2.5 Modelo Funcional

El modelo de objetos describe las propiedades estructurales del sistema. El modelo dinámico y funcional describen su comportamiento.

- El modelo funcional describe los comportamientos y operaciones de los objetos.
- El modelo funcional muestra la dependencia de datos en el sistema.
- El modelo funcional describe la computación dentro del sistema, cómo los valores de salida se derivan de los valores de entrada, sin importar el orden en que son computados.
- El modelo funcional consiste de múltiples *diagramas de flujo de datos*.
- El modelo de objetos inicialmente incluye solo algunas operaciones de alto nivel definidas en ciertas clases.
- Las clases que tienen operaciones y comportamientos complejos son modelados funcionalmente.
- Construir un modelo funcional para estas clases ayuda a identificar las otras clases con las cuales se interacciona y con las cual existe alguna dependencia.
- Creando un modelo funcional también ayuda a identificar nuevos objetos, atributos, y operaciones.
- Los objetos, atributos, y operaciones identificadas deben ser luego añadidas al modelo de objetos
- El modelo funcional provee un medio para identificar la restricciones operacionales del mundo real.
- No todos los sistemas tienen un modelo funcional importante, al igual que no todos los sitemas tienen un modelo dinámico importante. En cualquier tipo de sistema el modelo de objetos es por lo general el más importante.

Ejemplo: Los programas no interactivos, como compiladores, tienen un modelo dinámico sencillo, ya que sólo computan una función, siendo el modelo funcional el más importante.

Ejemplo: Las bases de datos suelen tener modelos funcionales sencillos, ya que su especialidad es almacenar y organizar datos, y no transformarlos.

Ejemplo: La hoja de cálculo es un tipo de modelo funcional en el que los valores se definen de otros valores.

Ejemplo: El código de impuestos es una descripción funcional, que especifica fórmulas para computar impuestos basados en ganancias, gastos, etc.

El modelo funcional se define por medio de:
1) Diagramas de Flujos de Datos
2) Especificación de Procesos

2.5.1 Diagramas de Flujo de Datos y sus Componentes

El diagrama de flujo de datos es útil para mostrar la funcionalidad de un sistema a diferentes niveles. Los diagramas pueden estar anidados a cualquier profundidad, y el conjunto completo de diagramas anidados forma un árbol, y la descripción completa del modelo funcional.

- El *diagrama de flujo de datos* contiene *procesos* que transforman datos, *flujo de datos* que mueven datos, objetos *actores* que producen y consumen datos y *almacenamiento de datos* que guardan datos.

- Los *diagramas de flujos de datos* proveen una descripción detallada de las operaciones definidas en el modelo de objetos y las actividades definidas en el modelo dinámico. Los atributos de los objetos son modelados como *flujos de datos* en el modelo funcional.

- Los *diagramas de flujos de datos* representan los datos y procesos a varios niveles de descomposición, describiendo en el nivel más bajos las funciones simples del sistema. El nivel de la jerarquía depende de la aplicación.

Los diagramas de flujos de datos están compuestos por:
1) Procesos
2) Flujo de datos
3) Almacenamiento de datos
4) Actores
5) Flujo de Control

2.5.1.1 Procesos

Los *procesos* representan transformaciones de valores de datos.

- Los procesos son implementados como métodos correspondiendo a las operaciones, o parte de ellas, en las clases de objetos.

- Los resultados precisos de un proceso dependen del comportamiento del sistema, especificado en el modelo dinámico. Los procesos corresponden a las actividades en los diagramas de estado del modelo dinámico. (El modelo funcional también captura las acciones del modelo dinámico, como las consultas.)

- Los procesos en el modelo funcional correponden a las operaciones (y métodos) en el modelo de objetos
  - Los procesos de alto nivel corresponden a operaciones en objetos compuestos
  - Los procesos de bajo nivel corresponden a operaciones en objetos básicos

- Un proceso de alto nivel puede ser expandido en otros diagramas de flujo de datos de nivel más bajo. Cada entrada y salida de un proceso es una entrada o salida de un nuevo diagrama.

- Eventualmente los procesos de nivel más bajo son descritos como *funciones*, sin efectos secundarios.

Ejemplo de procesos *funcionales*: la suma de dos números, el interés en una tarjeta de crédito, o la curva a través de una lista de puntos.

- Un proceso puede tener efectos secundarios si contiene componentes *no-funcionales*, como almacenamiento de datos los cuales pueden modificar los valores de la memoria.

Ejemplo de procesos *no-funcionales*: leer y escribir archivos, algoritmos para reconocimiento de voz que aprenden por experiencia, despliegue de imágenes en una pantalla.

- Un gráfico completo de flujo de datos se puede definir como un proceso de nivel alto.

<u>Notación OMT</u>

Un proceso en el diagrama de flujo de datos se dibuja como una elipse conteniendo una descripción de la transformación, usualmente su nombre.

La notación se muestra en la Figura 2.205

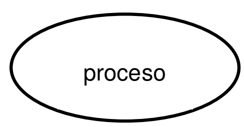

Figura 2.205. Notación para *procesos*.

Ejemplo: Un proceso para la divisón de números enteros se muestra en la Figura 2.206.

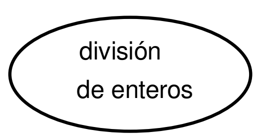

Figura 2.206. Proceso para la división de números enteros.

2.5.1.2 Flujo de Datos

Los *flujos de datos* representan las entradas y salidas de las transformaciones y computaciones de los procesos. Los flujos de datos también representan los valores intermedios dentro de una computación. (El valor intermedio no es cambiado por el flujo de datos.)

- Cada proceso tiene un número de datos de entrada, y un número de datos de salida.

- Cada flujo de datos representa un valor en algún punto de la computación. Flujos internos al diagrama representan valores intermedios en la computación y no tienen necesariamente significado en el mundo real. Flujos en los bordes del diagrama de flujo de datos son sus entradas y salidas.

- El flujo de datos en el modelo funcional puede corresponder a objetos, valores de atributos, o valores puros sin identificadores.

- Los flujos de datos discretos pueden corresponder a eventos en el modelo dinamico.

- Los dato pueden tener diferente formato

Ejemplo: *entero*, *cadena*.

- Pueden aplicarse diferentes transformaciones a los datos: copia, combinación y descomposición.

Ejemplo: Los datos se pueden dividir de un agregado a sus componentes, y viceversa.

<u>Notación OMT</u>

La flecha es etiquetada con la descripción del dato, usualmente su nombre o tipo. El flujo de datos es dibujado como una flecha entre el productor y el consumidor del dato. Estos flujos pueden no estar conectados (si el diagrama es parte de un sistema completo) o pueden estar conectados a un objeto.

La notación se muestra en la Figura 2.207

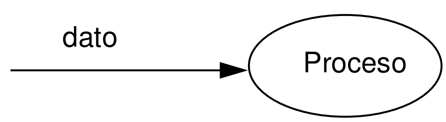

Figura 2.207. Notación para *procesos* con *flujo de datos*.

Ejemplo: Un proceso con flujo de datos para la *divisón de enteros* se muestra en la Figura 2.208. Las entradas en el diagrama de la *división de enteros* son *dividendo* y *divisor*, y las salidas son *cociente* y *resto*.

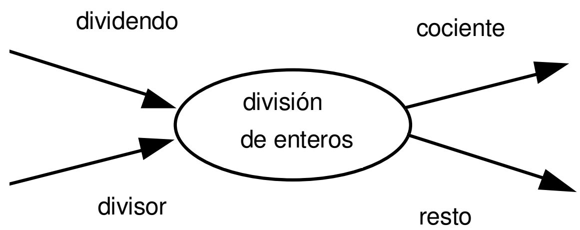

Figura 2.208. Proceso con flujo de datos para la división de números enteros.

Copia de Valores

El mismo valor puede ser mandado a varios lugares, eso se indica con varias flechas emergentes, como se muestra en la Figura 2.209. (La flecha de salida no tiene etiqueta si es igual a la que causa la división.)

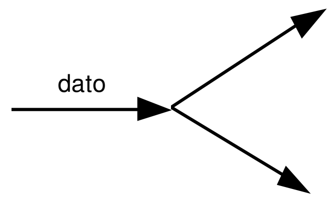

Figura 2.209. Notación para la replica de datos.

Ejemplo: Para replicar un número a varias procesos basta con crear un diagrama de flujos como se muestra en la Figura 2.210.

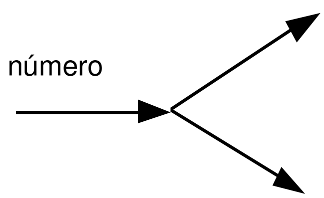

Figura 2.210. Replica de un *número*.

Descomposición de Valores Agregados

Una notación similar se utiliza para separar el flujo de un agregado en sus componentes, donde cada valor puede ir a diferentes procesos, con la diferencia que todas las flechas son etiquetadas, como se muestra en la Figura 2.211.

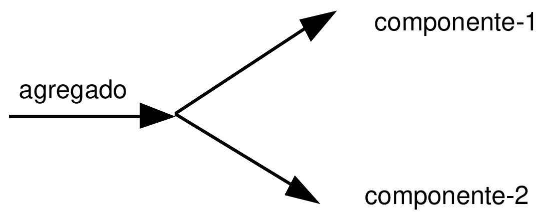

Figura 2.211. Notación para la descomposición de un agregado en sus componentes.

Ejemplo: *Dirección* es un agregado que se puede descomponer en *calle*, *ciudad*, *estado*, y *código postal*, como se muestra en la Figura 2.212.

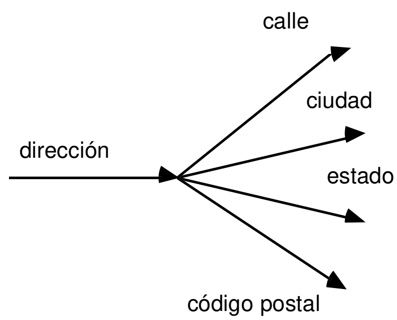

Figura 2.212. El agregado de datos *dirección* es dividido en sus componentes, *calle*, *ciudad*, *estado*, y *código postal*.

Combinación de Valores Agregados

El opuesto de descomponer un agregado es el combinar sus componentes, donde los diferentes valor que pueden venir de diferentes procesos, se mandan en conjunto a uno solo. La notación se muestra en la Figura 2.213.

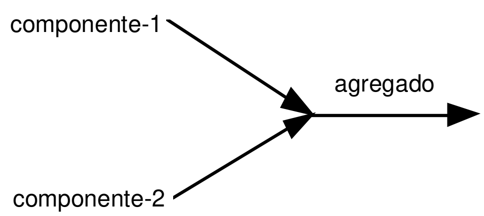

Figura 2.213. Notación para la combinación de los componentes en un agregado.

Ejemplo: Se pueden combinar *calle*, *ciudad*, *estado*, y *código postal* para formar el agregado *dirección*, como se muestra en la Figura 2.214.

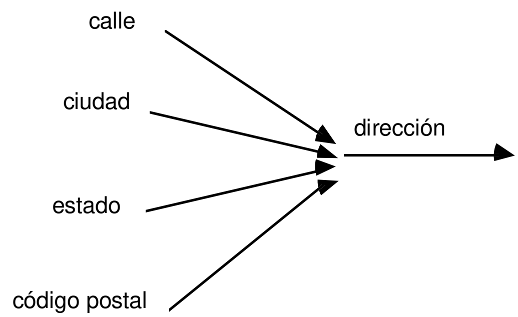

Figura 2.214. Se combinan los componentes, *calle*, *ciudad*, *estado*, y *código postal* en el agregado *dirección*.

Generación de Objetos

El diagrama de flujo puede mostrar la generación de un objeto de un simple dato. La notación se muestra en la Figura 2.215.

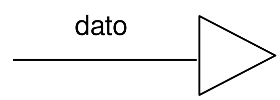

Figura 2.215. Notación para la generación de un objeto.

Ejemplo: Se puede generar un nuevo *objeto cuenta* de un *dato cuenta*, como se muestra en la Figura 2.216.

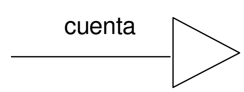

Figura 2.216. Generación de un objeto *cuenta* de un dato *cuenta*.

2.5.1.3 Almacenamiento de Datos

Los *almacenamientos de datos* son objetos "pasivos" que guardan datos para su acceso posterior. El almacenamiento de datos no genera sus propias operaciones, sólo responde a pedidos de almacenar y accesar datos, permitiendo a los procesos accesar valores en diferente orden del que han sido guardados.

Ejemplos: sistema de reservaciones, cuentas de banco.

Notación OMT

El almacenamiento de datos es dibujado como un par de líneas paralelas conteniendo el nombre del almacenamiento en el medio, como se muestra en la Figura 2.217.

Figura 2.217. Notación para un *almacenamiento de datos*.

Ejemplo: Una cuenta bancaria se muestra en el diagrama de la Figura 2.218.

Figura 2.218. Almacenamiento de datos para una *cuenta* bancaria.

<u>Acceso a Almacenamiento de Datos</u>
Las flechas de entrada indican información u operaciones que modifican los datos.

Ejemplo: *añadir elementos*, *modificar valores*, o *remover elementos*.

Las flechas de salida indican información leída del almacenamiento.

Ejemplo: *ver elementos*.

Se pueden leer valores enteros o componentes del almacenamiento de datos. La estructura del almacenamiento de datos debe ser descrita en el modelo de objeto, junto con la descripción de los accesos permitidos y su actualización.

Ejemplo: El almacenamiento de datos para una cuenta bancaria se muestra en la Figura 2.219.

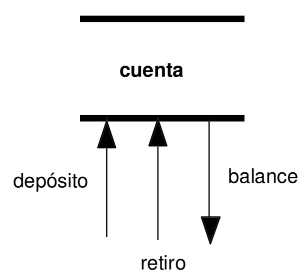

Figura 2.219. Almacenamiento de datos para una *cuenta* de un banco, donde están permitidas los accesos: *depósito*, *retiro*, y *balance*.

Se puede utilizar una flecha doble como entrada y salida a la vez.

Ejemplo: En la Figura 2.220 se muestra una variante para el almacenamiento de una cuenta bancaria donde *dinero* sirve de entrada y salida a la cuenta.

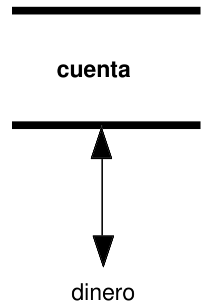

Figura 2.220. Almacenamiento de datos para una *cuenta* de un banco, donde *dinero* sirve de entrada y salida.

Ejemplo: Una lista de precios para artículos se muestra en la Figura 2.221. La entrada al almacenamiento de datos consiste de artículos y costo. El proceso *encontrar costo* recibe como entrada el *nombre del artículo* el cual se compara a los artículos guardados en la *lista de precios* para saber su *costo*. La flecha no etiquetada de salida del almacenamiento de datos indica que toda la información guardada en la lista de precios es accesada por el proceso *encontrar costo*.

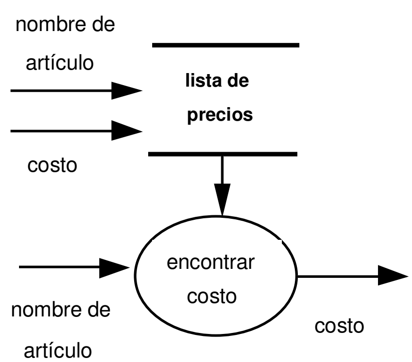

Figura 2.221. Almacenamiento de datos para una lista de precios.

Ejemplo: La *tabla periódica* para accesar el peso atómicos de los diferentes elementos químicos se muestra en la Figura 2.222. Las propiedades de los elementos químicos son constantes y no varían en el programa. Es conveniente representar el proceso *encontrar peso* como un simple acceso a un almacenamiento de datos constante.

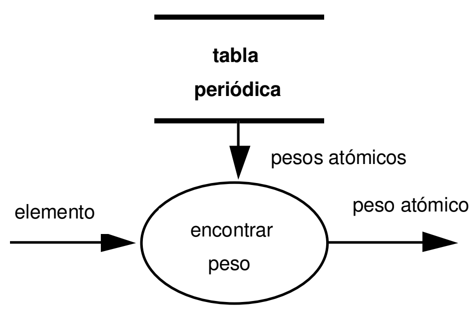

Figura 2.222. Almacenamiento de datos para una *tabla periódica*.

- Un almacenamiento de datos puede contener valores sencillos u objetos.

- Un triángulo vacío indica un valor de flujo de datos que es tratado como un objeto, generando el objeto como entrada a un almacenamiento de datos, como se muestra en la Figura 2.223.

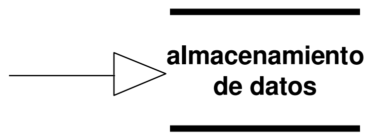

Figura 2.223. Almacenamiento de datos creado como un objeto.

Ejemplo: El resultado del proceso *crear cuenta* en un banco es una nueva cuenta que es almacenada en el banco. El nombre del cliente y el depósito son guardados en la cuenta. El objeto cuenta se ve a la vez como un valor de dato guardado en el banco, y como un almacenamiento de dato usado para guardar y leer valores. El diagrama se muestra en la Figura 2.224.

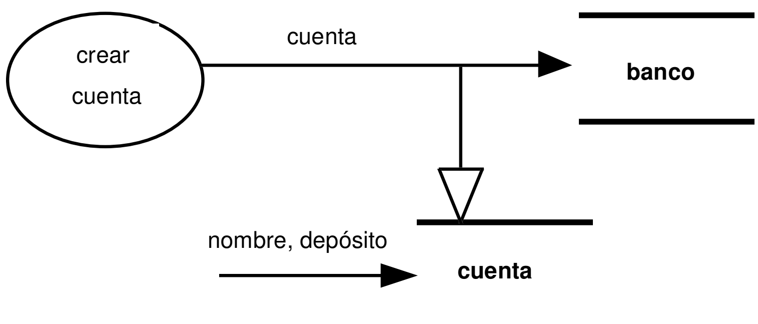

Figura 2.224. Diagrama de flujo de datos para crear una nueva cuenta.

2.5.1.4 Actores

Los *actores* son objetos activos que producen y consumen datos, representando fuentes y terminales en el flujo de datos del sistema.

Ejemplo: El usuario en un sistema de cómputo se considera un agenete externo, o actor.

- Los actores deben corresponder a objetos en el modelo de objetos.

- Las operaciones de los actores están fuera del diagrama y deben ser descritas como parte del modelo dinámico.

- Aunque los actores y almacenamiento de datos se consideran ambos objetos, estos se distinguen por su comportamiento y utlización dentro del modelo funcional.

Ejemplo: Un almacenamiento de datos se puede implementar como un archivo, mientras que un actor se puede implementar como un dispositivo externo.

Notación OMT

El actor se dibuja como un rectángulo en el diagrama de flujo de datos, como se muestra en la Figura 2.225.

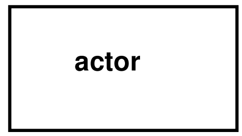

Figura 2.225. Notación para *actor*.

Ejemplo: El cliente es un actor escogiendo una cuenta del banco. El resultado de la operación es el propio objeto de la cuenta, que es luego usado como un almacenamiento de datos en la operación de actualización. El diagrama de flujo de datos se muestra en la Figura 2.226.

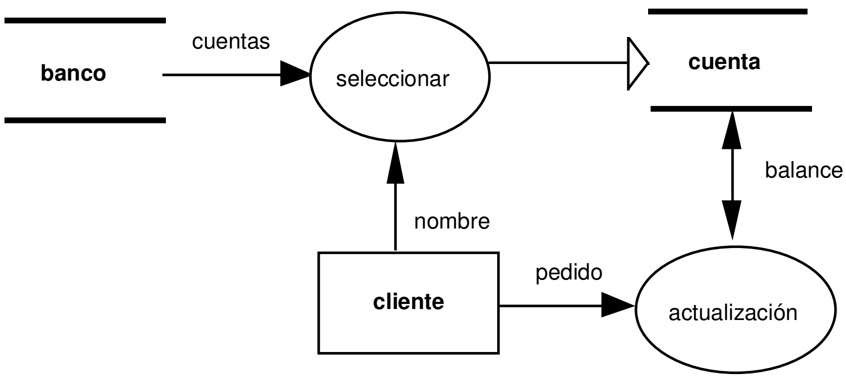

Figura 2.226. Diagrama de flujo de datos para seleccionar una cuenta.

2.5.1.5 Flujo de Control

El control de las operaciones, la decisión de cuando se ejecutan, están descritas por el modelo dinámico. Es a veces útil incluir una descripción de control en el modelo funcional, para mostrar la dependencia de los datos. Esto se puede hacer por medio de un *flujo de control* en el diagrama de flujo de datos, que es un valor *booleano* controlando si un proceso es evaluado. No es un valor de entrada al propio proceso.

Notación OMT

El control se muestra como una línea punteada del proceso produciendo el valor booleano al proceso que es controlado, como se muestra en la Figura 2.227.

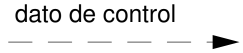

Figura 2.227. Notación para *control de flujo*.

Ejemplo: El diagrama de flujo de datos para el retiro de una cuenta de banco se muestra en la Figura 2.228. El cliente entra un código y una cantidad. El retiro ocurre solamente si el código es correcto.

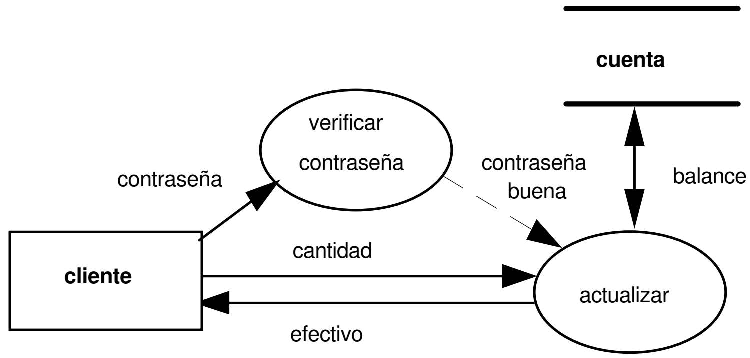

Figura 2.228. Diagrama de flujo de datos con flujo de control para el retiro de una cuenta.

2.5.2 Diagrama de Contexto

Los *diagramas de contexto* muestran los bordes del sistema, describiendo todas las entradas y salidas entre los actores y el proceso principal de las clases principales en el sistema.

- El diagrama de contexto describe el proceso de más alto nivel, el proceso principal o "0".

<u>Notación OMT</u>

La notación general se muestra en la Figura 2.229.

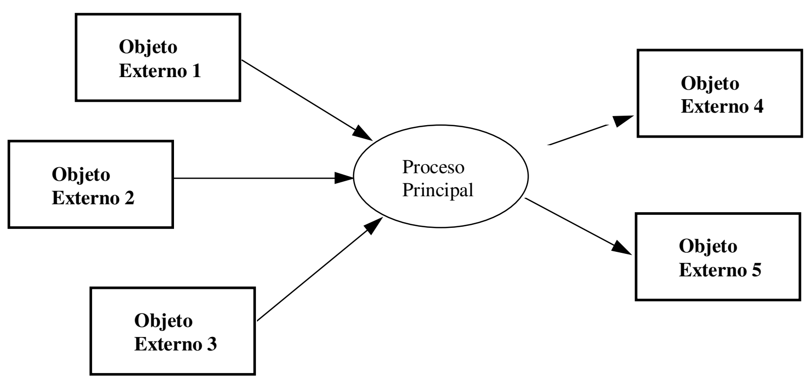

Figura 2.229. Notación para un *diagrama de contexto*.

Ejemplo: Un *cliente* es un objeto externo que interactúa con el proceso principal del *banco*, *crear cuenta*, como se muestra en la Figura 2.230.

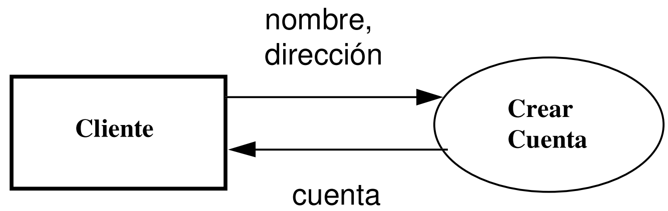

Figura 2.230. *Cliente* es un actor en el diagrama de contexto para un *banco*.

2.5.3 Descomposición de Diagramas de Flujo de Datos

Los *diagramas de flujo de datos* muestran la descomposición funcional del sistema en varios niveles.

- El diagrama de flujo de datos de más alto nivel describe el proceso "0", correspondiente al diagrama de contexto.

- Se descompone un proceso en múltiples procesos de más bajo nivel. Todas las entradas y salidas del proceso antecesor deben estar representadas.

- Los subdiagramas correspondientes a procesos de más bajo nivel se asignan el número del proceso antecesor. Los procesos dentro del subdiagrama se numeran incrementalmente con notación de punto.

Ejemplo: Un proceso descendiente de un proceso "1" se llamaría "1.x".

- El esquema de numeración es el del número de nivel usado para identificar los párrafos dentro de un documento.

- Los procesos que no se descomponen más, se representan con una *especificación de proceso*. Los procesos de más bajo nivel son funciones puras sin efectos secundarios.

- Un proceso puede tener efectos secundarios si incluye un almacenamiento de datos u objetos externos.

- Los almacenamientos de datos deben aparecer en el nivel más alto donde sirven como interface entre procesos.

Ejemplo: Si un almacenamiento interactúa con procesos que solo aparecen en el nivel "3", en ese nivel se incluiría el almacenamiento.

- Las interacciones pueden estar ocultas en los diagramas de alto nivel y deben ser capturadas en las descomposiciones de más bajo nivel.

Notación OMT

La notación para el diagrama de flujo de datos "0" (DFD0 - Diagrama de Flujo de Datos 0), correspondiente al diagrama de contexto (pero sin incluir los actores) se muestra en la Figura 2.231.

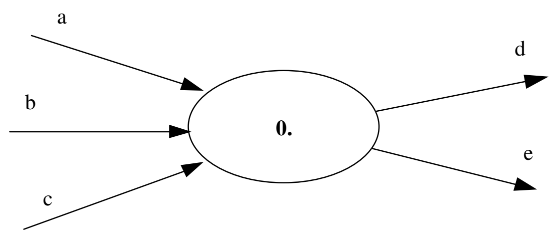

Figura 2.231. Notación para un diagrama de flujo de datos "0".

El siguiente diagrama es la descripción del proceso "0" (DFD1) en un diagrama de más bajo nivel con múltiples procesos, como se muestra en la Figura 2.232. Se deben *balancear* las entradas y salidas, para que sean las mismas que en el proceso de más alto nivel.

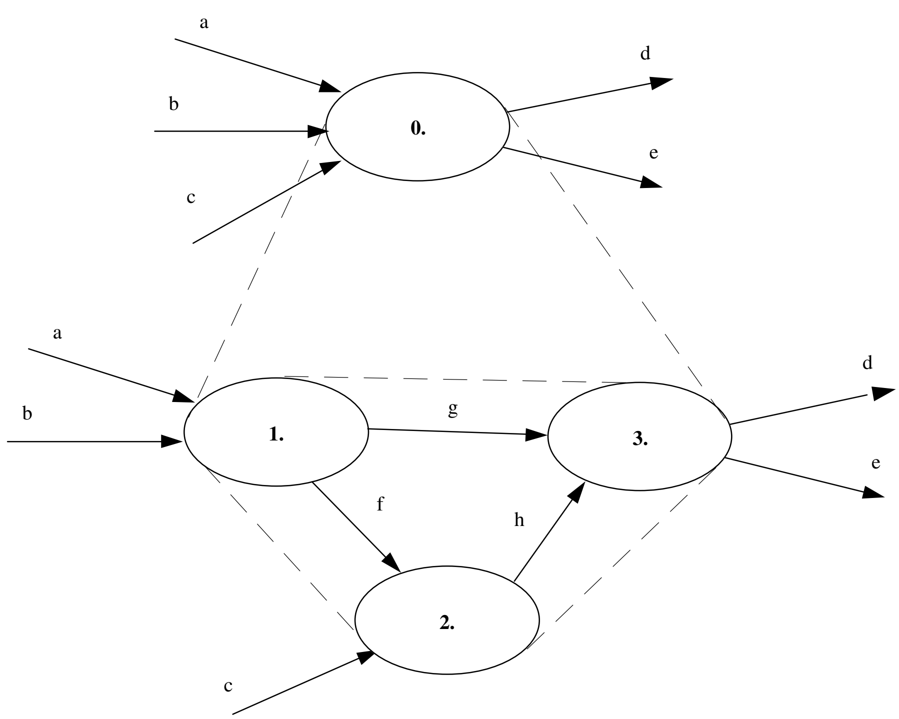

Figura 2.232. Notación para un diagrama de flujo de datos "1".

Los siguientes descomponen los procesos con con múltiples procesos de más bajo nivel. En la Figura 2.233 se muestra la descomposición del proceso "1" en dos nuevos subprocesos. Se deben también *balancear* las entradas y salidas, para que sean las mismas que en el proceso de más alto nivel. Los procesos "1" y "3" se pueden descomponer de manera similar.

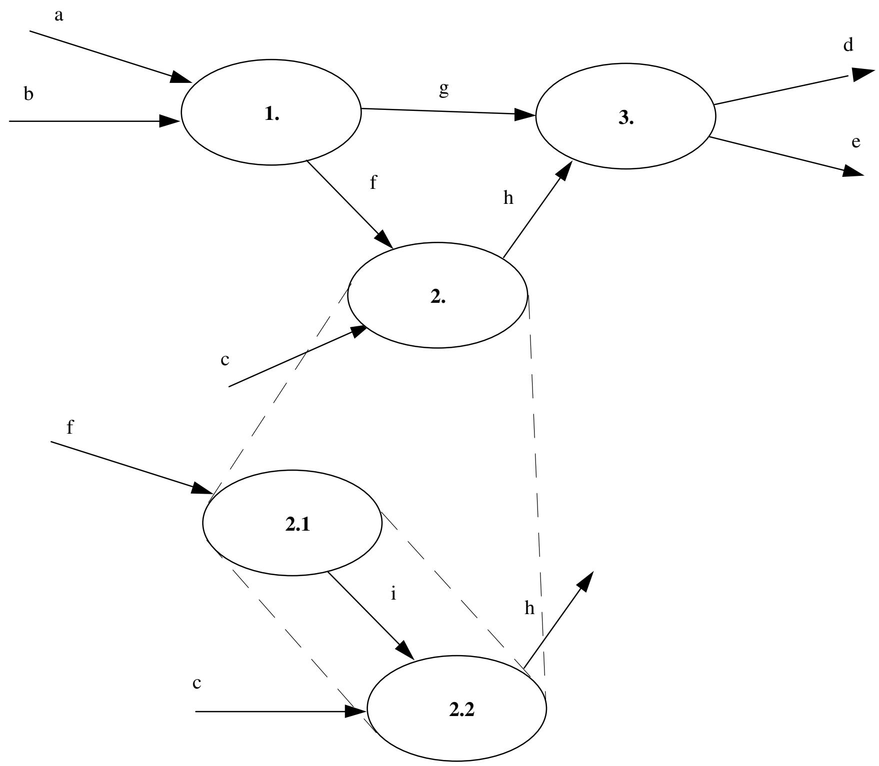

Figura 2.233. Notación para diagrama de flujo de datos intermedios.

Valores Agregados

Cuando los valores de agregados en un nivel se descomponen en sus valores componentes en el siguiente nivel es necesario incluir esta información en el diccionario de datos, como se muestra en la Figura 2.234.

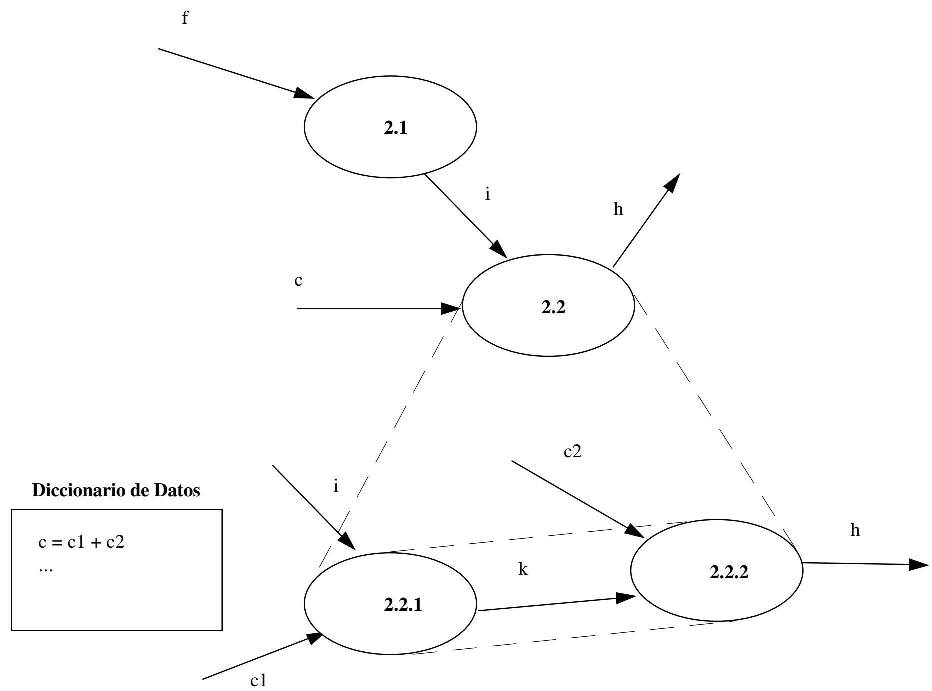

Figura 2.234. Notación para diagrama de flujo de datos intermedios descomponiendo valores agregados.

2.5.4 Especificación de Funciones

La *especificación de funciones* es generada para cada función simple, conteniendo descripciones y restricciones funcionales.

- La especifición de función se escribe para los procesos de más bajo nivel a menos que el proceso sea trivial.
- Una buena especificación de procesos permite su implementación como métodos de operaciones en las clases de objetos.
- El modelo funcional de una clase de objetos debe contener procesos para las operaciones de alto nivel definidas en la clase del objeto.
- El modelo funcional descompone los procesos en suboperaciones las cuales pueden ser incorporados al objeto apropiado dentro del modelo de objetos.
- Los flujos de datos suelen corresponder a valores de los atributos del objeto el cual envía los datos.
- Nuevos objetos suelen identificarse durante el proceso de decomposición funcional. Los nuevos objetos deben ser añadidos al modelo de objetos.
- Los procesos del nivel más bajo corresponden a operaciones.

- Los procesos de nivel alto pueden también ser considerados operaciones, aunque su implementación puede ser diferente a su representación.

- Las acciones, actividades, y eventos identificados dentro del modelo dinámico pueden ser descompuestos dentro del modelo funcional.

- La *acción* es definida en término de actualizaciones a atributos y ligas. Las acciones se derivan de procesos del modelo funcional.

Ejemplo: *Borrar una figura de la pantalla* es una acción, ya que se considera instantánea y tiene efectos secundarios.

- La *actividad* se especifica en los modelo dinámicos y funcionales.

Ejemplo: Un *daemon* en un sistema operativo, como un *spooler*, se le considera un actor ya que tiene un rol activo en el control del flujo de información, y su operación se considera una actividad ya que tiene duración y efectos secundarios.

- Los evento enviado a un objeto corresponde a una operación en el objeto. Dependiendo de la arquitectura del sistema, los eventos pueden ser implementados directamente, incluyendo un *manejador de eventos* como parte del sistema, o pueden ser implementados como métodos explícitos. Estas decisiones se toman durante la etapa de diseño del sistema.

2.5.4.1 Descripciones Funcionales

La descripción funcional es parte de la especificación del proceso.

- Se debe escribir una descripción para cada función en el sistema.

- Se puede usar diferentes formatos para definir la función.

- Se debe enfocar en lo que la función hace y no en la implementación.

- Se debe especificar que sucede para todos los posibles valores de la función.

- La descripción de la función puede ser declarativa o procedural.

- Una descripción declarativa especifica las relaciones entre los valores de entrada y salida.

Ejemplo: la descripción de la función
ordenar y remover valores duplicado
puede ser descrita de forma declarativa como
cada valor en la lista de entrada aparece una sola vez en la lista de salida,
la lista de salida contiene solo valores de la lista de entrada,
y los valores en la lista de salida son estrictamente en orden ascendente.

- Una descripción procedural especifica las funciones por medio de un algoritmo. El propósito del algoritmo es especificar de forma sencilla, aunque puede que ineficiente, lo que la función hace. Esto no implica que el programa deba usar el mismo algoritmo, sólo que el resultado debe ser idéntico. Durante la implementación puede substituirse por un algoritmo más eficiente que compute lo mismo. Algunas funciones pueden ser especificadas sin dar un algoritmo preciso.

Ejemplo: Invertir una matriz A es definido como

la matriz B, por la cual A por B da la matriz de identidad.

Multiplicación de matrices es fácil de definir, pero derivar un algoritmo para computar el inverso requiere bastante álgebra linear. Dar el algoritmo es parte del diseño.

- Descripciones declarativas son preferibles a las procedurales, ya que no implican una implementación, pero las procedurales se deben usar si son mas fáciles de escribir.

- La especificación de una operación incluye una *firma* y una *transformación*.
  - La *firma* define el interfaz a la operación, los argumentos que requiere (número, orden, tipos), y el valor que devuelve (número, orden, tipos).
  - La *transformación* define el efecto visible externamente de una operación, los valores de salida como funciones de los valores de entrada y los efectos secundarios sobre los objetos involucrados. Se indica lo que la operación hace y no cómo.

- La descripción de una función incluye el *nombre de la función*, las *salidas y entradas*, a *transformación* de valores, e *información de estado*.
  - Descripción de procesos los cuales contienen construcciones de control, como IF-THEN-ELSE, pueden identificar nuevos estados.
  - Nuevos estados identificados pueden ser definidos por algunos atributos que todavía no hayan sido identificados en el objeto, los cuales deben ser añadidos.
  - Nuevos estados identificados deben ser añadidos en el modelo dinámico.

Ejemplo: *conmutador de teléfono* para *conectar una llamada*. La descripción de la función no está dada, en este caso, como un algoritmo para determinar las conexiones. La especificación es informal y un poco ambigua, siendo una especificación inicial de nivel alto. No hay, por ejemplo, una especificación en detalle de la topología de la red.

Función: *conectar llamada*
Entradas: *línea telefónica*, *número discado*, *estado actual de los circuitos*
Salidas: *nuevo estado del conmutador*, *estado de la conexión*
Transformación: Conectar el teléfono que llama con el que es llamado por medio del cierre de conexiones en el conmutador, observando las siguientes restricciones:
Restricciones: Sólo dos líneas pueden estar conectadas en un circuito a la vez.
Las conexiones no deben ser interrumpidas.
Si la línea está en uso, el estado se reporta como ocupado y no se cierra el circuito.
Si la conexión es imposible de hacer porque están en uso demasiados circuitos, los circuitos no se cierran, y el conmutador se reporta como ocupado.

2.5.4.2 Restricciones Funcionales

- Las restricciones funcionales especifican condiciones adicionales que debe ser satisfechas por las operaciones. Se debe establecer los tiempos y condiciones de las restricciones.

- Una restricción puede ser:
  - Entre dos objetos a la vez,
  - Entre instancias de un mismo objeto en tiempos diferentes (*invariante*),
  - Entre instancias de diferentes objetos en diferentes tiempos.

- Los tipos de *restricción* son:
  - <u>Función Total</u>: Un valor es totalmente especificado por otro.

Ejemplo: frecuencia y longitud de onda.

  - <u>Función parcial</u>: Un valor es restringido, pero no completamente especificado por otro.

Ejemplo: La segunda ley de termodinámica expresa restricciones parciales, que la entropía o desorden del universo nunca puede disminuir.

  - <u>Pre-condiciones</u>: Restricciones que los valores de entrada deben satisfacer.
  - <u>Post-condiciones</u>: Restricciones que los valores de salida deben garantizar.
  - <u>Eficacia</u>: limitaciones de tiempo y fuentes.

- Las restricciones pueden aparecer en todos los modelos:
  - <u>restricción de objetos</u>: especificando qué objetos dependen total o parcialmente de otros objetos.
  - <u>restricción dinámica</u>: especificando la relación entre estados o eventos de objetos diferentes.
  - <u>restricción funcional</u>: especificando la relación entre operaciones.

- Cuando existen restricciones funcionales, las construcciones condicionales deben estar claramente indicadas en la descripción funcional.

2.5.4.3 Formatos

- Las funciones pueden ser especificadas de varias formas:
  - funciones matemáticas
  - tablas de valores de entrada y salida
  - ecuaciones de salidas y entradas
  - pre y post condiciones
  - árboles de decisión
  - tablas de decisión
  - pseudo código
  - lenguaje natural

- enumeraciones de posibles entradas

Ejemplo: Costo de membresía para una revista

- Estudiante (E)
  - Si ha sido miembro de la revista por menos de cuatro años: N$100
  - Si ha sido miembro de la revista por cuatro años o más: N$80
- Profesional (P)
  - Sobre la edad de 65 años: N$50
  - Si ha sido miembro de la revista por menos de cuatro años y es menor a 65 años: N$150
  - Si ha sido miembro de la revista por cuatro años o más y es menor a 65 años: N$100

(Menor 65 (J), Mayor o igual 65 (M); menos de 4 años (L), más de 4 años (G))

Tablas de Decisión

Ejemplo: Combinaciones válidas de condiciones, como se muestra en la Figura 2.235.

Condición	1	2	3	4	5
Tipo de Membresía	E	E	P	P	P
Duración de Membresía	L	G	L	G	-
Edad	-	-	J	J	M
Acción	100	80	150	100	50

Figura 2.235. Tabla de decisión.

Arboles de Decisión

Ejemplo: Arbol de decisión, como se muestra en la Figura 2.236.

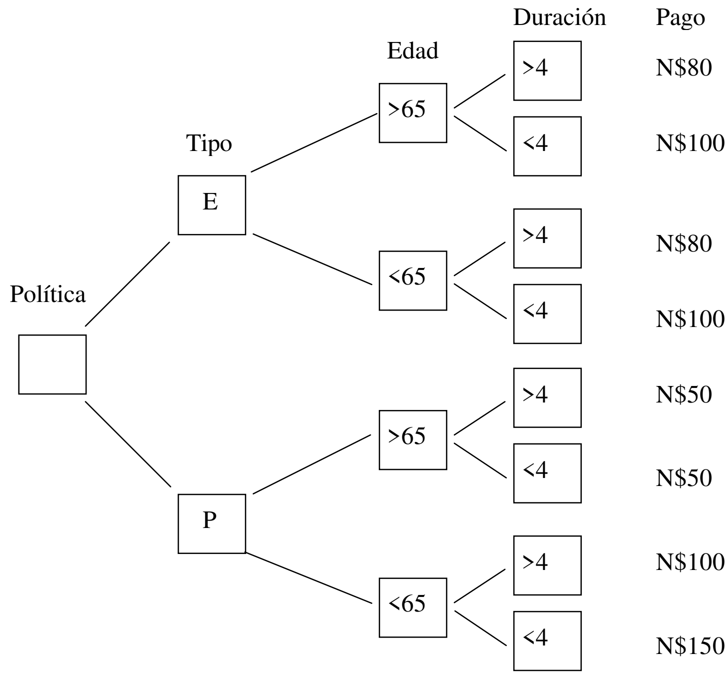

Figura 2.236. Arbol de decisión.

Lenguaje Natural

- Se debe usar un vocabulario mínimo
- Construcciones sintácticas limitadas a *secuencias*, *repetición*, *decisión* y *selección*.

Secuencia

- Una oración simple o compuesta

Repetición

- Oración o condición de repetición

Decisión

- IF condición THEN
  oración simple o compuesta

- IF condición-1 THEN
  oración simple o compuesta
  ELSE-IF condición-2 THEN
  oración simple o compuesta
  ...
  ELSE condición-N
  oración simple o compuesta

Selección

- SELECT caso
  CASE-1 condición-1:

```
        oración simple o compuesta
    CASE-2 condición-2:
        oración simple o compuesta
    ....
    DEFAULT
        oración simple o compuesta
```

Ejemplo: Lenguaje natural.

Función: *determinar costo de membresía*
Entradas: *tipo de membresía*, *duración de membresía*, *edad*
Salidas: *costo de membresía*
Transformación:

```
    IF tipo de membresía es E
        IF duración de membresía = L
            costo de membresía = 100
        ELSE duración de membresía = G
            costo de membresía = 80
    ELSE tipo de membresía es P
        IF edad = M
            costo de membresía = 50
        ELSE edad = J
            IF duración de membresía = L
                costo de membresía = 150
            ELSE duración de membresía = G
                costo de membresía = 100
```

Restricciones: ninguno

Pseudo-Código

- Pseudo-Código es similar a lenguaje natural con algunas variantes y extensiones.

Secuencia

- <oración simple o compuesta>

Repetición

- REPEAT
    <oración simple o compuesta>
  UNTIL <condición>

- WHILE <condición>
    <oración simple o compuesta>
  ENDWHILE

Decisión

- IF <condición> THEN
    <oración simple o compuesta>

- IF <condición-1> THEN

```
    <oración simple o compuesta>
ELSEIF <condición-2> THEN
    <oración simple o compuesta>
...
ELSE <condición-N>
    <oración simple o compuesta>
ENDIF
```

Selección

- CASE <selector>

```
    CASE <valor-1>:
            <oración simple o compuesta>
    CASE <valor-2>:
            <oración simple o compuesta>
    ....
    DEFAULT
            <oración simple o compuesta>
```