones ya que se limita a listar un fuente desde el fichero ADABAS: **FUSER** donde el sistema guarda los distintos módulos. La clave de acceso a este fichero tiene una longitud de 18 posiciones, de las que las 8 primeras se redefinen como aplicación o librería, las 8 siguientes como objeto, y las 2 siguientes se reservan como secuenciador para aquellos módulos largos.

2.1 - Definición de datos

Las áreas de datos deben especificarse para contener solo las definiciones de aquellas variables que se usarán dentro de una cadena o transacción, y no para contener definiciones generales, que amplían innecesariamente la memoria a usar por los módulos que contengan dicha área. En este punto cabe recordar los comentarios y las recomendaciones hechas el pasado mes al hablar de las características de las Áreas de Datos, sobre todo en lo relativo a las **views** internas.

Por otra parte, y siempre con las miras de dar claridad al mantenimiento⁴, es muy útil utilizar distintos prefijos para referenciar las variables. Así, a modo de ejemplo, se exponen los siguientes:

- P-** para asociar a las variables que se reciben como parámetros
- W-** para asociar a las variables de trabajo (working)
- T-** para asociar a variables tipo tabla o array.
- G-** para asociar a variables del área de datos global.

Un error muy común consiste en incluir todos los campos de una vista en una área de datos local, ya que esto implica, no solo un aumento de la memoria a usar, sino que además complica la labor de ADABAS por tener que crear mas **IFB**⁵.

2.2 - Recepción de los parámetros iniciales

El siguiente punto del esquema definido para un objeto de tipo proceso, es el que se refiere a la recepción de parámetros iniciales. Este punto es opcional, y puede hacerse de varias formas, dependiendo del tipo de ejecutable que se esté tratando. Evidentemente, en el caso de no recibir datos iniciales, cuando se ejecuta este tipo de objeto, se debe examinar una situación, un tipo de registro, etc., ya que por definición, un programa elabora información en base a una entrada.

- En objetos tipo **Programa** solo puede hacerse la recepción de parámetros ejecutando la sentencia **INPUT**. Natural, al encontrarse con esta sentencia examina el área de usuario dedicada al **STACK** para ver si este contiene datos; y, en caso

⁴Esta idea, la de facilitar el mantenimiento, debe estar presente en cualquier desarrollo ya que ambas funciones no suelen realizarse por las mismas personas.

⁵IFB: Internal Format Buffer (ver primera parte del artículo).

de que así sea, no detiene la ejecución del programa, sino que asigna a cada una de las variables referenciadas en el **INPUT**, los distintos valores que se encuentren apilados en el **STACK**⁶.

```

UTILIDAD JMPDES <<<<<<< Utilidad para desarrollo >>>>>>> 13/05/96 18:11:20
Libreria: UTILIDAD Objeto: LISPGMOP
-----
0860 SET KEY PF5 NAMED 'MENU'
0870 INPUT NO ERASE (AD=I HE='LISPGMOH') MARK 2 AND SOUND ALARM
0880 08/19 '===== '
0890 09/19 '* <<<<<< LISTAR PROGRAMA >>>>>> LOP *'
0900 10/19 '===== '
0910 11/19 '* *'
0920 12/19 '* LIBRERIA : ' *OUTIN W-LIBRERIA-AC
0930 (HE='LISPGMLH') ' *'
0940 13/19 '* OBJETO : ' *OUTIN W-PROGRAMA-AC
0950 (HE='LISPGMZH',W-LIBRERIA-AC) ' *'
0960 14/19 '* *'
0970 15/19 '===== '
0980 16/19 '* <DEJE TODO A BLANCO PARA IR AL MENU> *'
0990 17/19 '===== '
1000 IF *PF-KEY = 'PF3'

Proxima pantalla listar desde 1010 Buscar: _____
-----
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10---PF11---PF12---
AYUDA SALIR LOCAL MAPA PRINT OBJ VIEW EXPLO <<--- --->>

```

Figura 1

La **figura 1** muestra un Hardcopy con el **INPUT** directo que pide el nombre de librería y el nombre del fuente a listar. Al objeto de hacer completo el ejemplo, se ha incluido una llamada a una HELPROUTINE simple (sirve para seleccionar la librería entre todas las de la instalación) , y otra llamada a la HELPROUTINE que permite seleccionar un programa de entre todos los de una librería, por lo que se debe pasar también el nombre de la librería como parámetro.

- En objetos tipo **Subrutina**, los datos iniciales pueden recibirse vía **STACK**, igual que en los de tipo programa; o definiendo las distintas variables dentro de un área de datos **parameter**. En este caso, no es necesario ejecutar ninguna sentencia para asociar los distintos valores a las variables ahí definidas, ya que cuando comienza la ejecución del modulo, los valores son cargados en las correspondientes variables. En cualquier caso, los valores pasan del modulo llamante al modulo llamado de la siguiente forma:

```
Perform Subrutina var_1 var_2 ...
```

- En Objetos tipo **Subprograma**, se pueden hacer las mismas consideraciones que en los objetos tipo subroutine. No obstante, este tipo de objetos tiene la particularidad de no compartir los datos de las áreas **globales**⁷, ya que en principio, este tipo de objetos se crea para descentralizar la ejecución de programas, permitiendo incluso que se corra el programa principal en una maquina, y el subprograma en otra. Este tipo de objetos normalmente es usado para definir funciones muy concretas y sencillas, que reciben datos y devuelven otros datos elaborados. La sentencia que se usa para invocar un subprograma tiene la forma siguiente:

```
Callnat Subprograma param_1 param_2 ...
```

La **figura 2** muestra un Hardcopy con la definición del área de parámetros correspondiente a la HELPROUTINE que ayuda a seleccionar el nombre del fuente.

⁶Ver líneas de comentario previas al Primer INPUT del LISPGM1P.

⁷En el caso de utilizar areas globales dentro de un subprograma, esta se inicializa cada vez que se invoca al subprograma.

```

UTILIDAD JMPDES <<<<<< Utilidad para desarrollo >>>>>> 14/05/96 09:27:18
Libreria: UTILIDAD Objeto: LISPCGM2H
-----
0010 *****
0020 *
0030 * APLICACION : INTERNACIONAL DESARROLLO
0040 * PROGRAMA : LISPCGM2H
0050 * OBJETIVO : LISTAR PROGRAMAS DE UNA LIBRERIA
0060 * AUTOR : JMP
0070 * FECHA : 23/08/94
0080 *
0090 *****
0100 *
0110 DEFINE DATA
0120 PARAMETER 1 P-LIBRERIA (A8)
0130 1 P-PROGRAMA (A8)
0140 LOCAL
0150 *

Proxima pantalla listar desde 160_ Buscar: _____
-----
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10---PF11---PF12---
AYUDA SALIR LOCAL MAPA PRINT OBJ VIEW EXPLO

```

Figura 2

La variable del sistema ***data** permite consultar el numero de entradas que se encuentran en la pila de datos del área de usuario. Esta variable puede consultarse por programa como cualquier otra variable, y en caso de que devuelva el valor **-1** significa que el stack contiene un comando en lugar de un dato.

```

UTILIDAD JMPDES <<<<<< Utilidad para desarrollo >>>>>> 14/05/96 09:28:00
Libreria: UTILIDAD Objeto: LL _____
-----
0110 DEFINE DATA LOCAL
0120 1 W-PROGRAMA (A8)
0130 END-DEFINE
0140 IF *DATA = 1
0150 INPUT W-PROGRAMA
0160 RELEASE STACK
0170 END-IF
0180 IF W-PROGRAMA = ' '
0190 MOVE '*' TO W-PROGRAMA
0200 END-IF
0210 STACK TOP DATA 'UTILIDAD' W-PROGRAMA
0220 FETCH 'LISPCGMOP'
0230 END

Proxima pantalla listar desde 0100 Buscar: _____
-----
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10---PF11---PF12---
AYUDA SALIR LOCAL MAPA PRINT OBJ VIEW EXPLO

```

Figura 3

La **figura 3** muestra el listado del programa correspondiente al comando **LL**⁸. Este comando solo es un nombre de programa abreviado que permite invocar al programa raíz de la utilidad, pero escribiendo menos.

Es de resaltar el hecho de que la pila o **STACK** solo contiene valores, no nombres de variables (continentes)

La pila o stack solo contiene valores no nombres de variables

En este apartado se puede comentar que en objetos tipo programa y subrutina, también puede recibirse datos utilizando el **GDA** (Área de Datos Global) para recoger datos del programa llamante, pero no es aconsejable ya que elimina la claridad de mantenimiento del objeto pues se requerirá un estudio de todo el programa para conocer cuales son los datos iniciales que sirven de parámetros.

⁸Se ha optado por llamarle **LL** ya que **L** es la abreviatura del comando del sistema que lista un fuente.

2.3 - Preparación inicial de datos

Este punto del esquema, opcional, no suele ser un punto complejo, pues simplemente tiene por objeto completar, en base a los parámetros iniciales, los datos que se presentaran al usuario para su tratamiento en el punto siguiente del esquema.

Los objetos externos que puede incluir este punto son generalmente llamadas para descodificar algún valor, cosa que puede hacerse vía **Perform** o **CALLNAT**, para así poder presentar estos valores en el siguiente apartado.

Para el caso que nos ocupa, en el programa **LISPGMOP**, este punto se corresponde con la rutina encargada de determinar el tipo de objeto al que pertenece el fuente; y, en el programa **LISPGMIP** este punto se corresponde con la rutina que carga la tabla de números de línea y de texto.

2.4 - Presentación de un mapa con los datos iniciales, que sirva de interface para el tratamiento de la información.

Este punto del esquema suele invocar a otro objeto ejecutable de tipo **MAPA** mediante la sentencia:

```
INPUT using MAP nombre_mapa
```

Es un tipo especial, en el sentido de que este tipo además de servir de interface ante el usuario, es la cara por la que el usuario va a conocer la aplicación. Este objeto esta adornado con todo tipo de facilidades como ya se ha visto, tales como reglas de proceso, **helprutinas** y **helptextos**.

La **figura 4** muestra cómo asociar la HELPROUTINE⁹ que permite seleccionar un fuente de entre todos los de una librería, en un mapa haciendo uso del atributo **HE**.

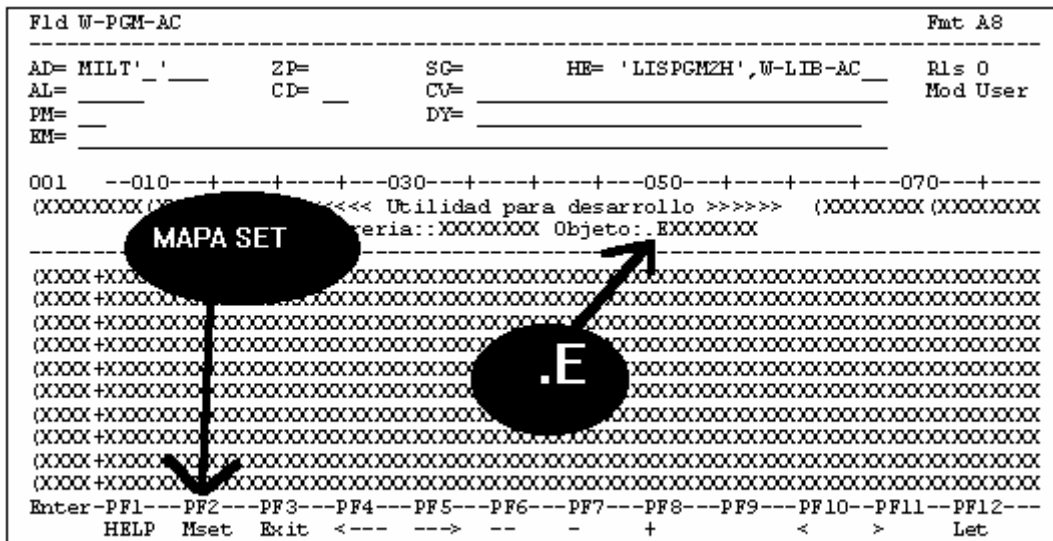


Figura 4

Antes de invocar al mapa se deben definir las **Pfs** que se usaran, o desactivaran, etc.; e inmediatamente después de invocarle, se puede incluir el proceso de examinar la PF pulsada por el usuario, a fin de eliminar validaciones innecesarias en el caso de que éste decida cancelar el proceso.

Para asociar un mapa de ayuda a un mapa concreto, se especifica éste en el campo **HELP** del **mapa set** de la figura 8. Y, para activar la llamada a este mapa, debe

⁹Para invocar a una helproutine, no es suficiente con que el campo contenga una interrogacion "?" sino que se debe escribir la interrogacion, y, estando el cursor sobre el campo en cuestion, pulsar la tecla **INTRO**

programarse una PF (normalmente la PF1) con la palabra HELP (sin encerrar entre comillas), tal y como se indica a continuación:

```
SET KEY PF1 = HELP NAMED 'AYUDA'
```

De esta forma, dicho mapa de ayuda es invocado cuando se pulse dicha PF.

La **figura 5** muestra cómo el tipo de objeto **MAPA** cuenta con un área de parámetros, por lo que el paso de datos al mapa puede hacerse de :

Forma implícita: sin enumerar los parámetros a pasar. En este caso todos los nombres de las variables del área de parámetros del mapa deben existir y coincidir en nombre y formato con los del modulo llamante.

Forma explícita: Enumerando a continuación del nombre del mapa en el programa llamante, todas las variables del área de parámetros del mapa. En este caso, solo deben coincidir en el orden los formatos de las variables pasadas.

```

UTILIDAD JMPDES <<<<<< Utilidad para desarrollo >>>>>> 14/05/96 09:29:21
Libreria: UTILIDAD Objeto: LISPGMIM
-----
0001 * MAP2: PROTOTYPE
0002 * INPUT USING MAP 'XXXXXXXX'
0003 * T-LINEA(*) T-NUMERO(*) W-ARC-SCAN W-LIB-AC W-PGM-AC
0004 * W-PROXIMA-LINEA
0005 DEFINE DATA PARAMETER
0006 1 T-LINEA (A080/00001:00020)
0007 1 T-NUMERO (&004/00001:00020)
0008 1 W-ARC-SCAN (A020)
0009 1 W-LIB-AC (&008)
0010 1 W-PGM-AC (&008)
0011 1 W-PROXIMA-LINEA (NO4,0)
0012 END-DEFINE
0013 FORMAT PS=021 LS=101 ZP=OFF SG=OFF KD=ON IP=OFF
0014 * MAP2: MAP PROFILES ***** 200*****
0015 * ,TTAAAMMOO D I D I N D I D I ?_ )^&:+( 'LISPGMOH' *

Proxima pantalla listar desde 16_ Buscar: _____
-----
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10---PF11---PF12---
AYUDA SALIR LOCAL MAPA PRINT OBJ VIEW EXPLO

```

Figura 5

Otra consideración importante es la que se refiere a la ubicación del mapa. Este debe figurar en el programa principal, y no dentro de una subrutina interna, ya que de hacerlo así, cuando se ejecute la sentencia **REINPUT**¹⁰ por no haber superado una validación, si ya se ha salido de la subrutina que contiene el mapa, el sistema devuelve un error por no saber como acceder a él.

En el ejemplo, el programa **LISPGMOP** este punto se corresponde con la secuencia vacía.

2.5 - Validación de los datos especificados en el paso anterior.

Una vez captada la información mediante el mapa, y antes de iniciar la elaboración de la misma, se debe efectuar la validación de datos, a fin de evitar el deshacer todos los pasos realizados por culpa de no superar una validación tardía.

En este sentido, se debe tener presente dos instrucciones muy importantes en este entorno:

- **REINPUT** : retorna el flujo de proceso al ultimo **INPUT** ejecutado, que en el caso de que siga el esquema planteado seria el mapa, pero no deshace los cambios efectuados en los valores de las variables. Es decir si se debe actualizar algún contador, esta actualización debe ejecutarse una vez que se hayan superado todas las validaciones, pues en caso contrario su valor seria engañoso.

¹⁰La sentencia **REINPUT** cede el control del programa al ultimo **INPUT** ejecutado.

- **BACKOUT TRANSACTION** deshace todos los cambios realizados en la base de datos y libera todos los registros que se encuentren en **HOLD**¹¹. No obstante, esta sentencia no debería figurar en esta parte del programa, pues de acuerdo con el primer párrafo, solo se validan datos, no se actualizan.

2.6 - Tratamiento de los datos.

Una vez validados los datos, se debe proceder a elaborar definitivamente la información. Este punto es muy variado, y, dependiendo de que se trate de programas de consulta o de actualización podrá contener procesos mas o menos complejos.

Tanto en este paso del esquema como en el de las validaciones previas, se puede invocar a otros procesos, bien para validar, bien para obtener datos elementales en base a parámetros muy específicos. Así, por ejemplo, para validar el código de una divisa se invocaría a un subprograma, al cual se le pasaría como parámetros el código de divisa y un continente para que devuelva si es valido o no.

La **figura 6** muestra los objetos referenciados en el fuente de una aplicación operativa mediante la utilidad LISPGM7P que acompaña al artículo.

UTILIDAD JMPDES <<<<<<< Utilidad para desarrollo >>>>>> 13/05/96 18:13:30					
Libreria: UTILIDAD Objeto: LISPGM1P					
Programas invocados	Mapas	Subrutinas	Subprogram	Areas D.	Copycode
MENU	LISPGM1M				LISPGMER
LISPGM4P					
LISPGM5P					
LISPGM6P					
LISPGM7P					
LISPGM8P					

Enter	PF1	PF2	PF3	PF4	PF5	PF6	PF7	PF8	PF9	PF10	PF11	PF12
AYUDA	SALIR			P:LP	M:LMP	V:LVP				<<---	--->	

Figura 6

La diferencia entre utilizar objetos tipo subprograma o subrutina, estriba básicamente en dos hechos:

- 1.- Una **subrutina** comparte área global con el programa llamante, y además debe correr en la misma maquina en la que se esta ejecutando el programa llamador.
- 2.- Un **subprograma** puede correrse en una maquina distinta a la del programa llamante, y como consecuencia no comparte datos globales. En el caso de que se ejecuten ambos objetos en la misma maquina, y el área global que utilizaran fuera la misma, seria considerada como distinta, siendo inicializada en el subprograma cada vez que fuera invocado este.

Así, en el ejemplo anterior, si el código de divisa dependiera del código de banco, y este se encontrara cargado en el área global del programa principal, podría utilizarse una subrutina externa a fin de que ese dato global le recogiera del área. Pero también podría usarse un subprograma, solo que en este caso los parámetros a pasar serían código de banco y código de divisa, así como un continente para que devolviera el resultado de la validación.

Por ultimo, cabe reseñar que también se puede invocar un objeto tipo **programa** para hacer una validación o elaborar un dato elemental. Esta llamada se hace utilizando la sentencia **FETCH RETURN**. Esta opción se diferencia de las anteriores en:

¹¹Un registro se dice que se encuentra en **HOLD** cuando su lectura *puede suponer* una actualización del mismo, con independencia que se cumplan o no las condiciones para su actualización. Cuando un registro se encuentra en este estado, no puede ser actualizado por otro usuario, devolviéndole a este ultimo el error 3145.

- Los datos no son recogidos en el 'programa llamado' en un área de datos **parameter**, tal y como se podía realizar en los otros dos tipos, sino que se recogen vía **input** a través del **stack** como se comentó al hablar de las áreas de datos.
- Se comparte área de datos global, siendo esta otra forma de pasar datos al 'programa llamado'.
- La devolución de datos al programa llamante se debe realizar vía datos globales o vía **Stack**.

Como norma general se debe perseguir que los módulos sean lo mas pequeños posible ya que esto facilita su mantenimiento. A la hora de elegir el tipo de objeto a utilizar se debe tener en cuenta las características de cada tipo, y en caso de ambigüedad, elegir, de acuerdo con el cuadro de la **figura 7**, aquel que represente menor coste de llamada.



Figura 7 : Coste de una llamada

2.7 Procesos que necesitan pantalla de confirmación

Hay instalaciones en las que se requiere que antes de grabar datos se presente una pantalla con los datos elaborados y protegidos para solicitar al usuario que confirme la operación que se desea realizar.

Este requerimiento sigue ajustándose perfectamente al esquema dado, solo que en este caso, se debe:

09:31:00				Define Map Settings for MAP		96-05-14			
Delimiters				Format		Context			
Cls	Att	CD	Del	Page Size 23	Device Check _____		
T	D		BLANK	Line Size 100	WRITE Statement	_____		
T	I		?	Column Shift	... 0 (0/1)	INPUT Statement	... X		
A	D			Layout	Help 'LISPGMDH'	_____		
A	I)	dynamic N (Y/N)	as field default	N (Y/N)		
A	N		^	Zero Print N (Y/N)				
M	D		&	Case Default	... UC (UC/LC)				
M	I		:	Manual Skip	... N (Y/N)	Automatic Rule Rank	1		
O	D		+	Decimal Char	... ,	Profile Name SYSPROF		
O	I		(Standard Keys	.. Y (Y/N)				
				Justification	.. L (L/R)	Filler Characters	_____		
				Print Mode _	Optional, Partial _		
				Control Var _____	Required, Partial _		
						Optional, Complete	... _		
						Required, Complete	... _		
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10---PF11---PF12---									
Help				Exit				Let	

Figura 8

- Hacer uso de una variable de control para proteger los campos.
- Para poder hacer uso de la variable de control, ésta debe estar definida tanto en el modulo llamante como en el mapa. Esta variable, si se desea que afecte a todo el mapa, se definirá en el **mapa set**, o mapa del sistema que sirve de interface para especificar los atributos de un mapa de usuario (se accede a el pul-

mediante **REINPUT**, por lo que solo se llegara a activar **PF10** cuando se superen todas las validaciones. En este caso, además se asigna el valor de protección a la variable de control, por lo que cuando vuelva a aparecer el mapa, saldrán todos los campos protegidos, y, en caso de que se pulse **PF10**, ya no es necesario repetir las validaciones, sino que solo hay que realizar el tratamiento propio de la transacción.

- Hacer uso de la sentencias **END TRANSACTION** para pasar a definitivas las modificaciones realizadas en la base de datos, o de **BACKOUT TRANSACTION** para deshacerlas.

3. Los Objetos NATURAL y la POO

Por ultimo, el autor no quisiera terminar el articulo sin relacionar los objetos NATURAL con la **Programación Orientada a Objetos** tan en boga hoy en día. Por tanto, este apartado figura únicamente para establecer diferencias y evitar confusiones.

Así, desde el punto de vista del autor, en lugar de **OBJETOS NATURAL**, este articulo debería haberse titulado **TIPOS DE MÓDULOS** en NATURAL con lo que se evitaría relacionar los objetos vistos con los objetos de la **POO**, pero ha preferido usar la terminología del entorno para evitar confusiones y aclarar ahora que en el entorno NATURAL se entiende por **objeto** el código ejecutable que resulta de compilar un fuente; mientras que en cualquier entorno de sistemas, se entiende por **modulo** un 'trozo' de codificación en lenguaje maquina, pero no totalmente resuelto, y por tanto no directamente ejecutable. Por decirlo de otra forma, un modulo siempre forma parte de algo mayor, pero en sí, no es algo completo.

Por otra parte, si bien es verdad que una cosa es la **modularidad** y otra la programación orientada a objetos, también es verdad que entre ambas solo hay un paso, y NATURAL tiene dado un paso de gigante para llegar a la POO pues NATURAL realmente es un *programa principal* que ejecuta comandos del sistema y comandos creados por el usuario, con lo que en cualquier caso ya se ha interpuesto la capa necesaria para la **POO** entre el usuario y los datos. Ahora solo habría que hacer que los datos fueran sensibles a cualquier evento externo para que se ejecutara cualquier modulo de los comentados, pero esto, como el lector comprenderá, se sale de los objetivos de este tema.

4. DIAGRAMA DE FLUJO

La regla *Un programa un mapa*, permite discriminar los objetos de tipo programa ya que estos serán los únicos que incluirán llamadas a mapas, aunque no todos los mapas tienen que incluir llamadas a mapas (Ejemplo: programas Batch).

Para transacciones On-line, al seguir esta regla, se pueden usar como documentación los esquemas modulares o diagramas de flujo con la simbología expresada en la **figura 11**.

Hay que tener en cuenta, que si bien en cada librería se encuentran todos los módulos de una aplicación, *gestión de almacén* por ejemplo, ésta se compone de muchas **transacciones**, entendiéndose por tal, un conjunto de objetos que permiten realizar una acción muy concreta dentro de una aplicación. Así, dentro de la aplicación de Almacén, una transacción seria *Recepción de productos*, la cual a su vez puede tener distintas opciones.

Otra consideración a tener en cuenta se centra en la **nomenclatura** utilizada por casi todas las instalaciones, mediante la cual se asigna el nombre de la transacción a los 4 0 5 caracteres iniciales del nombre de los objetos; y reserva, de los 4 0 3 caracteres restantes, una posición fija para el carácter asociado al tipo de objeto, de acuerdo con la tabla de la **figura 1** del pasado mes.

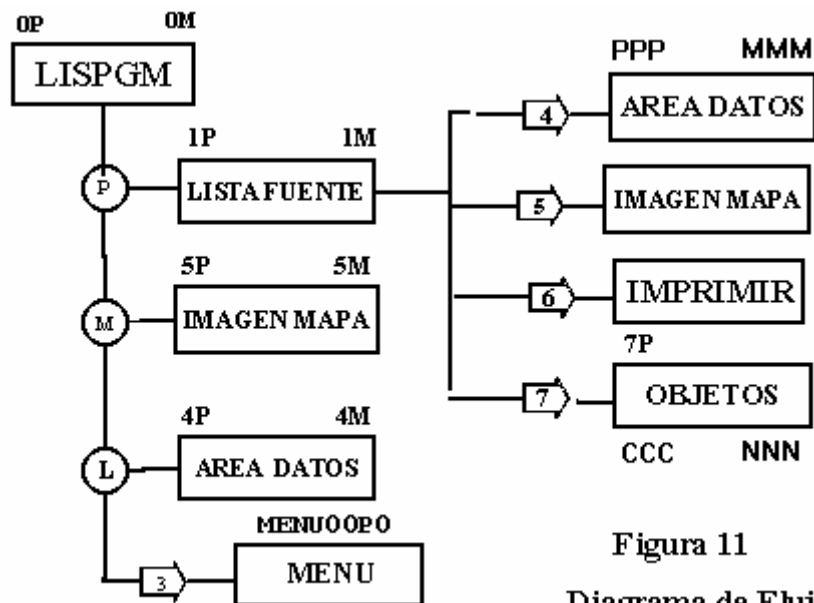


Figura 11
Diagrama de Flujo

Pues bien, en el rectángulo superior, o raíz del árbol jerárquico que representa, figurará el nombre de la transacción; y, como generalmente cada transacción cuenta con un menú inicial, de la raíz reseñada, saldrá una línea con las distintas opciones de dicho menú encerradas en un círculo.

A la derecha de estas opciones, otras líneas conducen a los rectángulos que representan los programas de cada una de ellas. Dentro de este rectángulo se describirá en 2 o 3 palabras la función del módulo (Alta Pedido), y en los distintos ángulos, los sufijos que identifican los distintos módulos que acompañan a ese programa, tal y como muestra la figura.

De cada uno de estos módulos de nivel inferior, también pueden salir nuevas opciones, por lo que, si esto fuera así, estas se reflejarían en el correspondiente diagrama.

Cuando la bifurcación proviniera de una PF, esta se representaría como una opción de menú, con la salvedad de ir encerrada dentro de un triángulo el número de la PF que conduce a otro módulo.

Los comentarios a la **figura 11** son los siguientes:

(Opción) : representa las distintas opciones en el caso de que la función presente un menú o lista seleccionable con distintas opciones.

PPP - es el nombre del programa que se ejecuta puede ser Programa, Subprograma o Subrutina, pero siempre es el nombre del módulo principal de la función.

MMM - es el nombre del mapa que es llamado en ese programa. Solo se puede asociar un mapa a una función.

NNN - es el nombre de los subprogramas invocados o que se ejecutan en otra máquina.

CCC - es el nombre de los COPYCODES que se encuentran incluidos en el fuente del módulo principal de la función. En el caso de que hubiera varios, se especificarían uno debajo de otro.

5. BUFFERS de NATURAL

Como anexo al artículo, y con objeto de ampliar conceptos se añade este apartado con la descripción de los distintos buffers usados por NATURAL.

A la hora de diseñar una aplicación es importante tener en cuenta los buffers que se definen en NATURAL por cada sesión, ya que existe una serie de áreas especiales o "threads" que son utilizadas por NATURAL de forma diferente, según se encuentre en tiempo de edición, compilación y ejecución.

- **USIZE** : este parámetro, con un tamaño que oscila entre 8 y 32K, determina el tamaño del **USER ÁREA**, es decir del área dedicada:

en la versión 1.2 : a la ejecución de los programas NATURAL;

en la versión 2.1 :

- en tiempo de compilación: áreas de datos y tablas de funciones necesarias para la compilación;

- en tiempo de ejecución: Áreas de datos locales (LDA); tabla de bucles; área de stack del CALLNAT.

- **ESIZE** : Este parámetro establece el tamaño del **EXTENDED USER ÁREA**, pudiendo tomar los valores de 1 a 64K, y esta dedicada a contener:

- Área de edición de fuentes;

- Tabla de asignación de teclas de función

- Stacks de subrutinas

- Áreas de datos Globales

- Stack de comandos

- **FSIZE** : Este parámetro establece el tamaño del **FILE BUFFER**, pudiendo tomar los valores de 2 a 64K, y esta dedicada a contener:

- DDMs durante la compilación

- Tabla de símbolos.

- **DSIZE** : Este parámetro establece el tamaño del **DEBUG BUFFER**, pudiendo tomar los valores de 1 a 64K, y esta dedicada a contener la salida de la utilidad ADA-LOG.

- **SORTSIZE** : Este parámetro establece el tamaño del **SORT BUFFER**, pudiendo tomar los valores de 0 a 64K, y esta dedicada a ser el almacenamiento temporal para realizar el sort en procesos online.

Concepto de **Buffer Pool**: Cuando un usuario entra en el sistema, se le dedica un buffer para cada uno de los tipos reseñados, tomándoles de un espacio de memoria dedicado a estos menesteres. Este Buffer Pool tiene su espacio de memoria troceado en paginas de 2 o 4Kb.

En principio, el tamaño dedicado al USIZE es de **3500** Bytes, pero a medida que se vayan invocando objetos, este tamaño se va incrementando en el numero de bytes que ocupa cada uno de los objetos, liberándose el espacio dedicado a cada uno de ellos, a medida que el proceso abandona cada modulo.

Cada vez que un modulo, del tipo que sea, invoca a un programa haciendo uso de la sentencia **FETCH**¹², se libera toda la memoria dedicada al **USIZE**, volviendo a tomar el valor de 3500 Bytes, mas el tamaño del programa invocado, como se aprecia en la **figura 12**.

¹² Esta regla no es valida si se ejecuta **FETCH RETURN**

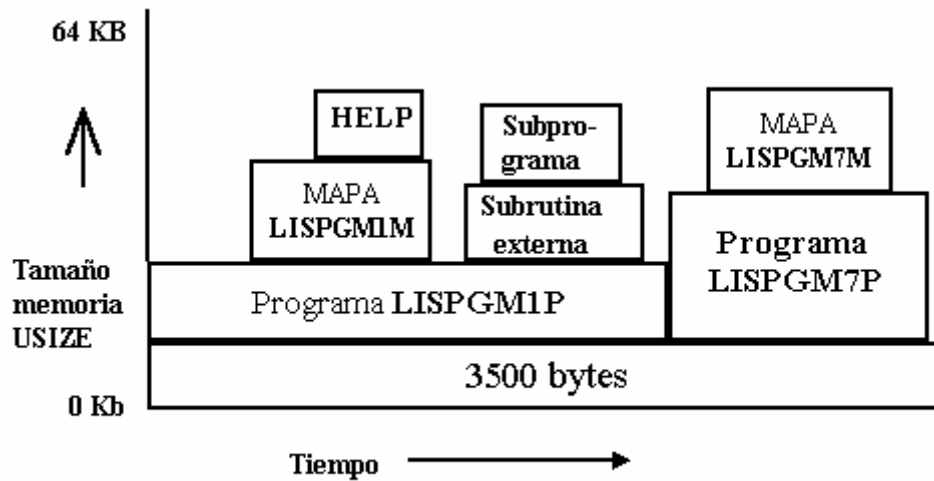


Figura 12 : Memoria dedicada al USIZE

6. Utilidad

La utilidad que acompaña a este artículo, complementa la dada el pasado mes. Es una herramienta muy fácil de desarrollar, prácticamente los fuentes vienen distribuidos con el disco que acompaña a la revista, y solo hay que sustituir el nombre de la view **SYSTEM-FUSER** por el nombre de la DDM que apunta al fichero **FUSER** del sistema, que en principio coincidirá con el de la utilidad.

Las ventajas son:

- Lista cualquier fuente de cualquier librería, saltándose el control de NATURAL-SECURITY
- Permite buscar cadenas dentro de fuentes.
- Con **PF7** se relacionan los objetos referenciados en un objeto.
- Su autor la califica como herramienta imprescindible en cualquier centro de desarrollo, por su potencia, sencillez de uso, y sobre todo para funciones de mantenimiento.

El Fichero **LISPGM.NAT** incluido en el disquete que acompaña a la revista contiene el listado formateado de los distintos fuentes de la utilidad.

Notas A pie de pagina

- 1.- En informática es habitual, aunque no académico, el que un programa pueda abandonarse por múltiples puntos. Por ejemplo usando la sentencia **FETCH**.
- 2.- Se entiende por tratamiento pesado aquel que tiene un gran numero de accesos a bases de datos y/o ficheros.
- 3.-Esta regla ha sido llevada a la practica por el autor de este articulo, y ha demostrado que siempre es posible seguirla, aunque a veces es necesario realizar un buen análisis previo.
- 4.- Esta idea, la de facilitar el mantenimiento, debe estar presente en cualquier desarrollo ya que ambas funciones no suelen realizarse por las mismas personas.
- 5.- IFB: Internal Format Buffer (ver primera parte del articulo).
- 6.- Ver líneas de comentario previas al Primer INPUT del LISPGM1P.
- 7.- En el caso de utilizar áreas globales dentro de un subprograma, esta se inicializa cada vez que se invoca al subprograma.
- 8.- Se ha optado por llamarle **LL** ya que **L** es la abreviatura del comando del sistema que lista un fuente.
- 9.- Para invocar a una helproutine, no es suficiente con que el campo contenga una interrogación “?” sino que se debe escribir la interrogación , y, estando el cursor sobre el campo en cuestión, pulsar la tecla **INTRO**
- 10.- La sentencia **REINPUT** cede el control del programa al último **INPUT** ejecutado.
- 11.- Un registro se dice que se encuentra en **HOLD** cuando su lectura *puede suponer* una actualización del mismo, con independencia que se cumplan o no las condiciones para su actualización. Cuando un registro se encuentra en este estado, no puede ser actualizado por otro usuario, devolviéndole a este último el error 3145.
- 12.- Esta regla no es valida si se ejecuta **FETCH RETURN**