

Fundamentos de Ingeniería del Software



Capítulo 4. Diseño



“Hay dos formas de realizar un diseño: una es hacerlo tan simple que obviamente no haya deficiencias; la otra es hacerlo tan complicado que no haya deficiencias obvias”

C.A.R. Hoare

Diseño. Estructura



1. El proceso de diseño.
2. Modelos de diseño.
3. Diseño estructurado.
 - Diagramas de estructura.
 - Estrategias de diseño:
 - Análisis de transformaciones.
 - Análisis de transacciones.
4. Métricas de calidad estructural.
 - Acoplamiento.
 - Cohesión.
5. Heurísticas de diseño estructural.

Diseño. Bibliografía



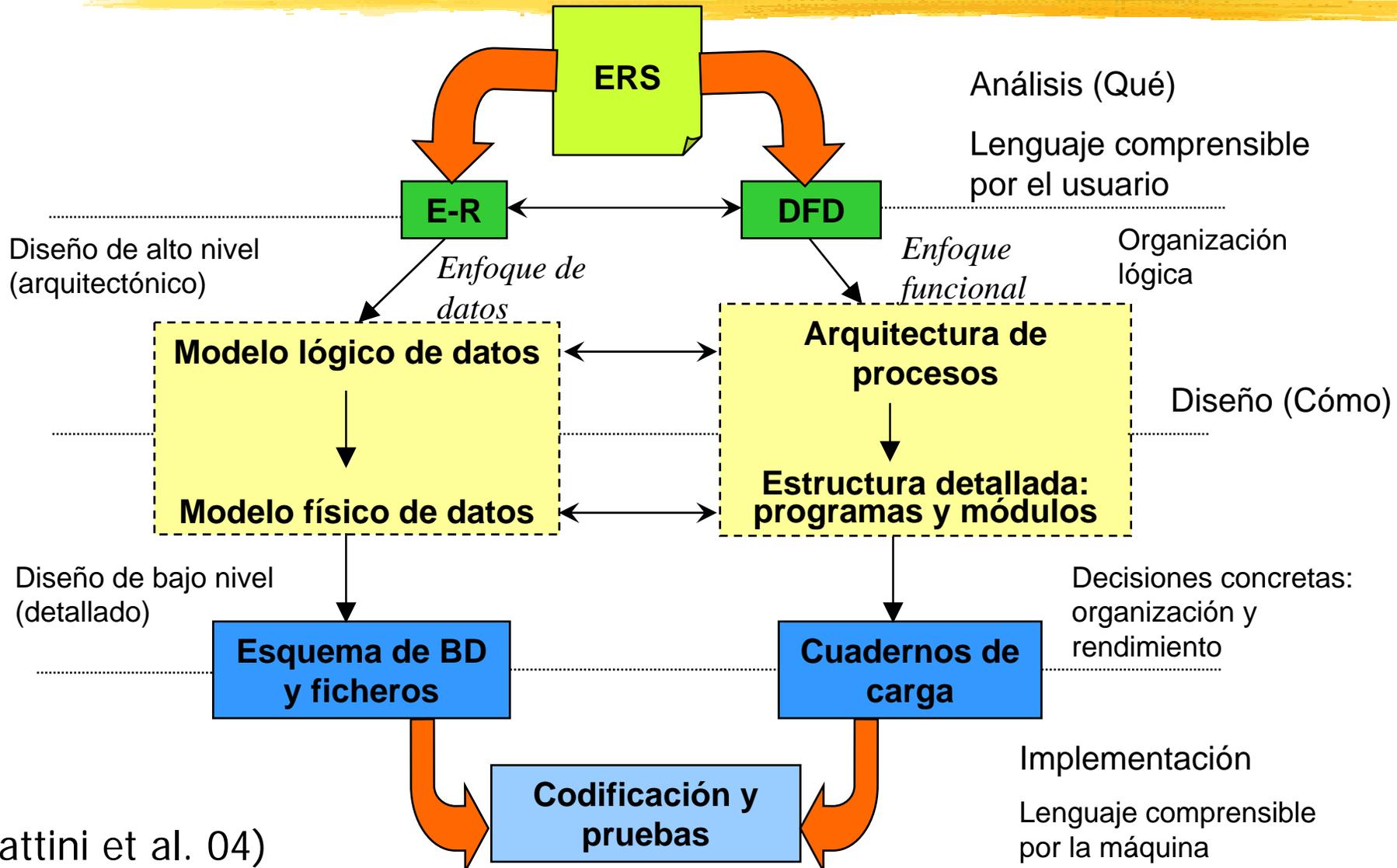
- (Piattini et *al.* 96) (Piattini et *al.* 04) Capítulo 8. Apartado 8.1.
- (Molina et *al.* 97) A. Molina, P. Letelier, P. Sánchez, J. Sánchez. "Metodología y Tecnología de la Programación". Servicio de Publicaciones. UPV. 1997.
- (Pressman 06) Capítulo 9 (resumido) y aptdo. 10.6.
- (Pressman 02) Capítulo 13 y aptdos. 14.5 a 14.8.
- (MAP 95) Ministerio de Administraciones Públicas. Guía de Técnicas de Métrica v.2.1. 1995.
- (MAP 01) Guía de técnicas y prácticas de Métrica v.3.
<http://www.csi.map.es/>
- (Page-Jones 88) M. Page-Jones. "The Practical Guide to Structured Systems Design". Yourdon Press. 1988.

1. *El proceso de diseño*



- “El proceso de aplicar distintas técnicas y principios con el propósito de definir un dispositivo, un proceso o un sistema con suficiente detalle como para permitir su realización física”.
- Proceso iterativo a través del cual se traducen los requisitos en una representación del software.
- La calidad sólo se puede conseguir a través de un proceso de diseño.

El proceso de diseño (II)



El proceso de diseño (III)



Cada modelo se asienta en el anterior:

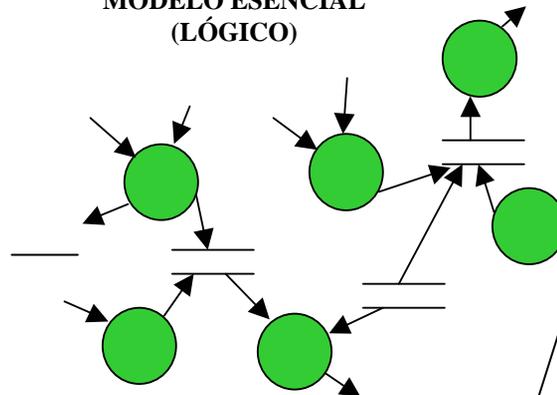
- **Diseño de datos.** Transforma el modelo del dominio de la información del análisis en las estructuras de datos necesarias para la implementación.
- **Diseño arquitectónico.** Estructura modular del programa/aplicación.
- **Diseño de interfaz.** Interfaces del sw. con otros sistemas y con los usuarios.
- **Diseño procedimental.** Descripción procedimental de los componentes del sw.



2. Modelos de diseño (Yourdon 93)

Análisis

MODELO ESENCIAL
(LÓGICO)

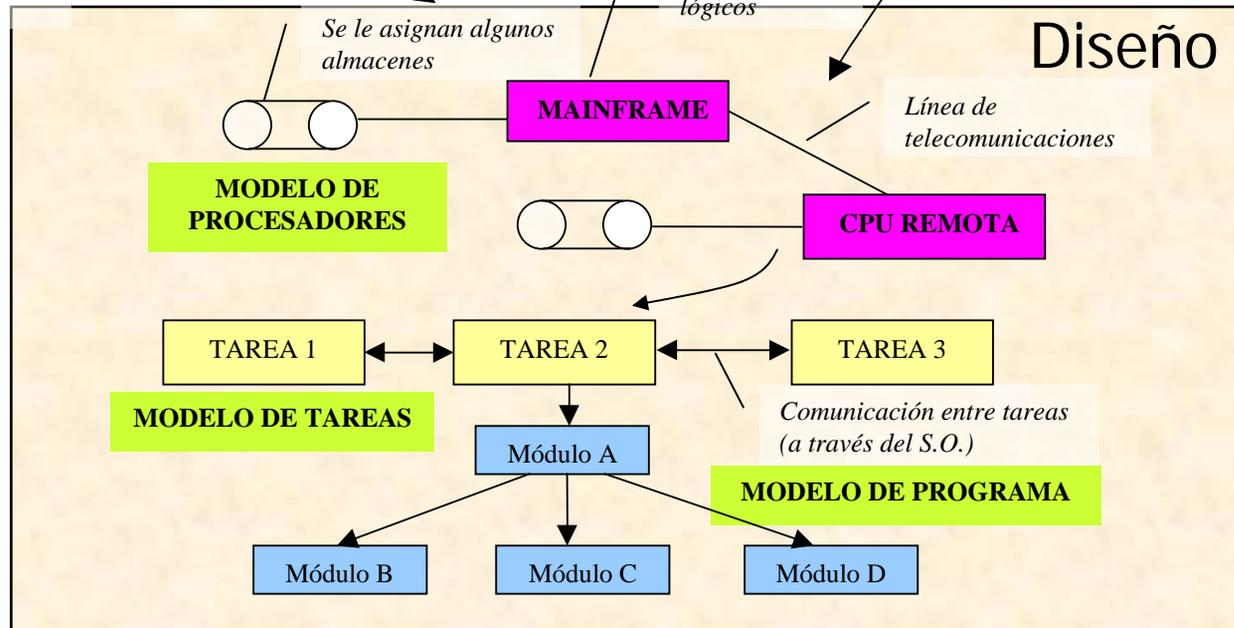


En la última etapa del análisis se ha establecido el límite de automatización del sistema

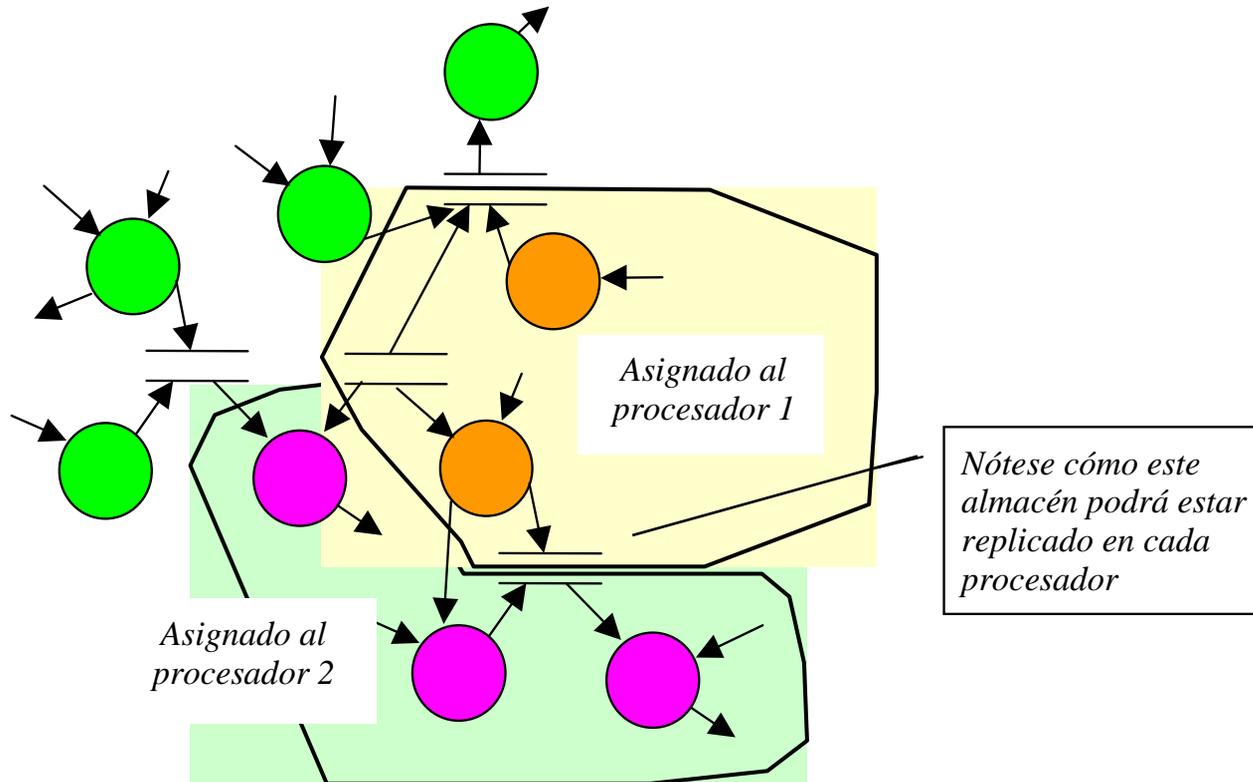
Se le asignan algunos procesos lógicos

Se le asignan algunos almacenes

Diseño



Asignación de procesos y almacenes a procesadores (Yourdon 93)



3. *Diseño estructurado*



- **Objetivos:**
 - Desarrollar la estructura modular del programa
 - Definir las relaciones entre módulos
- **Técnica ppal.: Diagrama de Estructura**
 - Pseudocódigo
 - Árboles de decisión
- **Documentación de partida: DFDs**
- **Estrategias de diseño:**
 - Análisis de transformaciones
 - Análisis de transacciones

Diseño estructurado



- Se dispone de:
 - Las entradas que suministran al sistema las entidades externas.
 - Las salidas aportadas por el sistema a dichas entidades externas.
 - Las funciones descompuestas que se han de realizar en ese sistema.
 - El esquema lógico de datos del sistema.

Diseño estructurado (II)



- Tareas a realizar:
 - Determinar qué módulos implantarán los procesos terminales (primitivos) obtenidos en el análisis.
 - Organizar la estructura de estos módulos y definir las conexiones entre los mismos.
 - Describir el pseudocódigo para cada módulo.
- Se basa en:
 - abstracción
 - modularidad
 - encapsulamiento y ocultamiento de información
- Añade un nivel de abstracción a la programación estructurada.

Diagrama de estructura (Diagrama de estructura de cuadros de Constantine)



- Diseño arquitectónico del sistema:
diagrama de módulos funcionales
 - Identifica qué módulos se necesitan, así como sus inputs/outputs (caja negra)
 - Refleja la comunicación de datos y control y la jerarquía entre módulos
- Diagrama de estructura:
 - Módulos
 - Conexiones
 - Comunicaciones

Diagrama de estructura.

Módulo



- “Aquella parte de código que se puede llamar” (Page-Jones 88).
- Representa un programa, subprograma, procedimiento, función o rutina, dependiendo del lenguaje de implementación que se vaya a utilizar.
- El diseño estructurado NO impone la restricción de que un módulo tenga que ser compilado independientemente.
- Admite parámetros de llamada y retorna algún valor, si es preciso.
- Tamaño ideal: 40-50 líneas
 - ¡pero hay muchas opiniones!
- Se representa en el diagrama mediante un rectángulo.

DE. Representación de módulos

MÓDULO

OBTENER
DATOS
CLIENTES

Módulo normal.

MÓDULO
PREDEFINIDO

IMPRIMIR
CHEQUE DE
PAGO

Un módulo predefinido está disponible en la biblioteca del sistema o de la propia aplicación (a veces, puede ser también un módulo de un sistema operativo o de un SGBD), y por tanto, no es necesario codificarlo.

CONECTOR

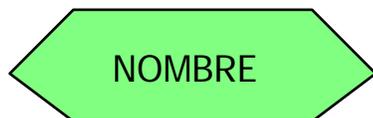
1

Para no perder la referencia, el diagrama de estructura debe dibujarse en una hoja tamaño A4. Si no cabe (bien), se deben explosionar los elementos (como en System Architect) o poner conectores.

DE. Representación de módulos (II)

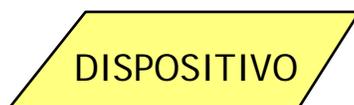
En la notación de Métrica típica también se dispone de:

Almacenes de datos



módulos que representan dónde van a estar físicamente los datos (tablas, ficheros)

Dispositivos físicos

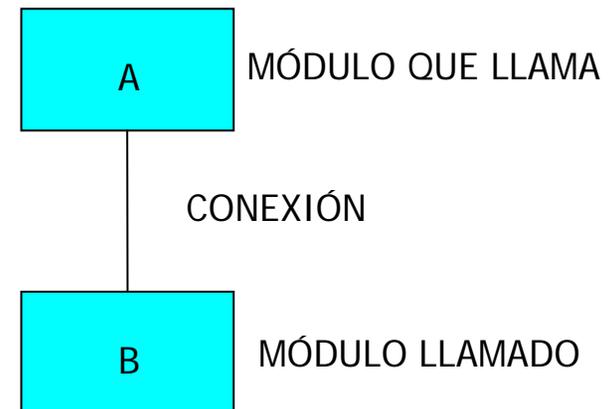


dispositivos (de cualquier tipo) por donde se puede recibir o enviar información que necesite el sistema.

Diagrama de estructura.

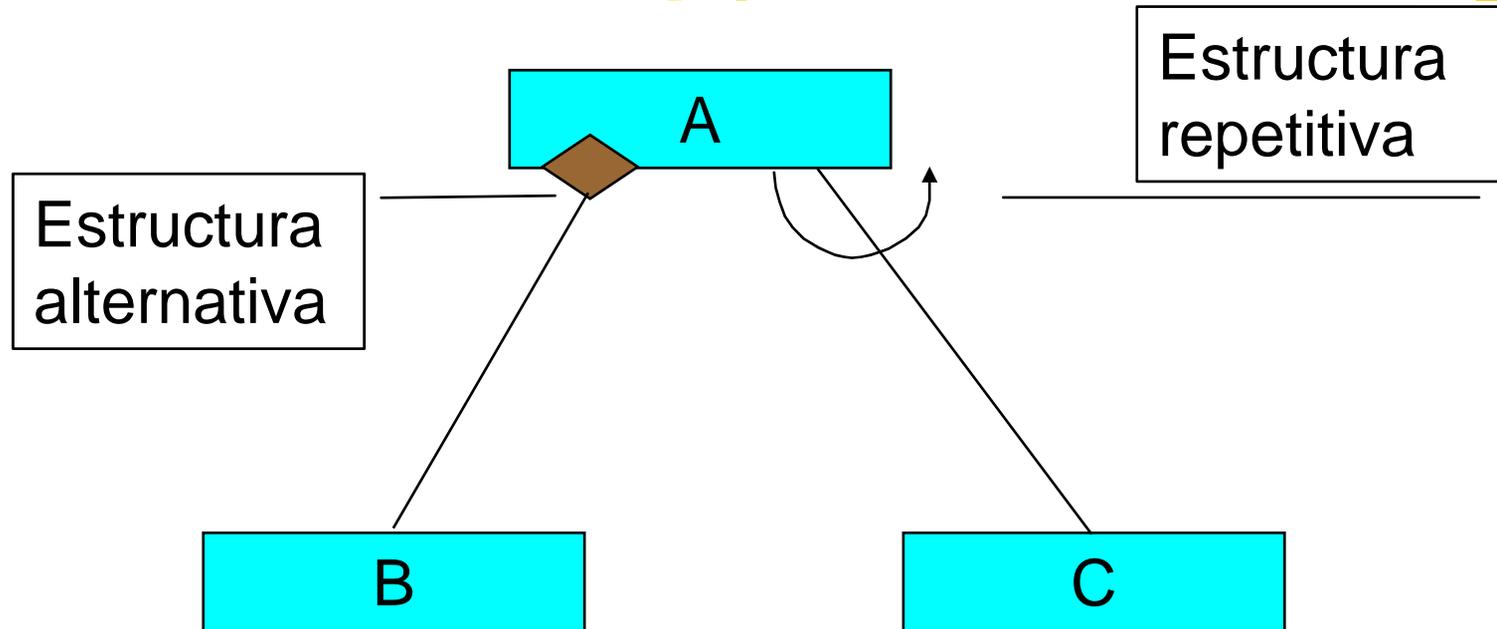
Conexión entre módulos

- La conexión entre módulos se representa mediante una línea.
- En la figura:
 - A llama a B.
 - B hace su función.
 - B retorna a A, inmediatamente después del lugar donde se produjo la llamada de A a B.
- El diagrama no dice nada sobre el código de A ni sobre el de B, lo único que sabe es que en A existe una sentencia del tipo CALL B.
- El diagrama no dice cuántas veces puede A invocar a B. Sólo se dice que A es capaz de invocar a B. Tal vez no realice ninguna invocación.



- Tampoco muestra detalles internos de los módulos como código, algoritmos o datos.

Diagrama de estructura. Conexión entre módulos (II)



Orden de ejecución de los módulos: de izquierda a derecha y de arriba abajo (Piattini et al. 04)

⇒ Según (Molina et al. 97) el orden no importa:

“El diagrama de estructura no representa aspectos procedurales del sistema como las secuencias, alternativas o bucles”

Diagrama de estructura. Conexión entre módulos (III)

Ejemplo típico de aplicación interactiva:

⇒ un conjunto de opciones de menú que se escogen cíclicamente por el usuario

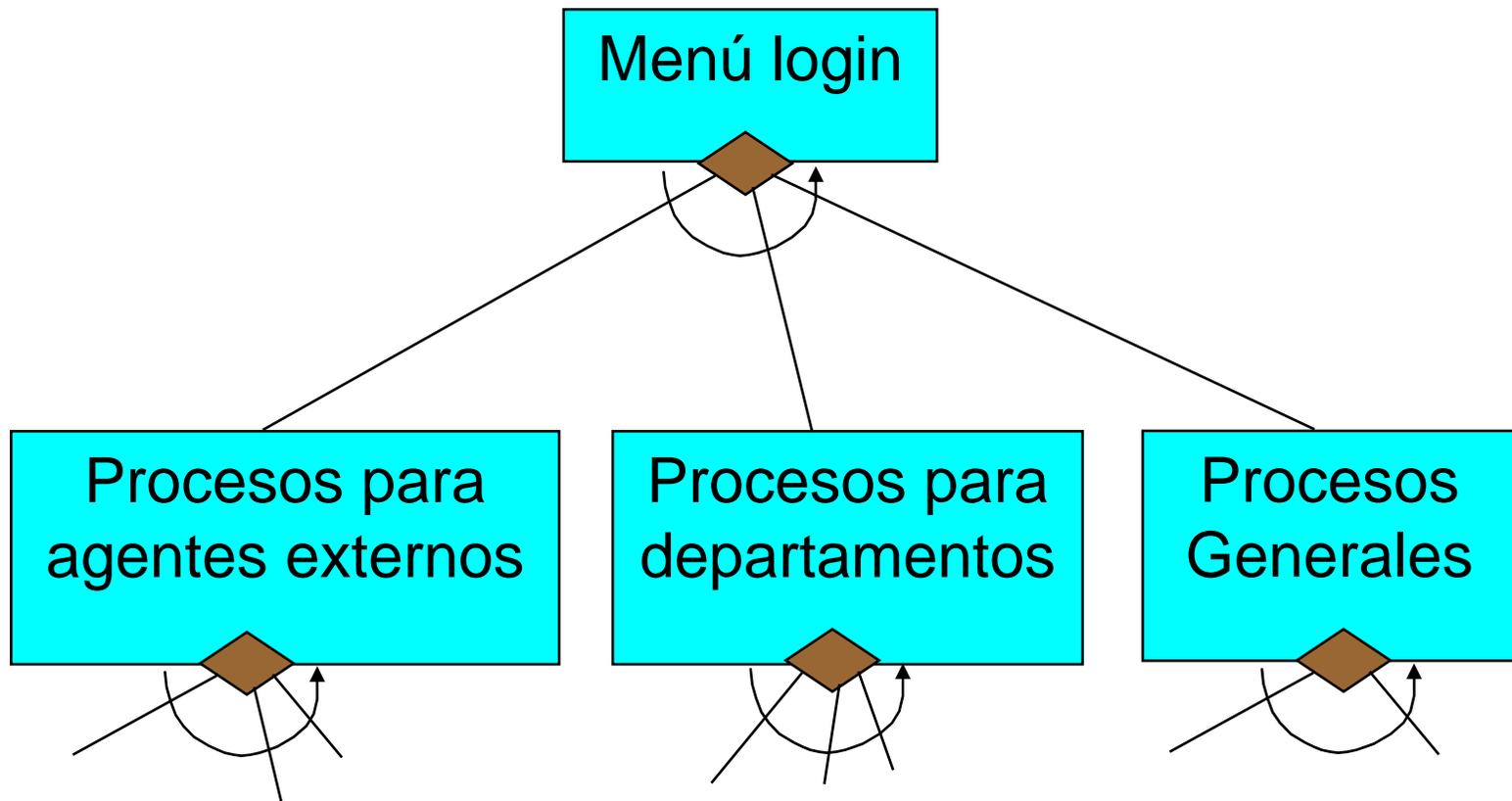


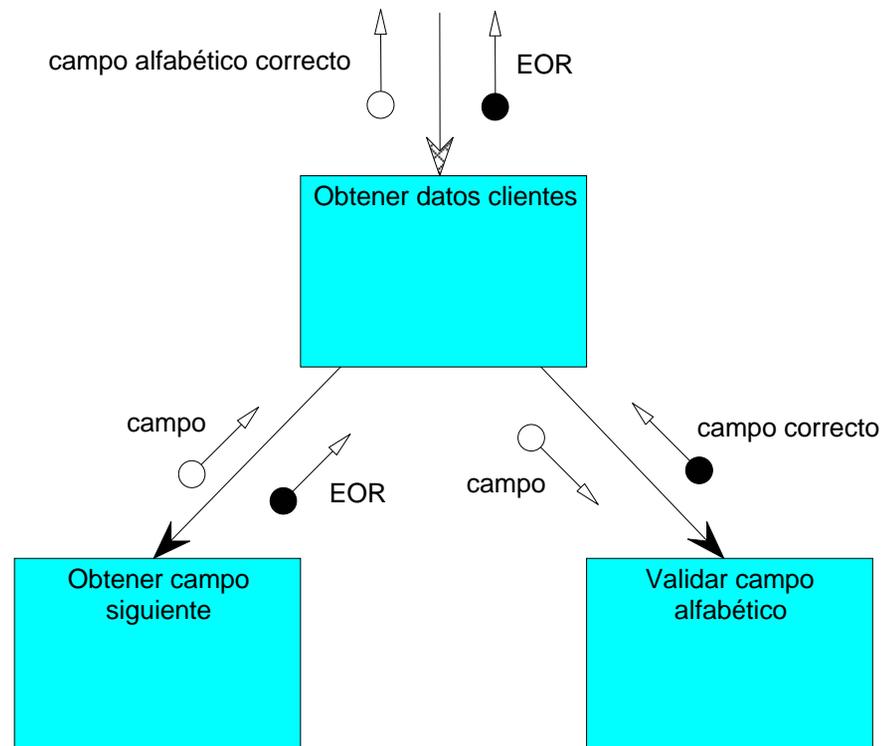
Diagrama de estructura. Comunicación entre módulos

- Los signos para llevar a cabo la comunicación entre módulos son:

Flags o controles



Datos



Flags o controles



- Mediante los “flags” o controles, se puede representar:
 - Paso de control entre módulos: un módulo comunica a otro módulo que ha terminado su proceso y traspasa al módulo llamado el control del sistema.
 - Comunicación de que se ha producido un error en el proceso.
 - Comunicación de que se puede proceder a una operación concreta.

Diferencias entre datos y flags



- Los datos se procesan.
- Los datos son la información compartida por los módulos.
- La posición de la flecha (hacia arriba o hacia abajo) indica el sentido de la comunicación.
- Los datos tienen importancia para el mundo exterior, están relacionados con el problema.
- Los controles sólo sirven para comunicar condiciones entre los módulos.
- Los controles indican al módulo que llama la terminación EOF, o un error del módulo llamado, y deben ir siempre en **sentido ascendente**.
- En sentido descendente dan lugar a un acoplamiento de control no deseable y la cohesión se ve comprometida.
- Se pretende que los módulos de arriba coordinen, los de abajo realicen el trabajo específico y señalen condiciones anormales o de terminación.
- Los flags tienen importancia en la comunicación de información en el interior; son los que sincronizan la operativa de los módulos.
- Los flags no se suelen mostrar en los diagramas (Piattini et al. 04)

Representación de parámetros

- Se pueden representar mediante *tablas de interfaz* (Piattini et al. 04)

Módulo	Parámetro formal	Entrada	Salida	Uso	Significado
F(x,y)	x	Sí	No	P	Fecha nacimiento
	y	No	Sí	M	Edad

...donde "Uso" es denotado por:

P → Procesado: $a = \mathbf{b} + 2$

M → Modificado: $\mathbf{a} = 3 + b$

T → Transferido por el módulo llamado a otro módulo que éste llama, sin modificar su valor

C → Es usado como una variable de Control

I → El parámetro es transferido a otro módulo y es modificado en este segundo módulo

Diagrama de estructura.

Ejemplo (Piattini et al. 96)

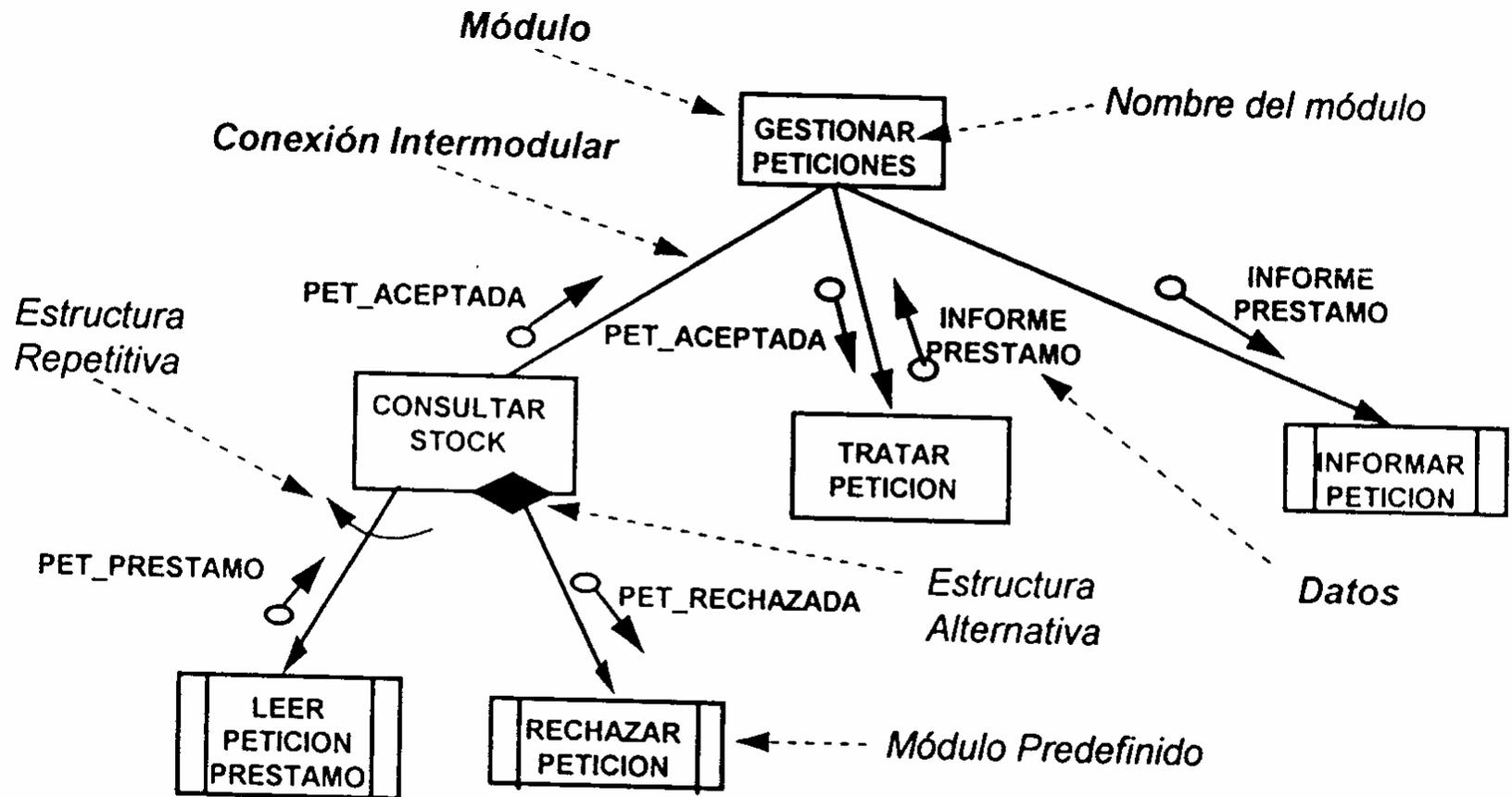
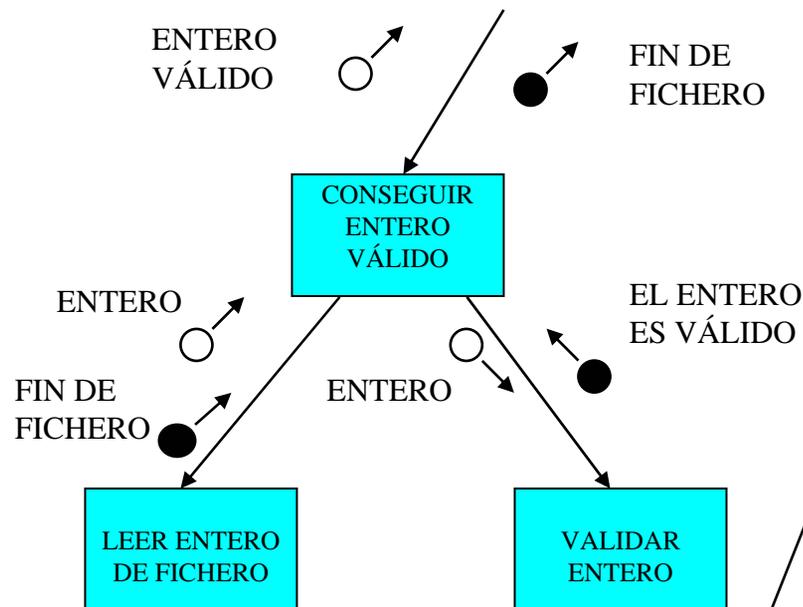


Diagrama de estructura.

Ejemplo (II) (Molina et al. 97)



“CONSEGUIR ENTERO VÁLIDO”:

...

`LEER_ENTERO(fin_fichero, entero) ;`

...

`if VALIDAR_ENTERO(entero) then`

...

...

Diagrama de estructura.

Ejemplo (III) (Molina et al. 97)

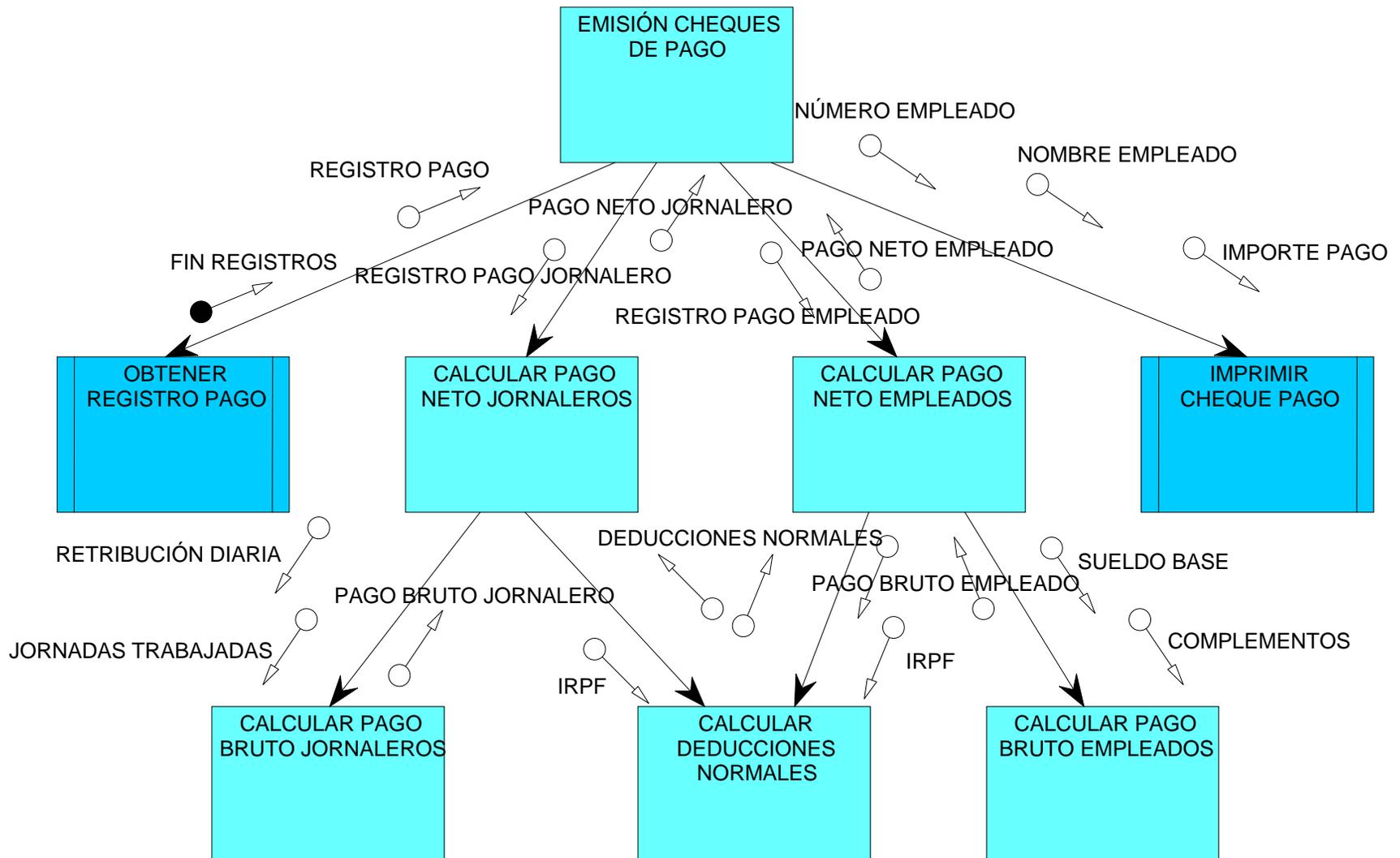


Diagrama de estructura.

Ejemplo (III) (Molina et al. 97)

```
program EMISION_CHEQUES ;
type
    treg_pago : RECORD...END ;
    treg_jornalero : RECORD...END ;
    treg_empleado : RECORD...END ;
var
    importe : real ;
    importe_pago_jorn, importe_pago_empl : real ;
    registro_pago : treg_pago ;
    registro_empleado : treg_empleado ;
    registro_jornalero : treg_jornalero ;
    fin_registros : boolean ;
    numero_empleado : integer ;
    nombre_empleado : string ;
begin
    OBTENER_REGISTRO_PAGO (registro_pago, fin_registros) ;
    ...
    importe_pago_jorn := CALCULAR_NETO_JORN (registro_jornalero) ;
    ...
    importe_pago_empl := CALCULAR_NETO_EMPL (registro_empleado) ;
    ...
    IMPRIMIR_CHEQUE_PAGO( numero_empleado, nombre_empleado, importe) ;
    ...
end.
```

```
procedure OBTENER_REG_PAGO ( var rp : treg_pago; var fin_reg : boolean ) ;
function CALCULAR_NETO_JORN ( rj : treg_jornalero ) : real ;
function CALCULAR_NETO_EMPL ( re : treg_empleado ) : real ;
function CALCULAR_BRUTO_JORN ( ret_diaria, jorn_trab : real ) : real ;
function CALCULAR_BRUTO_EMPL ( sueldo_base, complem : real ) : real ;
function CALCULAR_DEDUCCIONES ( pago_bruto, irpf : real ) : real ;
procedure IMPRIMIR_CHEQUE_PAGO( num_emp : integer ; nom_emp : string;
importe : real ) ;
```

Métodos para la especificación de módulos



- Interfaz-función (*módulo, entradas, salidas, función*).
- Pseudo-código.
 - Más preciso que el usado en análisis
 - Deja cierto grado de libertad al programador
 - No trata aspectos de eficiencia, a menos que estén directamente relacionados con requisitos
 - Permite verificar la calidad del diseño
- Herramientas complementarias:
 - Diagramas de flujo
 - Nassi-Schneiderman
 - Tablas y árboles de decisión

Estrategias de diseño



- Pasos generales a seguir para obtener un buen diseño a partir de un DFD de procesos primitivos
- A veces hay que refinar el DFD de partida
- Dos estrategias:
 - Análisis de transformaciones
 - Análisis de transacciones
- **Importante:** diseñar el DE de forma que:
 - Los módulos de nivel superior toman las decisiones de ejecución (coordinan)
 - Los de nivel inferior realizan la mayor parte del trabajo de entrada, de cálculo y de salida

Estrategias de diseño.

Pasos a seguir

- Revisar el modelo fundamental del sistema
 - ⇒ *DFD procesos primitivos*
 - ⇒ no hace falta "crear" el DFD de procesos primitivos
 - ⇒ se añaden procesos, si hace falta
 - ⇒ recomendado, como mínimo, tener 3 niveles de profundidad
- Determinar si el DFD tiene características de transformación o de transacción
 - *indica expresamente la característica del DFD!*

Estrategias de diseño.

Pasos a seguir (II)

- Según sea de **transformación** o **transacción**:
 - a) Aislar el centro de la transformación, especificando los límites del flujo de llegada y de salida
...o bien...
 - b) Identificar el centro de la transacción y las características del flujo de cada camino de acción.
- ⇒ *Consejo: indica expresamente los elementos anteriores!*

Estrategias de diseño.

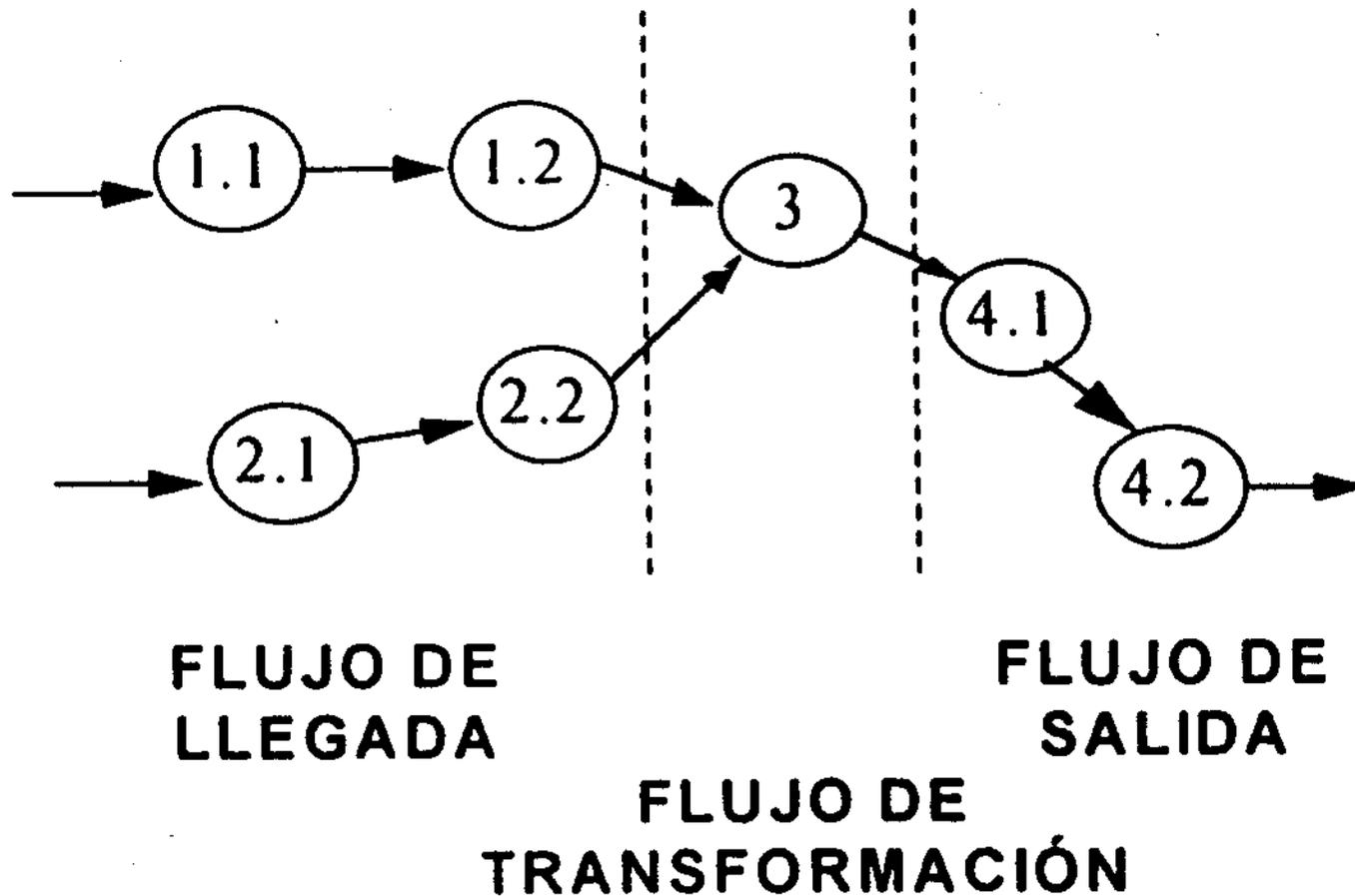
Pasos a seguir (III)



- Realizar el primer corte del diagrama de estructuras.
- Realizar el segundo nivel de factorización.
- Refinar la estructura del programa.
- Asegurarse del trabajo realizado por el diseño obtenido.

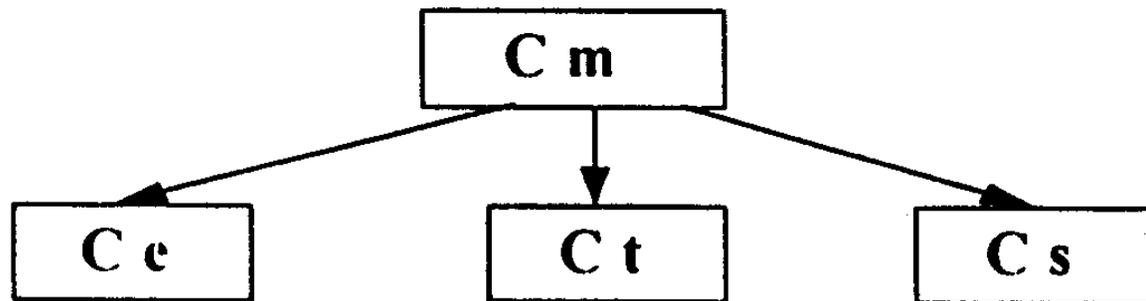
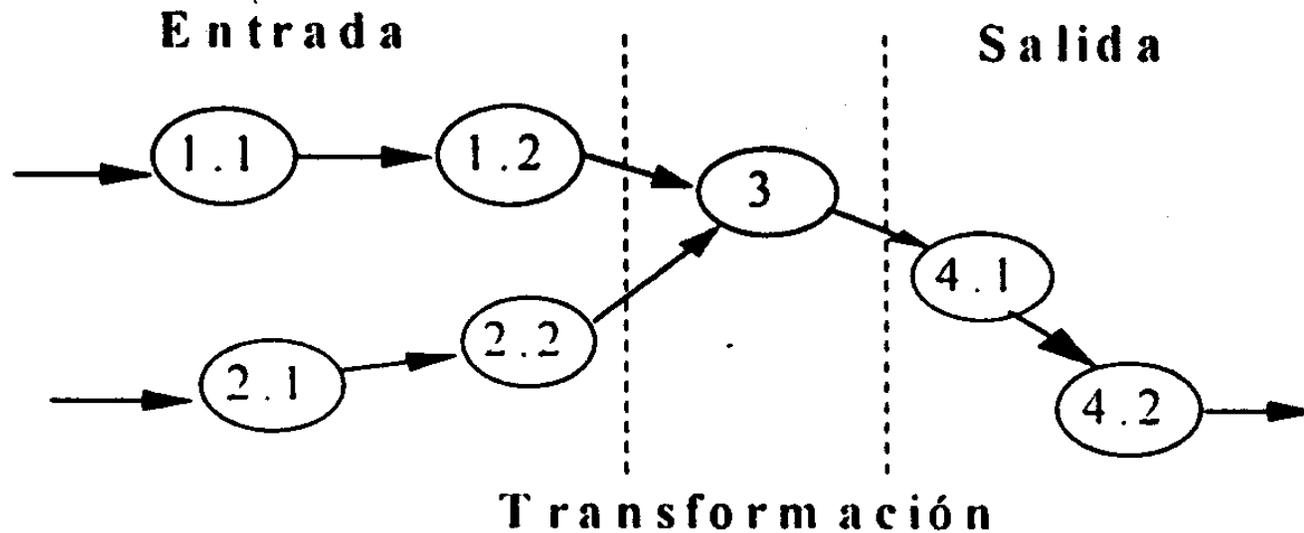
Análisis de transformación

(Piattini et al. 04)



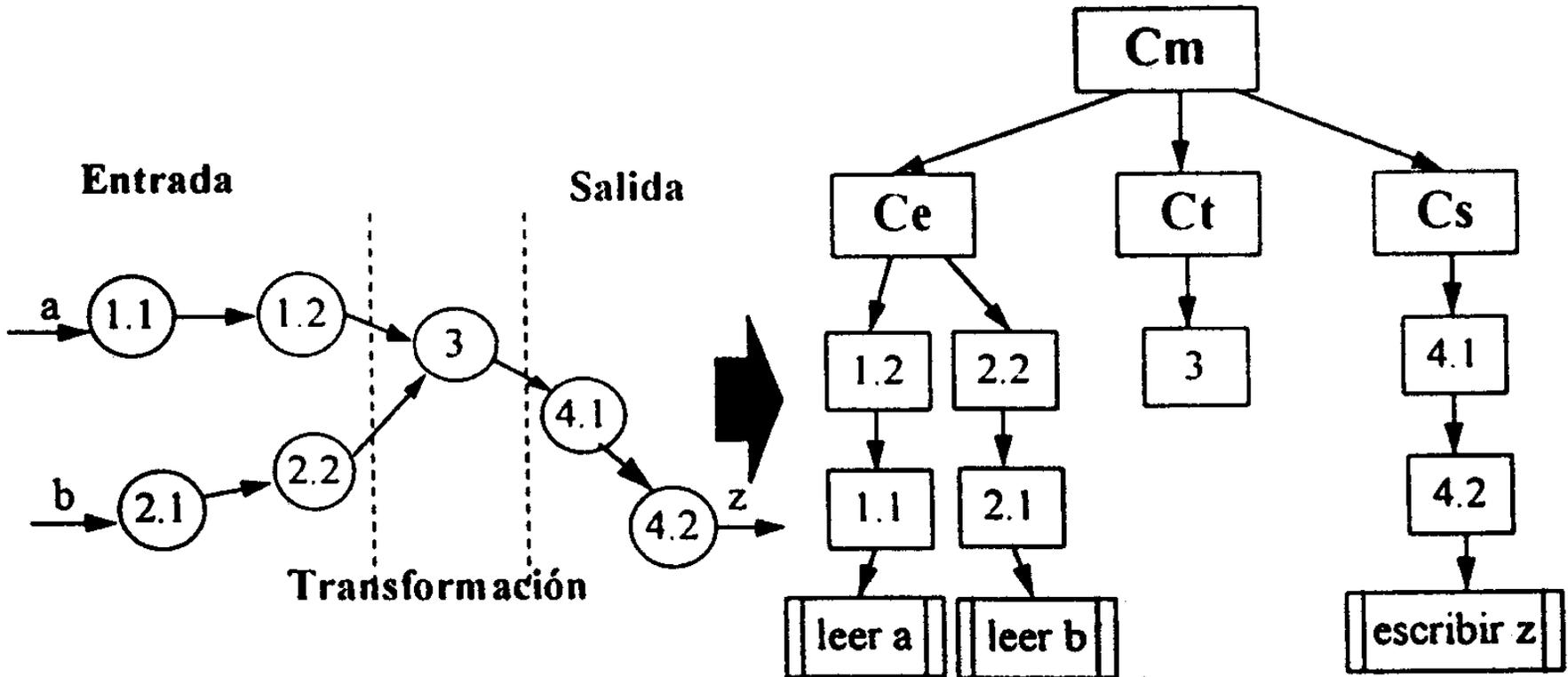
Análisis de transformación.

Primer corte del DE (Piattini et al. 04)



Análisis de transformación.

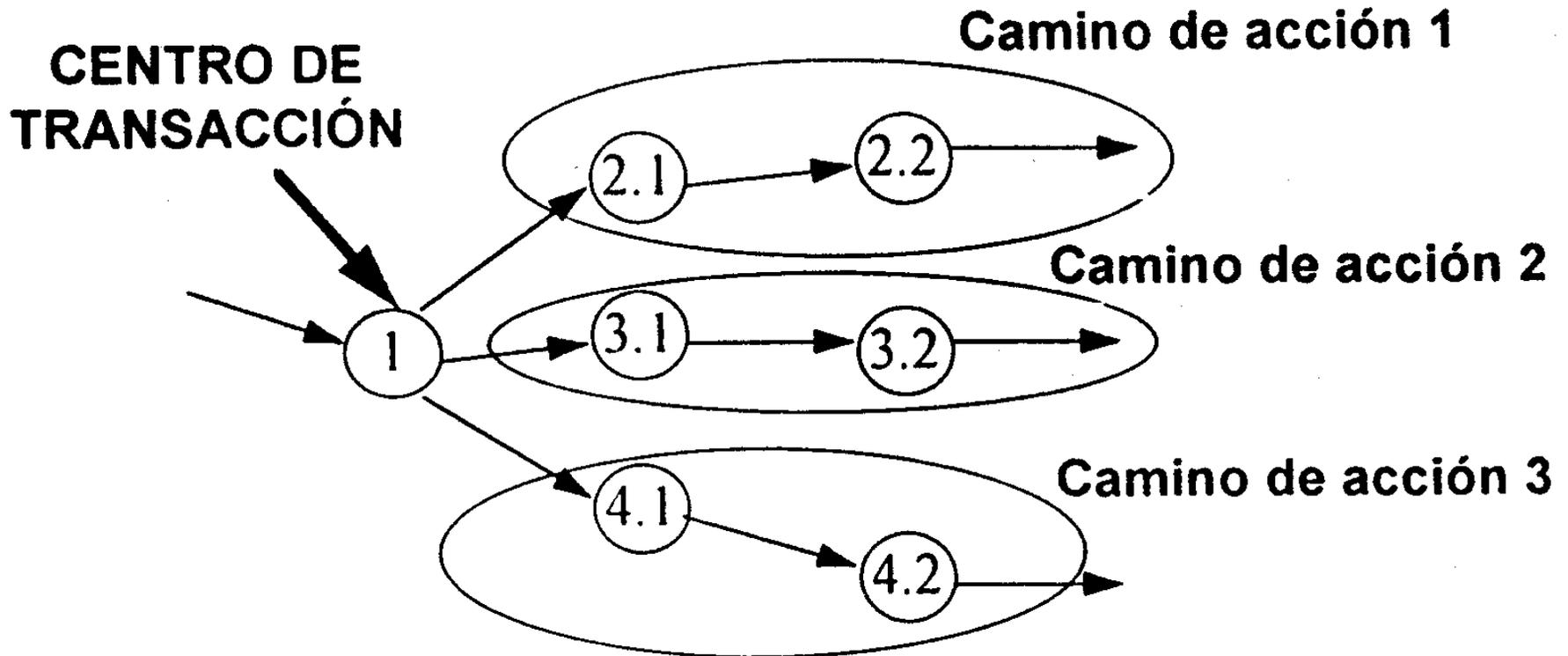
Segundo nivel de factorización



Finalmente, es preciso optimizar!

Análisis de transacción

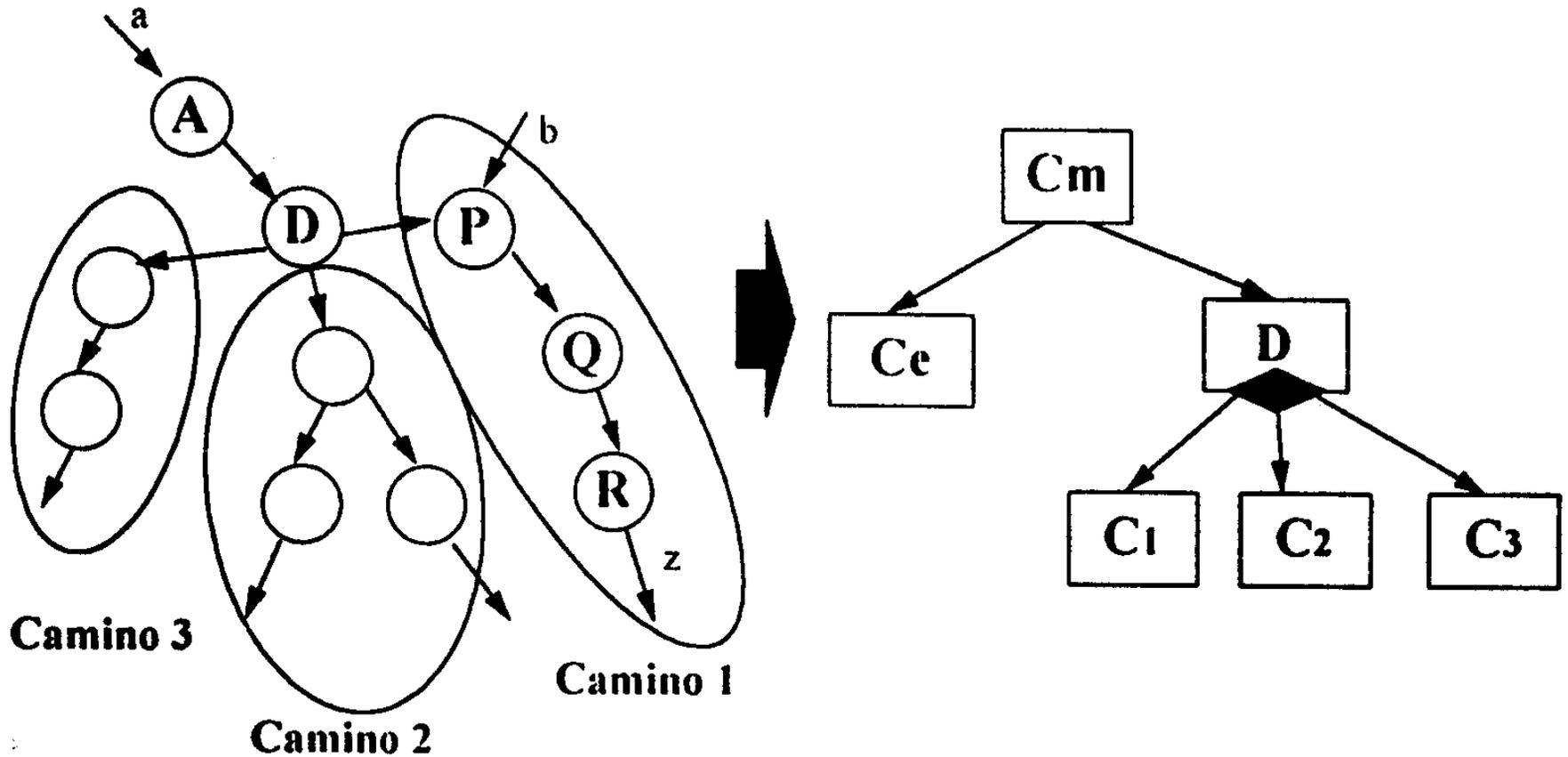
(Piattini et al. 04)



⇒ Son transacciones (caminos) independientes.

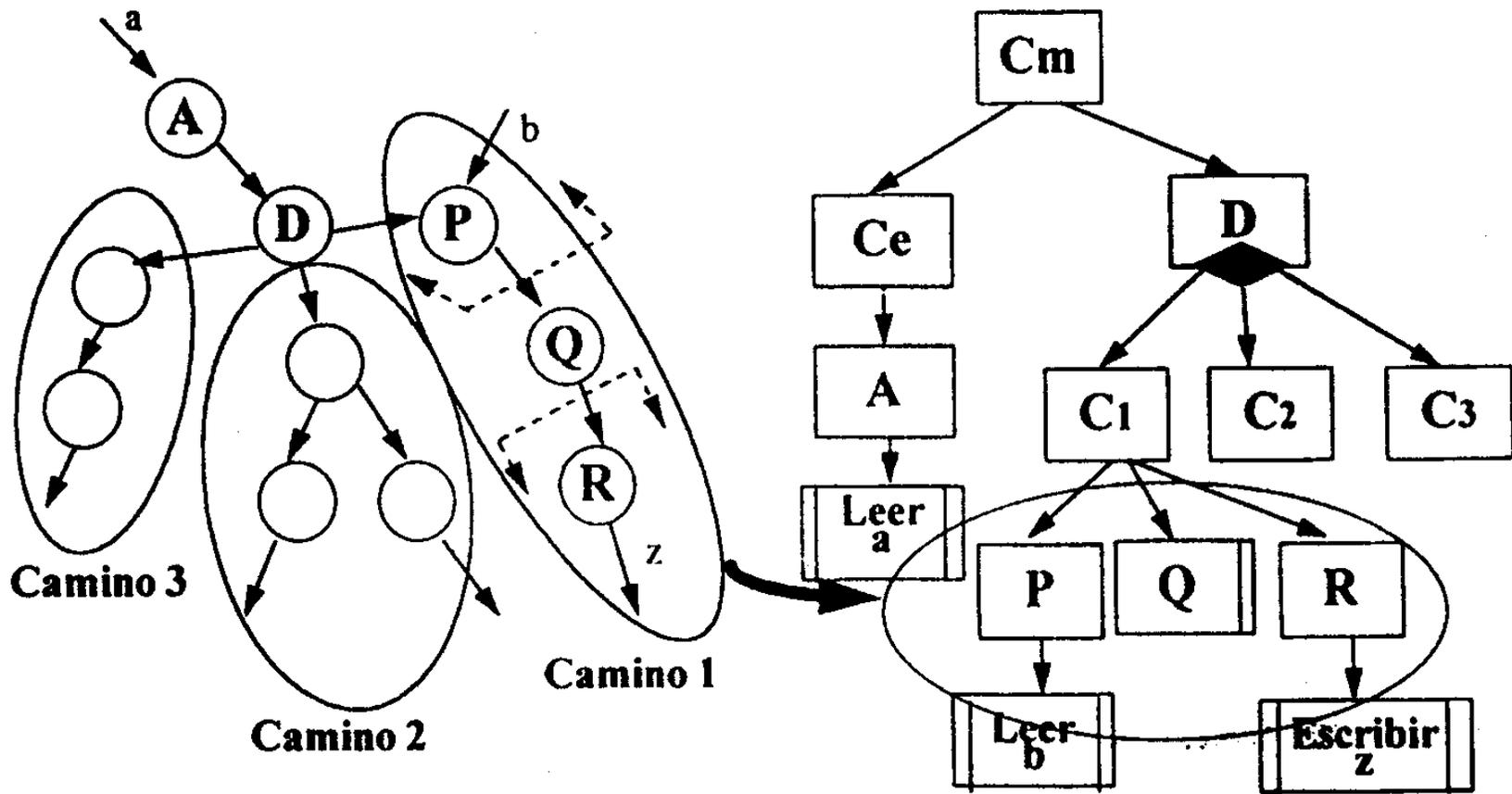
Análisis de transacción.

Primer nivel de factorización



Análisis de transacción.

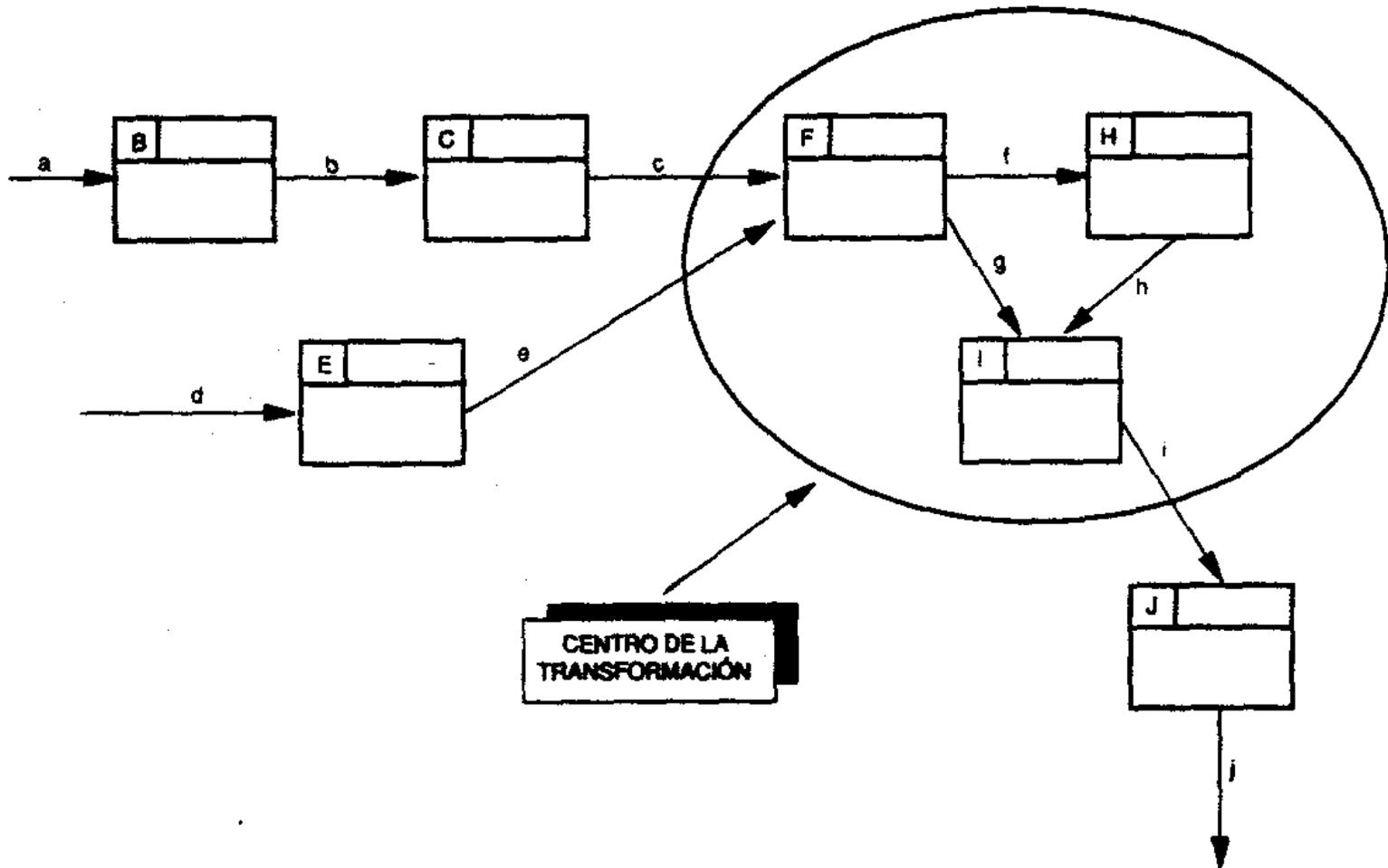
Segundo nivel de factorización



Finalmente, es preciso optimizar!

Análisis de transformación.

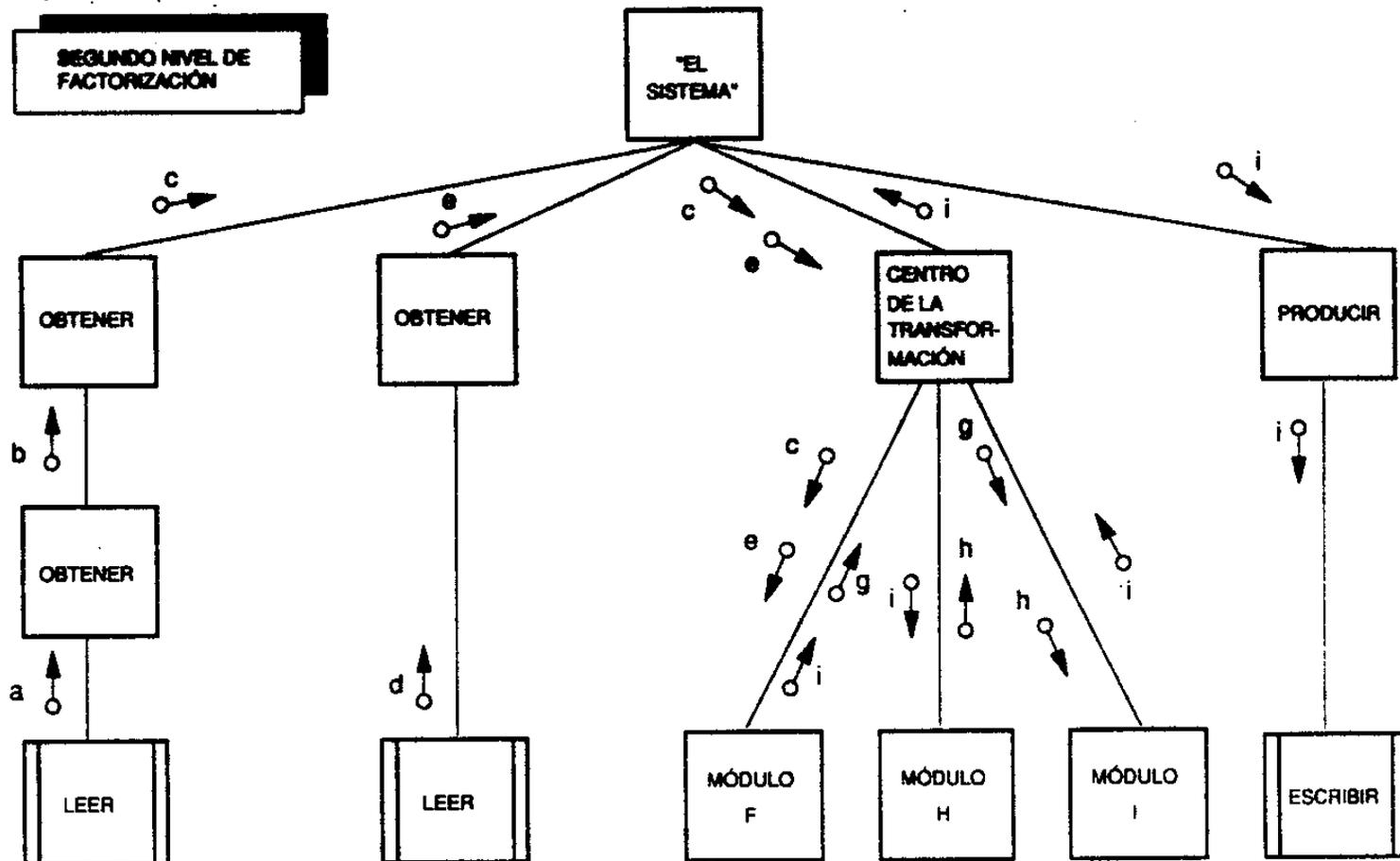
Ejemplo (Guía de técnicas de Métrica 2.1, MAP 95, p.144)



Análisis de transformación.

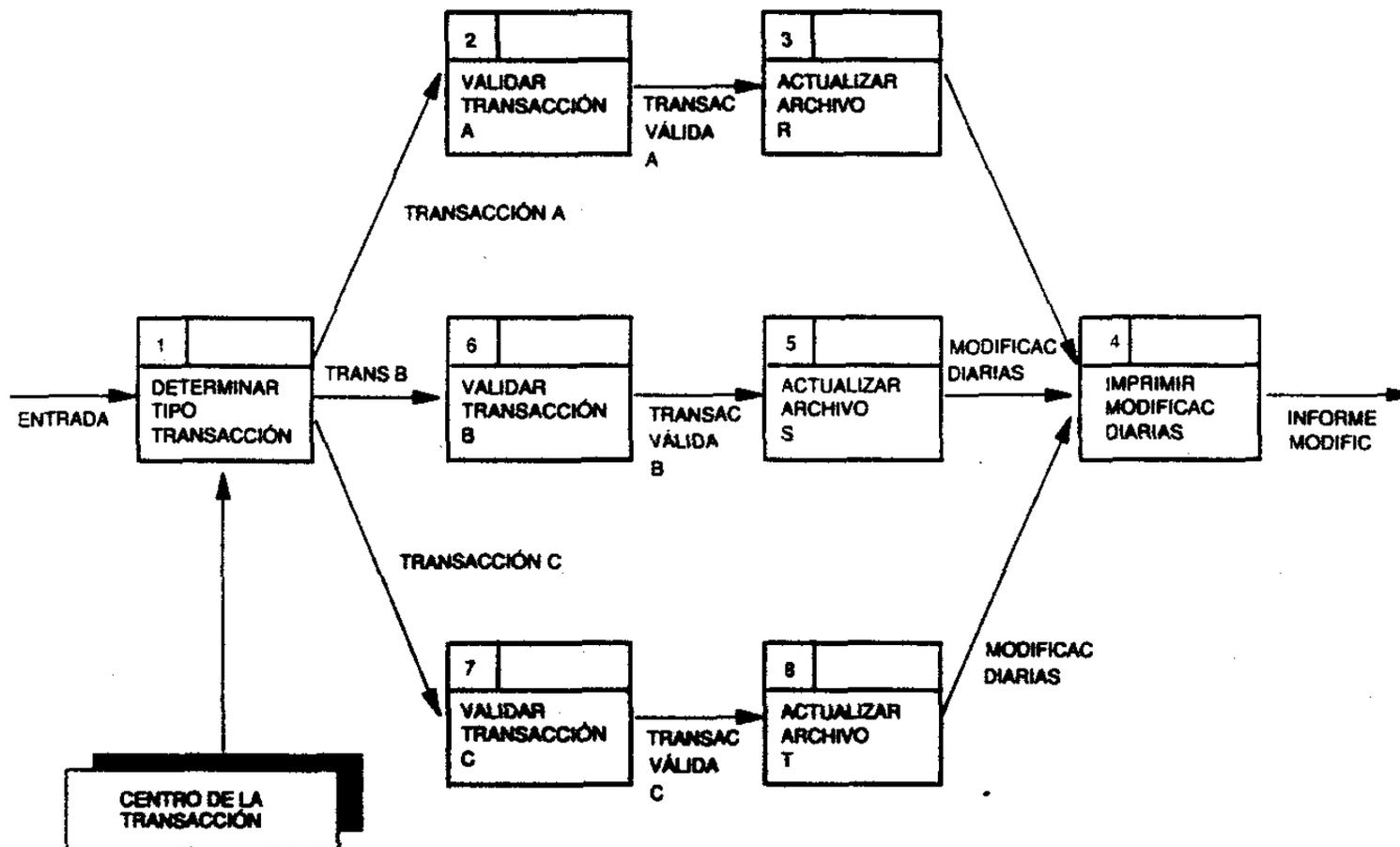
Ejemplo (II) (MAP 95)

Un planteamiento para discutir:



Análisis de transacciones.

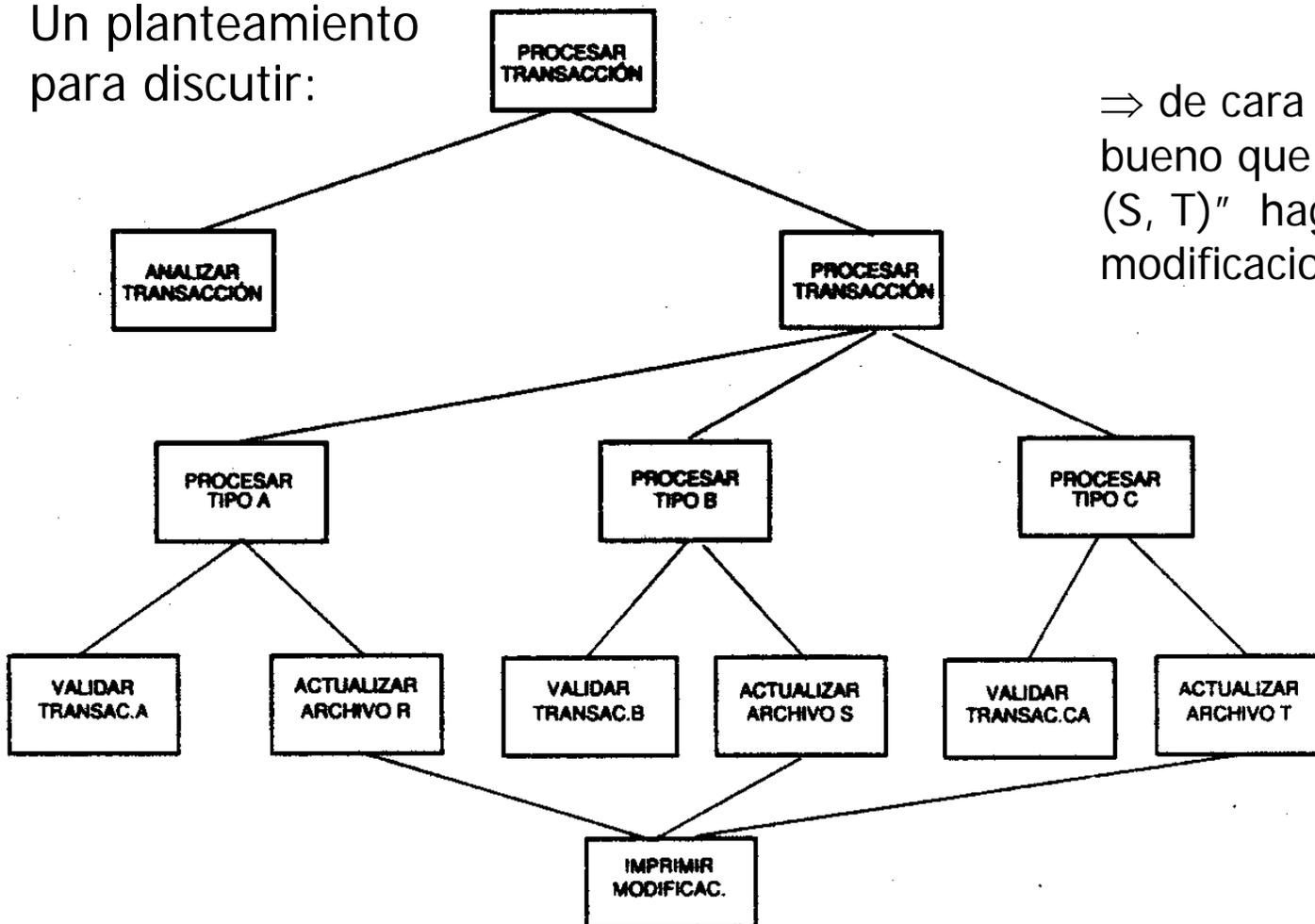
Ejemplo (Guía de técnicas de Métrica 2.1, p.148)



Análisis de transacciones.

Ejemplo (II) (MAP 95)

Un planteamiento para discutir:



⇒ de cara a la reutilización, ¿es bueno que “Actualizar archivo R (S, T)” haga uso de “imprimir modificaciones”?

Centros de transacción implícitos

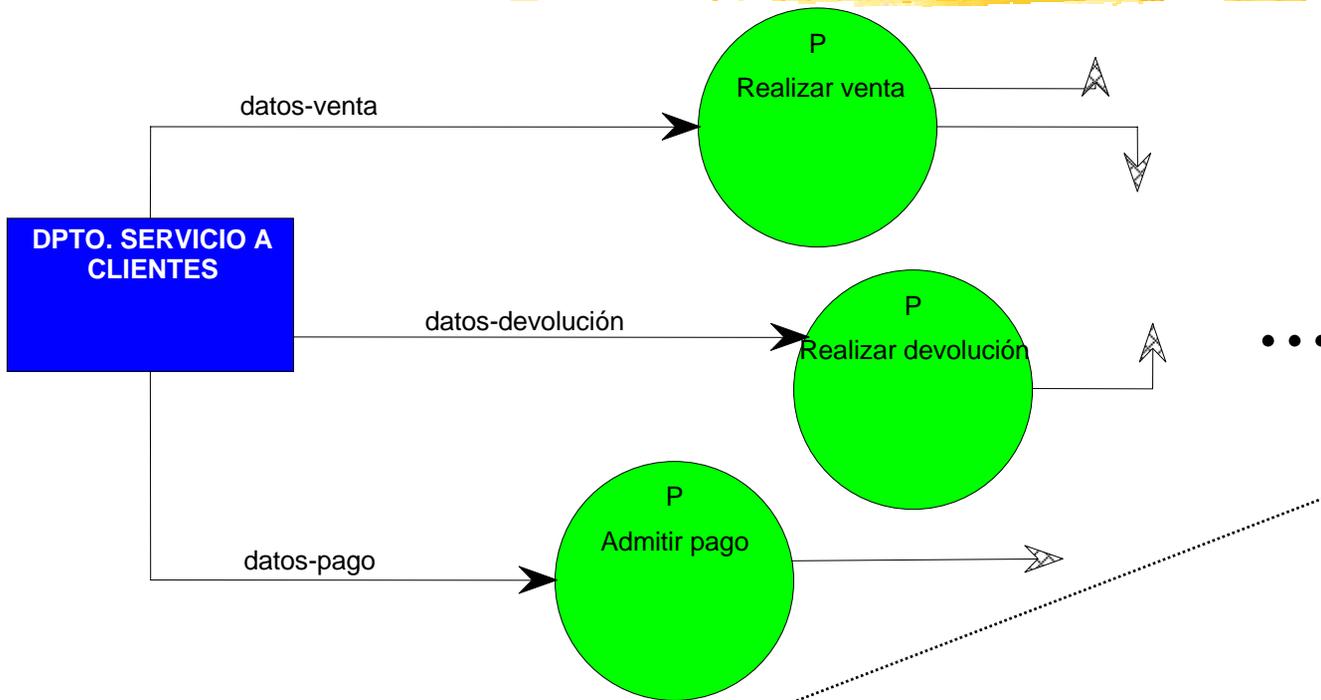


Normalmente el esquema de transacción no es tan claro:

⇒ el proceso de transacción no aparece explícitamente en el DFD

⇒ **Solución**: examinar el diagrama de contexto y la lista de eventos para determinar los tipos de transacciones en el sistema

Centros de transacción implícitos (II)



(Molina et al. 97)
p.172

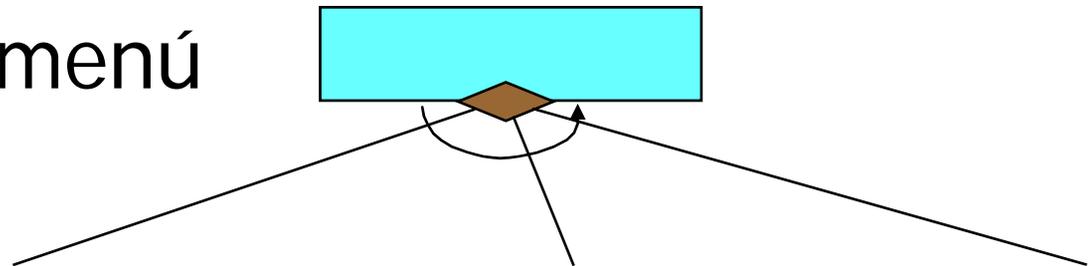
Seleccione la opción deseada:

1. Realizar venta
2. Realizar devolución
3. Admitir pago

Estrategia de diseño.

Pasos a seguir (IV) (Molina et al. 97) p.169

- Análisis de transacciones
 - Encontrar las transacciones en el DFD
- Análisis de transformaciones
 - DE para cada transacción
- Análisis de transacciones
 - Componer los DE en uno solo, usando un centro de transacciones
- DE para un menú



4. Métricas de calidad estructural



- Mayor calidad estructural \Rightarrow
mantenimiento más fácil
 - Extensibilidad ++
 - Reutilización ++
- Dos métricas:
 - Acoplamiento
 - Cohesión

Acoplamiento

- Grado de interdependencia entre módulos.
- Depende de la forma de interactuar entre los módulos: p.ej. n° / tipo de parámetros que se intercambian.
- Objetivo: minimizar el acoplamiento (CAJA NEGRA).
 - ↳ modificabilidad++, comprensión++, reutilización++
- Ventajas de un bajo acoplamiento:
 - Menos oportunidades para el “efecto onda”.
 - El cambio realizado en un módulo afecta lo menos posible a otros módulos.
 - Mientras se esté manteniendo un módulo, es deseable no necesitar preocuparse de los detalles internos de cualquier otro módulo.

Complejidad de la interfaz

■ Acoplamiento → Complejidad de la interfaz

P.ej. (Molina et al. 97)

- 1. CALL LONGITUD(X1, Y1, X2, Y2, D)
- 2. CALL LONGITUD(ORIGEN, DESTINO, D)
- 3. CALL LONGITUD(PuntosX, PuntosY, D)
- 4. CALL LONGITUD(LINEA, D)
- 5. CALL LONGITUD(TABLALINEA)
- 6. CALL LONGITUD.

⇒ *¿Qué interfaz es más fácil de entender?*

Complejidad de la interfaz

(II)



■ Depende de:

- Cantidad de información que debe comprenderse (como n° parámetros)
- Accesibilidad a esa información
- Indirección \Rightarrow complejidad++
- Información local \Rightarrow complejidad--
- Información en forma estándar \Rightarrow complejidad--
- Si el parámetro intenta controlar la lógica del módulo que lo recibe \Rightarrow complejidad++

Niveles de acoplamiento

Stevens, Myers y Constantine 74

■ Acoplamiento Normal

- De datos
- Por estampado
- De control

MEJOR (- Acopl.)

■ Acoplamiento común

■ Acoplamiento por contenido

Límite de aceptación



PEOR (+ Acopl.)

Si dos módulos presentan varios tipos de acoplamiento, se considera que tienen el peor de los acoplamientos que presentan.

Niveles de acoplamiento (II)



■ Acoplamiento normal:

A y B normalmente acoplados si:

- 1) A llama a B;
- 2) B retorna el control a A

■ *De datos:*

- se establece una comunicación básica por medio de elementos de datos

■ *De estampado:*

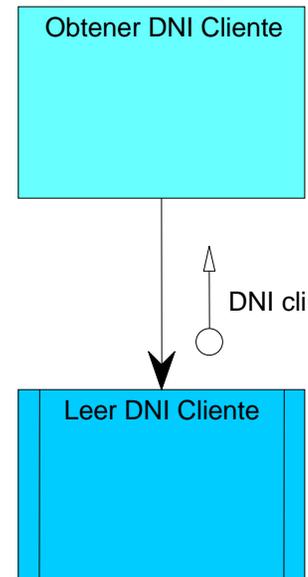
- se pasan datos con estructura de registro

■ *De control:*

- se comunican con flags de control

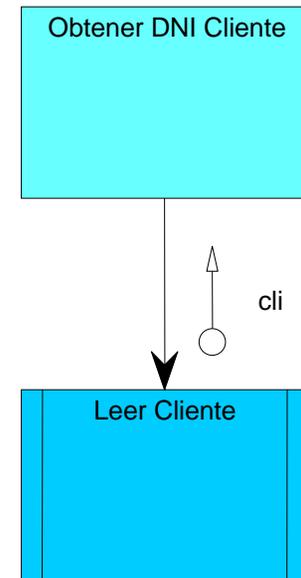
Acoplamiento normal de datos

- No genera problemas.
- Es el acoplamiento deseable en diseño estructurado
- Sólo debe haber parámetros con sentido
- En la medida de lo posible, minimizar el número de parámetros



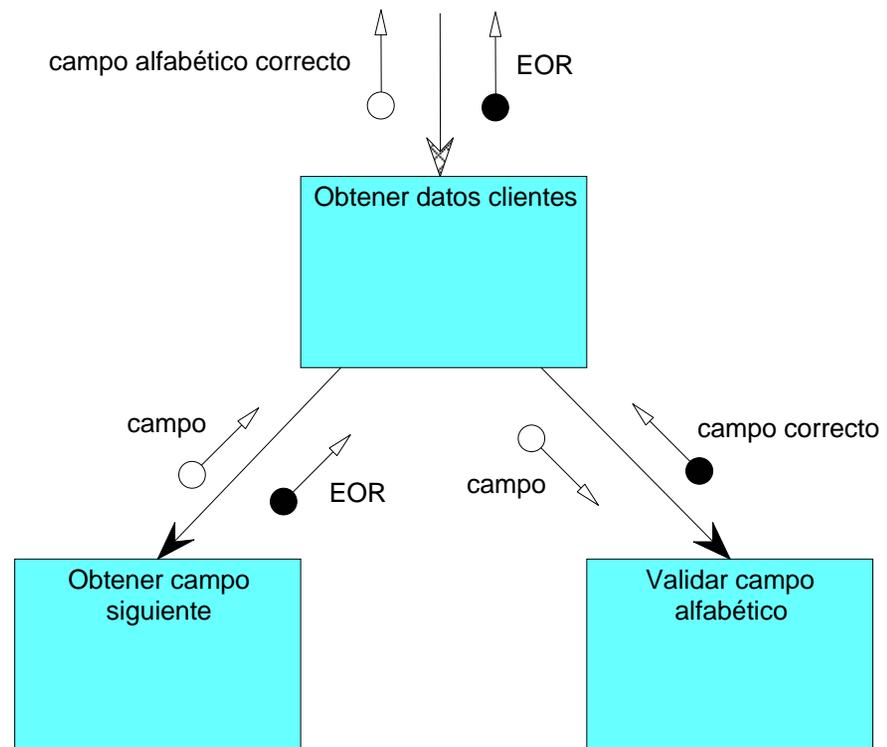
Acoplamiento normal por estampado

- Introduce indirección
- No es deseable si el módulo que recibe el registro sólo necesita parte de los datos.
- Pasar campos innecesarios oscurece el diseño y reduce la flexibilidad
- No crear registros artificiales, empaquetando datos no relacionados



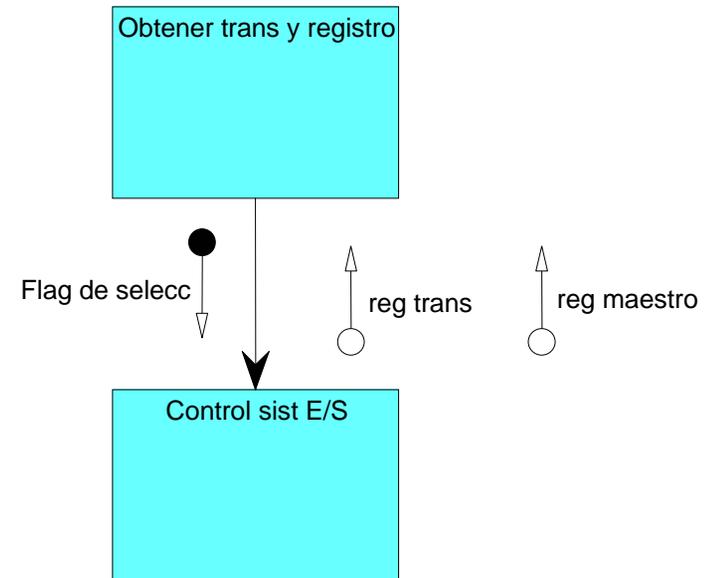
Acoplamiento normal de control

- Deseable sólo si los flags de control son descriptivos



Acoplamiento normal de control (II)

- No es deseable si el control tiene sentido descendente:
 - Un módulo controla a otro y no son realmente independientes
 - El módulo subordinado tiene poca cohesión
 - Solución: sustituir el módulo subordinado por tantos módulos como sea necesario, de forma que sólo intercambien datos



Flag de selecc:

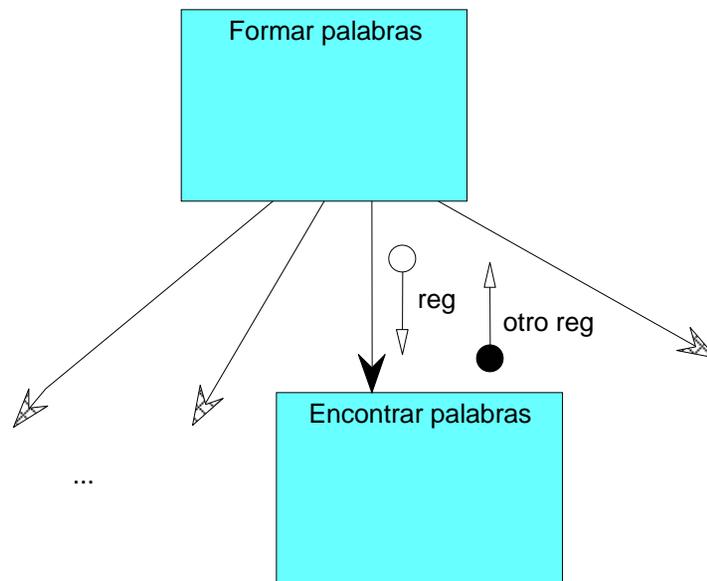
1. Obtiene RT
2. Obtiene RM
3. Obtiene RT y RM

Acoplamiento normal de control (III)

■ Peor es si hay "inversión de autoridad"

(Molina et al. 97) p.66

⇒ el módulo subordinado intenta controlar al módulo padre



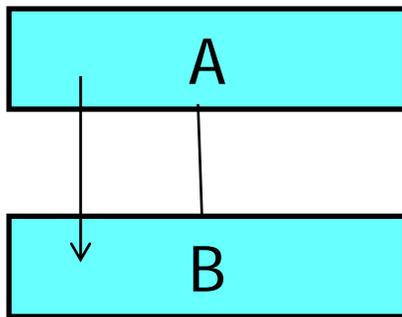
Niveles de acoplamiento (III)



- **Acoplamiento común.** Más de dos módulos hacen referencia a un área común de datos.
 - Dos módulos pueden estar acoplados globalmente y no estar conectados mediante una llamada.
 - En general, es desaconsejable, dado que (Molina et al. 97):
 - Un error de programación en un módulo que usa puede aparecer en otros módulos que compartan esa misma área global.
 - Reutilización--
 - Puede ser difícil averiguar de dónde procede información depositada en el área global.
 - Se pueden incluso usar para depositar información de distinta naturaleza.
 - Aplicaciones difíciles de mantener: es difícil saber qué datos son usados por un módulo.

Niveles de acoplamiento (IV)

- **Acoplamiento de contenido.** Ocurre cuando un módulo necesita o accede a una parte de otro, rompiendo la jerarquía de funcionalidad de la estructura.
- En la mayoría de los casos sólo se puede dar en ensamblador



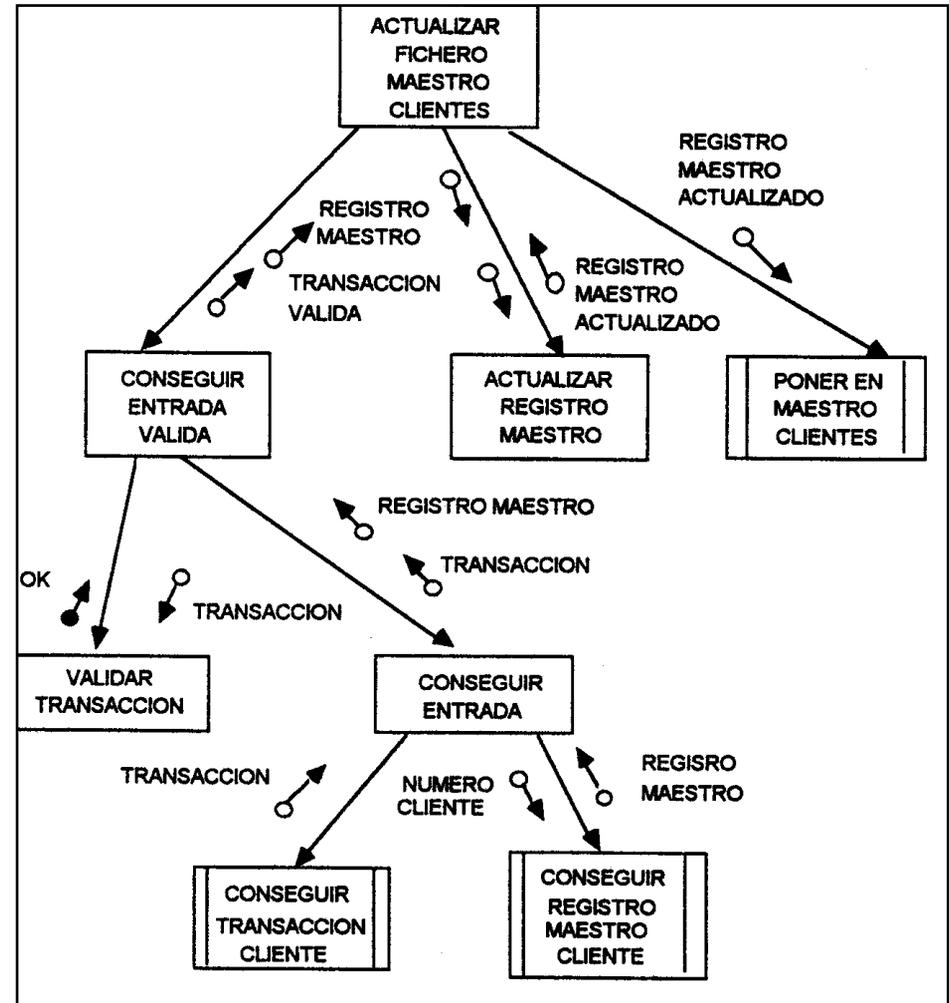
Hay que evitarlo descomponiendo el módulo al que se accede o duplicando esa parte de código en el módulo que llama

Datos vagabundos *(Molina et al. 97) p.62*

Deben evitarse también los *datos vagabundos* (que a veces están asociados al acoplamiento normal)

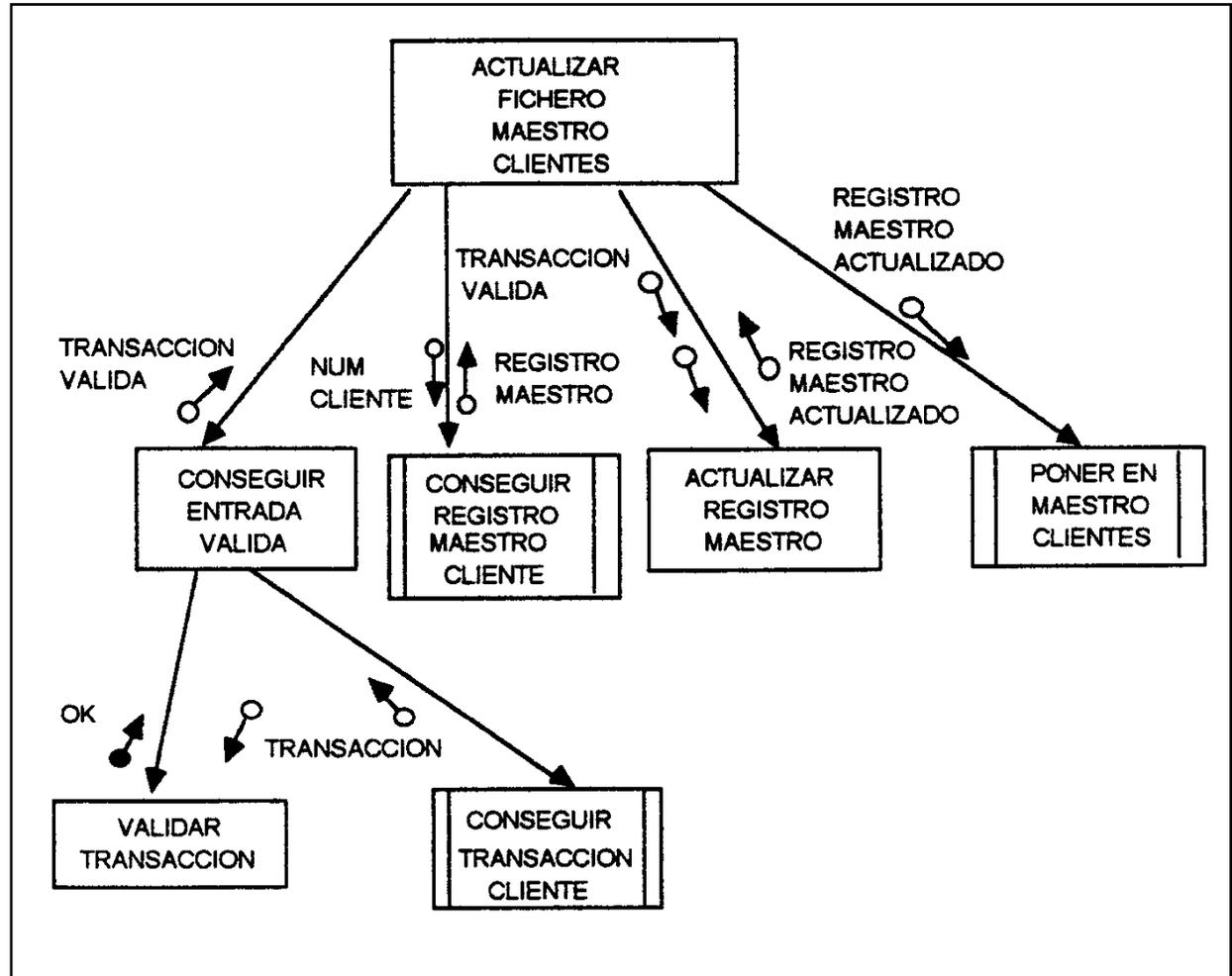
Posibles soluciones:

- Reorganizar el diagrama
- Introducir variables globales



Datos vagabundos (II)

Solución (a) El dato vagabundo "registro maestro" se ha eliminado reorganizando el DE



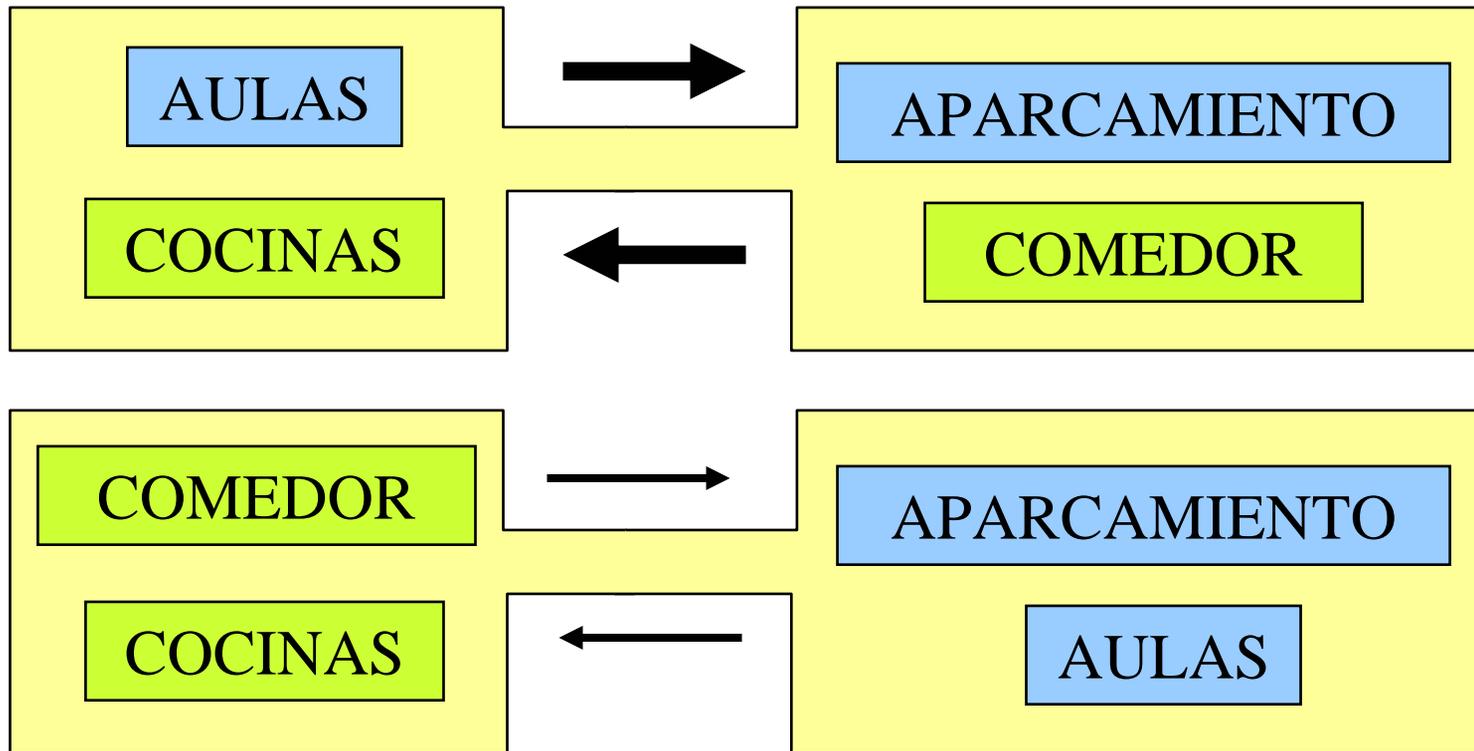
Cohesión



- Medida de la relación funcional entre los elementos de un módulo.
- Un módulo coherente sólo debe hacer una cosa.
- Un módulo coherente ejecuta una tarea bien definida en un programa y requiere poca interacción con otros procedimientos que se ejecutan en otras partes del programa.
- Objetivo: módulos con alta cohesión.
- La escala de cohesión no es lineal:
 - una cohesión baja es mucho peor que una de grado medio,
 - una de grado medio es casi tan buena como una alta.

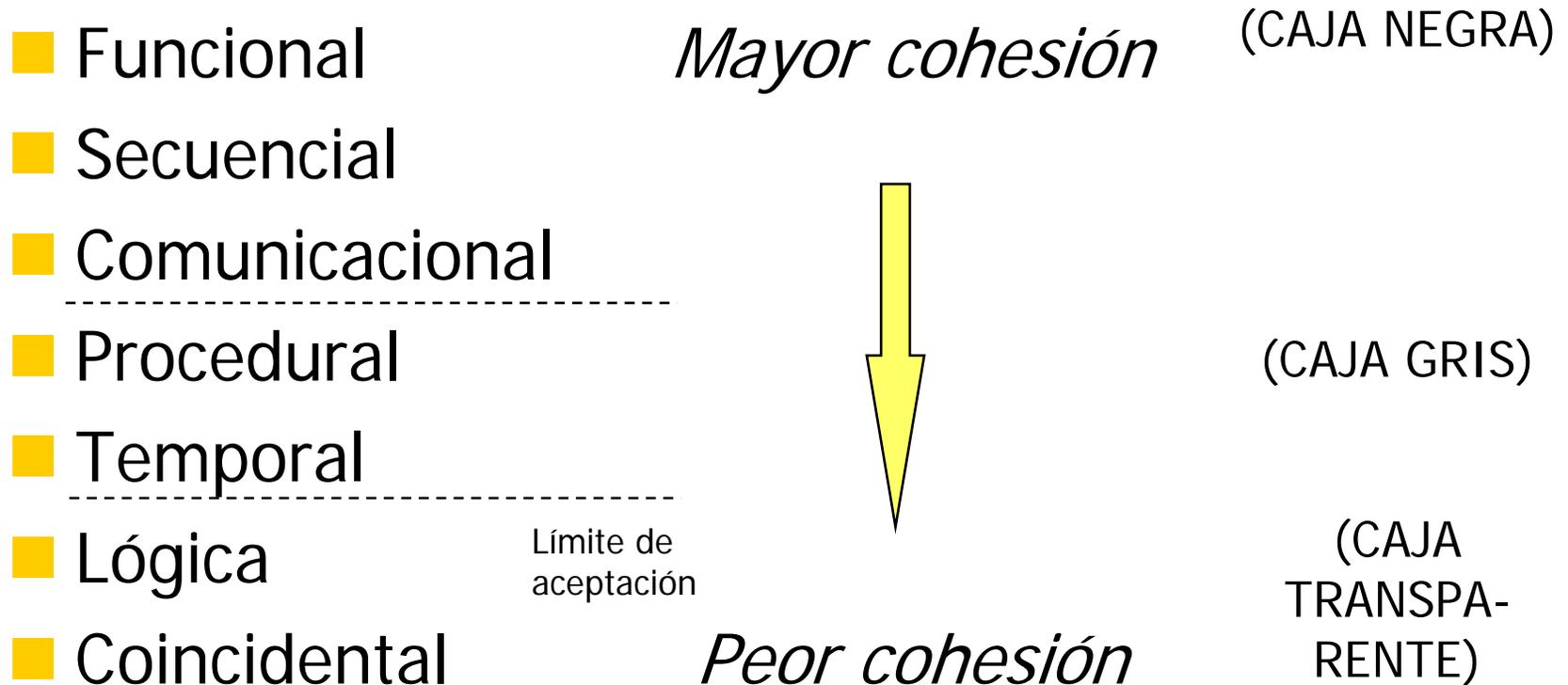
Cohesión y acoplamiento

Son criterios inversamente relacionados: (Molina et al. 97)



Niveles de cohesión

Stevens, Myers y Constantine 74



Niveles de cohesión (II)



- **Funcional**: se realiza una sola función.
- **Secuencial**: la salida de una tarea sirve como entrada a la siguiente: varios módulos con cohesión funcional que se pasan un dato
- **Comunicacional**: actividades que comparten datos (bien los mismos datos de entrada o de salida).
- **Procedural**: actividades diferentes, en las cuales el flujo de ejecución fluye de una a la siguiente.
- **Temporal**: actividades diferentes relacionadas por el tiempo.
- **Lógica**: actividades con la misma categoría lógica general, que se seleccionan fuera del módulo.
- **Coincidental** o **casual**: actividades diferentes sin relaciones significativas entre ellas.

Niveles de cohesión (III)

Ejemplos:

Funcional:

funciones matemáticas como $\cos(\alpha)$;
escribir registro (fichero, registro)

Secuencial:

Formatear registro = Leer registro + Aplicar formato registro

Comunicacional:

Obtener detalles cliente (num_cta, nombre_cli, saldo_prestamo)

Procedural:

Módulo "Detectar error, corregirlo y reanudar la ejecución"
⇒ ¿Por qué no es cohesión secuencial?

Temporal:

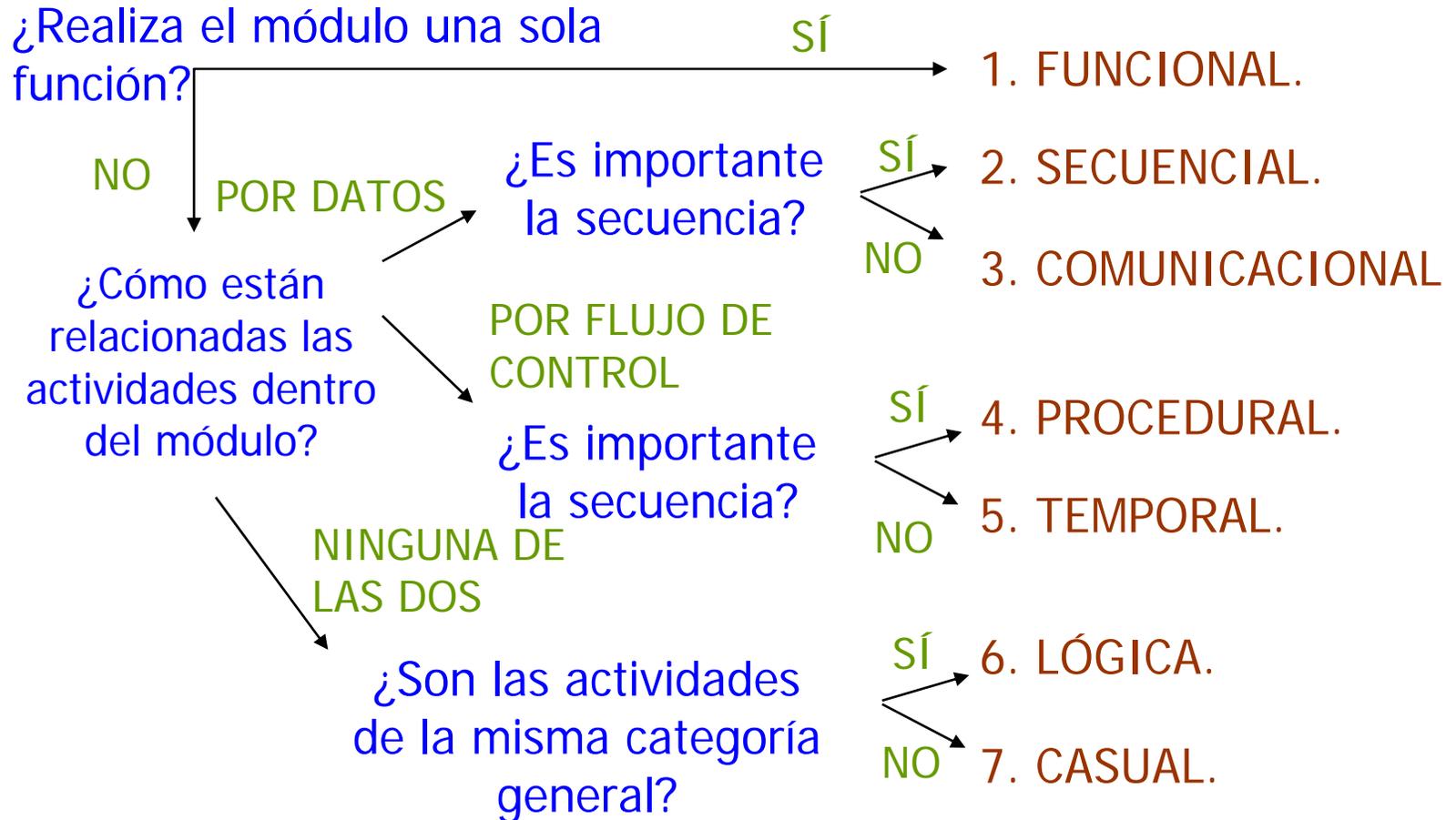
módulos de inicialización y terminación

Lógica:

rutina general de E/S

⇒ más ejemplos en (Piattini et al. 04) o (Molina et al. 97) p.78.

Determinación de la cohesión de un módulo (Molina et al. 97)



5. Heurísticas de diseño estructural



- **IMPORTANTE:** Los módulos superiores deben coordinar, y los inferiores realizar las tareas.
- Reducir el nº de parámetros que intercambian los módulos tanto como sea posible.
- Nunca pasar registros que contengan muchos campos cuando en realidad el módulo sólo necesita algunos.
- No agrupar parámetros sin relación en un registro.
- Evitar que un módulo hijo dé órdenes al padre.
- En la medida de lo posible, no usar variables globales.
- Se puede parar la descomposición cuando la interfaz con un módulo sea tan complicada como el módulo mismo.

Heurísticas de diseño estructural



- Debe evitarse la situación en que un módulo llama a muchos otros (puede ser difícil de entender).

Normalmente sucede cuando no se tiene en cuenta los niveles intermedios

⇒ Solución: combinar funciones subordinadas en una sola

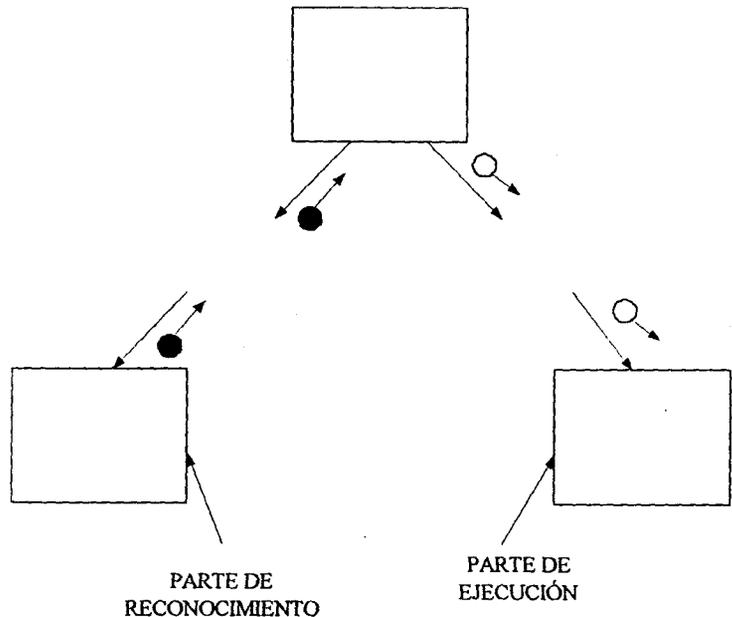
Excepción: cuando el módulo es un centro de transacciones

- Deben evitarse largas secuencias lineales. Soluciones:
 - (a) revisar las posibilidades de descomponer las funciones en subfunciones con entidad propia.
 - (b) comprimir módulos subordinados en un módulo de nivel superior.

Heurísticas de diseño estructural

(Molina et al. 97) cap. 6

- Factorizar funciones cuando sea posible.
- Inicializar las variables lo más tarde posible en el diagrama, cerca de las funciones a realizar.
- Evitar la **disgregación de decisiones**:



La parte de reconocimiento de la condición se separa de la parte de ejecución

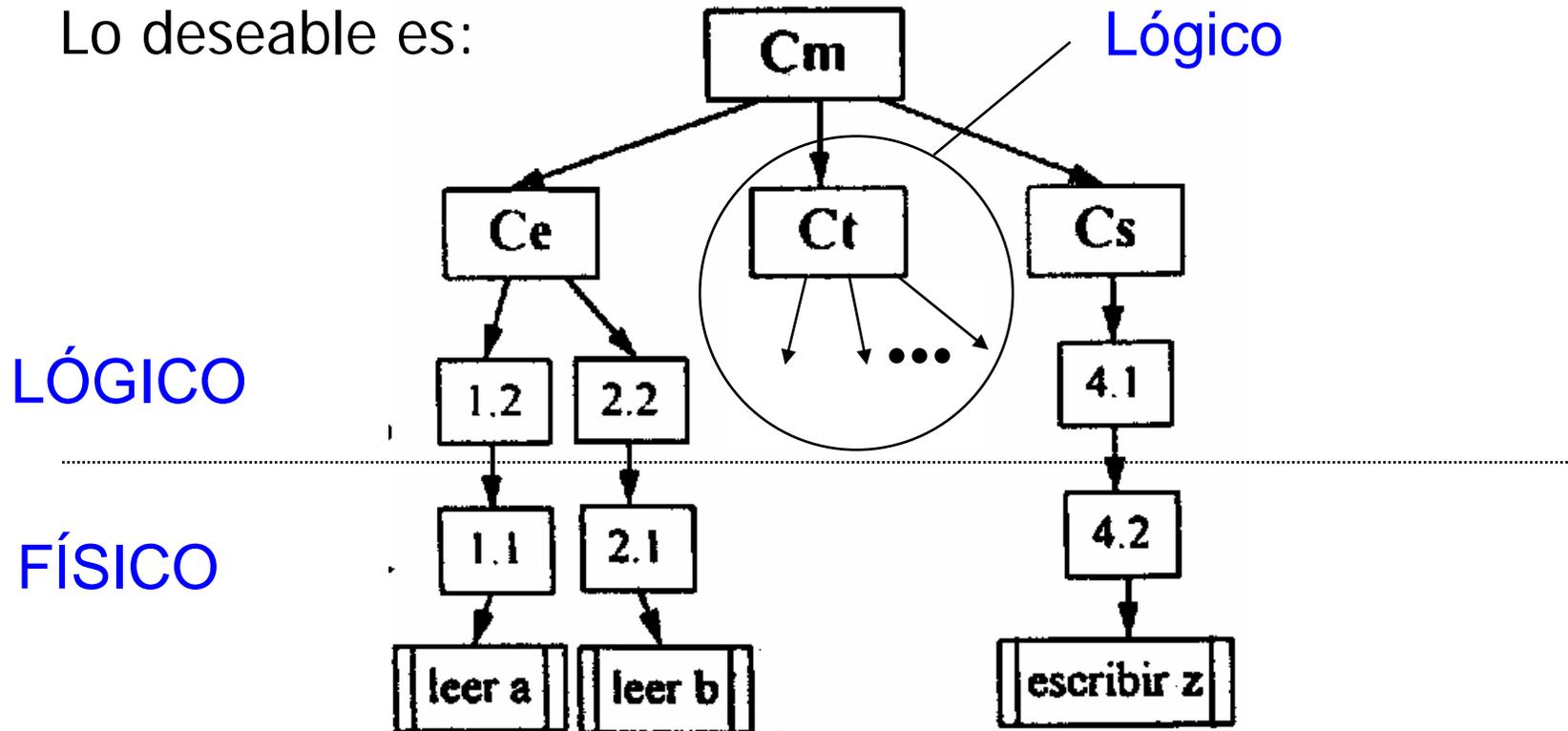
⇒ ¿Cuál puede ser un síntoma?

Heurísticas de diseño estructural (II)

(Molina et al. 97) cap. 6

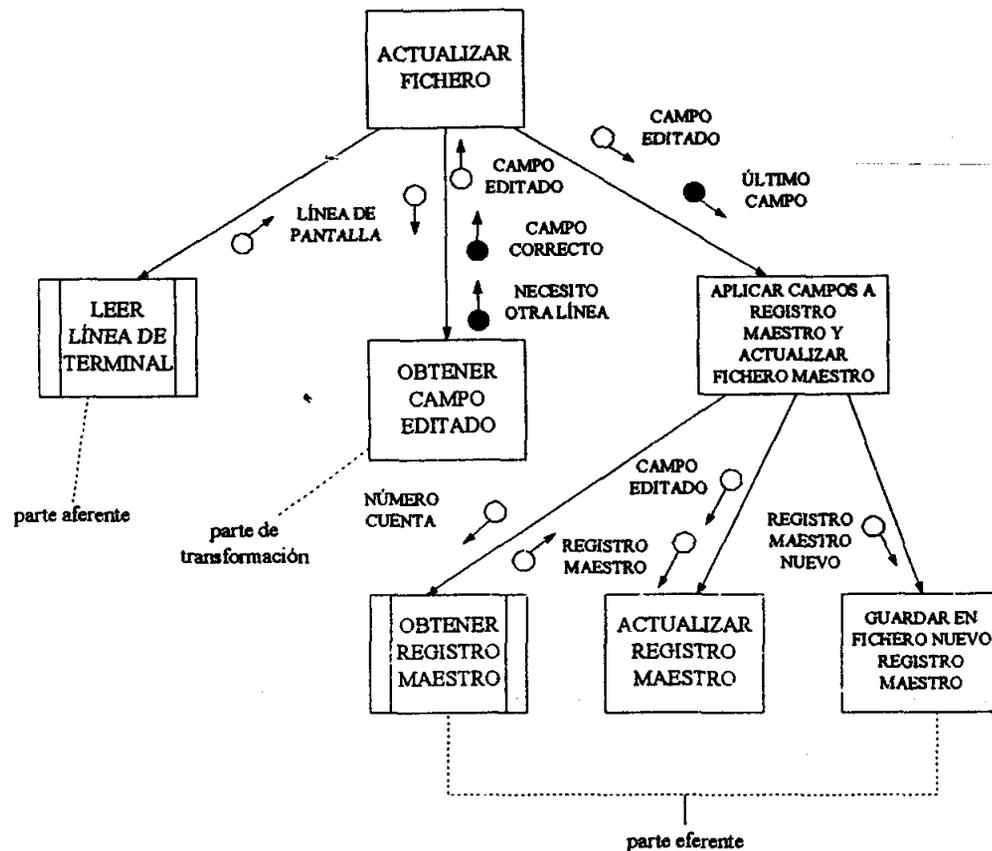
- Evitar **sistemas dirigidos por la entrada física** (sistemas que no procesan suficientemente la entrada).

Lo deseable es:



Heurísticas de diseño estructural (III) (Molina et al. 97) cap. 6

■ Ejemplo de sistema dirigido por la entrada física



- El acoplamiento suele ser alto.
- Ya que los módulos superiores en la jerarquía tratan con la entrada física, cualquier cambio en la especificación de ésta afectará a gran parte del sistema.

Heurísticas de diseño estructural (IV) (Molina et al. 97) cap. 6



- Edición de los datos de entrada
 - Validar sintaxis antes que semántica
 - Validar lo sencillo antes que lo cruzado

Ejemplo: “nombre-ciudad” y “cod-postal”

Validar:

1º sintaxis

nombre-ciudad → caracteres alfabéticos

cod-postal → numérico

2º Verificar que ambos datos existen

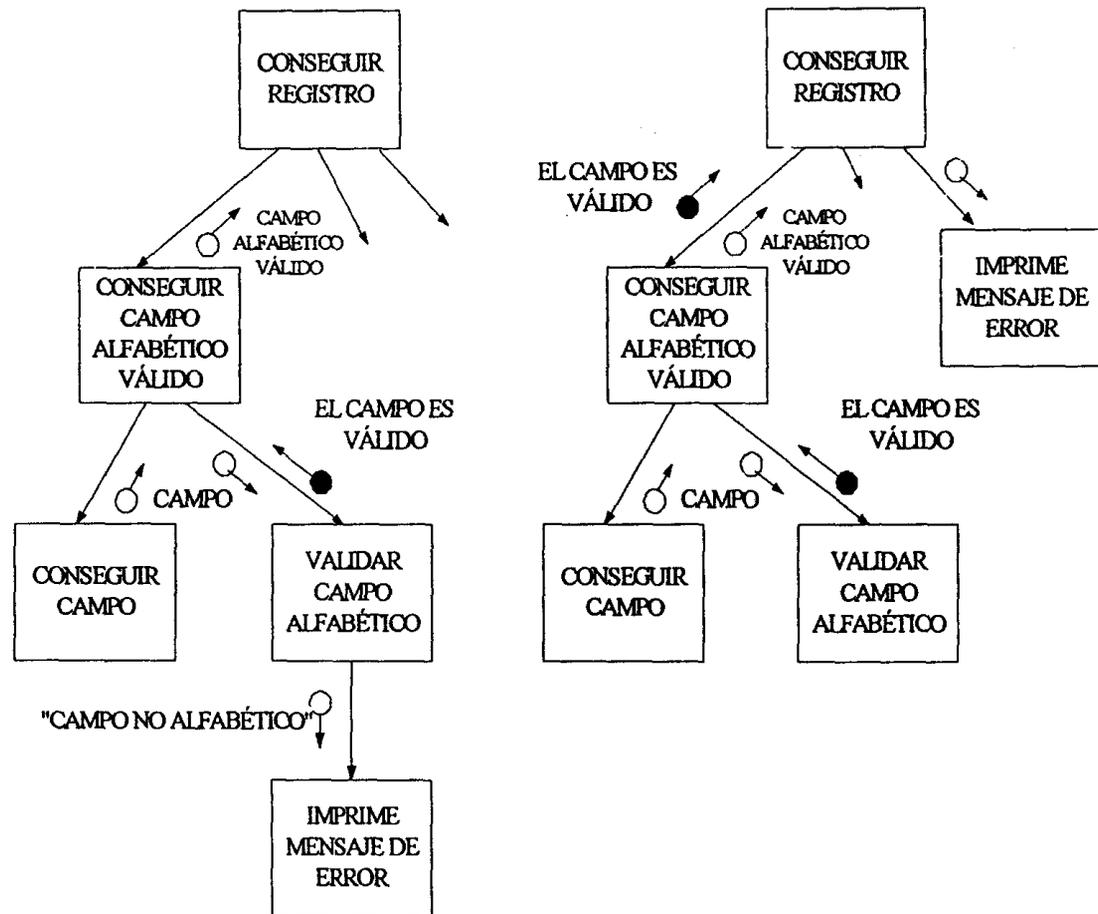
3º Validación cruzada: cod-postal \in nombre-ciudad

Heurísticas de diseño estructural (V) (Molina et al. 97) cap. 6

■ Información de los errores:

- *¿Qué módulo llama al módulo que escribe mensajes de error?*

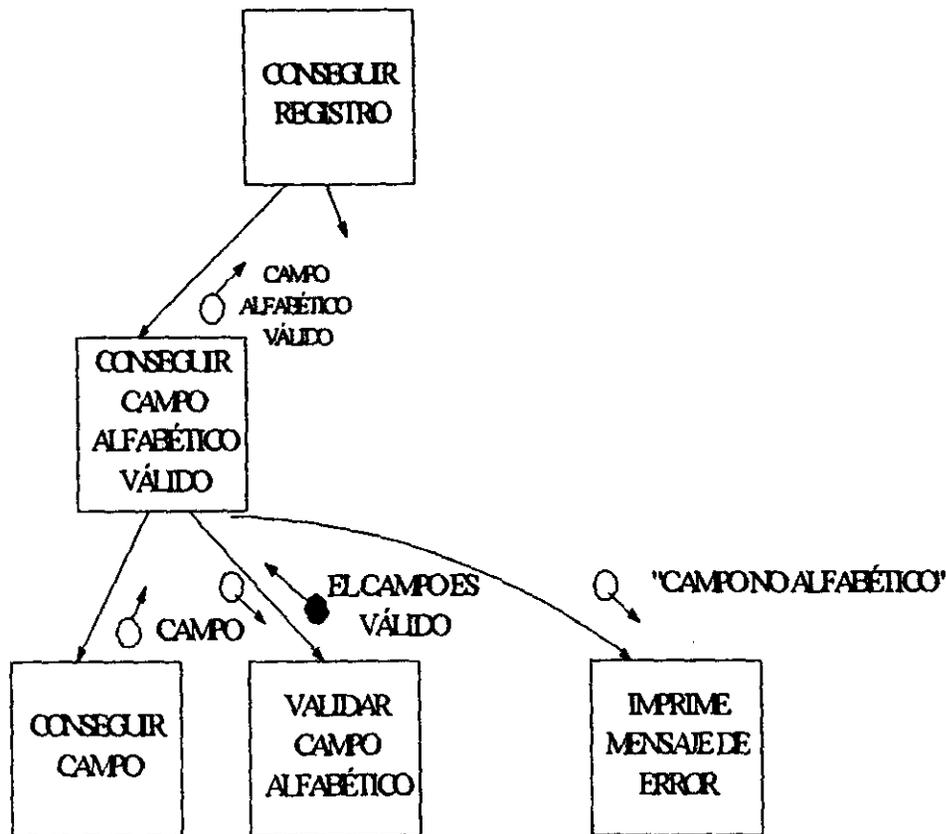
Dos soluciones no válidas:



Heurísticas de diseño estructural (VI)

(Molina et al. 97) cap. 6

Solución *válida*:



¿Dónde se pone el texto de los mensajes de error?

- *Diseminados por el sistema*
- *Dentro de un único módulo*