

# Herramientas y Metodologías de Análisis y Diseño Estructurado

## Apunte de la Cátedra Metodologías de Desarrollo de Software I Claudia Marcos – Edgardo Belloni

---

Revisión Abril 1999: Carlos Rodríguez, Marcos Rossi, Maximiliano Suárez, Verónica Targiano

Revisión Marzo 2000: Maximiliano Suárez

Revisión Abril 2003: Vanesa Dell'acqua, Gastón Martini

## Introducción

El desarrollo de sistemas pequeños, en la cual participan una o dos personas, es una tarea simple. Los cambios naturales que surgen durante el ciclo de desarrollo del sistema no producen una gran propagación de cambios en el sistema. Sin embargo, si el sistema es grande y en su desarrollo participan varios grupos de personas desarrollando una tarea específica, hay que tener en cuenta no solo la comunicación con el usuario sino también la inter-relación entre los distintos grupos de trabajo.

Algunos de los problemas comunes que los desarrolladores encuentran en la construcción de software de cierta complejidad son los siguientes:

- ✓ El dominio de aplicación no es conocido.
- ✓ La comunicación con el usuario.
- ✓ La comunicación con el grupo de desarrollo.
- ✓ La carencia de buena documentación.

Por esta razón, es necesario seguir una serie de pasos sistemáticos para que los diferentes grupos de desarrollo posean una buena comunicación. Estos pasos son brindados por los modelos de ciclo de vida, los cuales están constituidos por diferentes etapas:

**Especificación de requerimientos:** Se realizan entrevistas con el usuario identificando los requerimientos y necesidades del usuario.

**Análisis:** Modela los requerimientos del usuario.

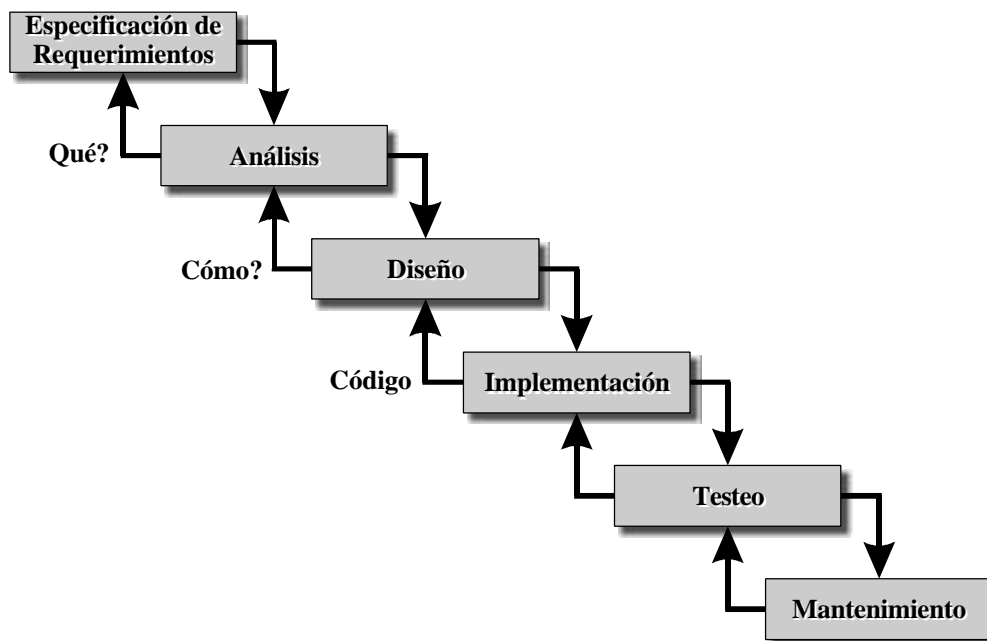
**Diseño:** Se modela la solución del sistema, teniendo en cuenta el ambiente de implementación a utilizar, por ejemplo, si el sistema es centralizado o distribuido, la base de datos a utilizar, lenguaje de programación, performance deseada, etc.

**Implementación:** Dado el lenguaje de programación elegido se implementa el sistema.

**Testeo:** En esta etapa se verifica y valida el sistema teniendo en cuenta algunos criterios determinados por el grupo correspondiente.

**Mantenimiento:** Es la etapa más difícil de desarrollo del sistema, actualiza y modifica el sistema si surgen nuevos requerimientos.

Existen varios métodos para describir el ciclo de vida de un sistema, uno de ellos es el desarrollo estructurado en cascada (Fig. 1).



**Fig. 1: Modelo de Ciclo de Vida en Cascada**

En un principio fue de gran utilidad pero el problema es que para pasar de una etapa a la otra había que terminar la primera, produciendo un gran problema si algún cambio era requerido. La etapa de Mantenimiento consumía el 80% del costo de producción.

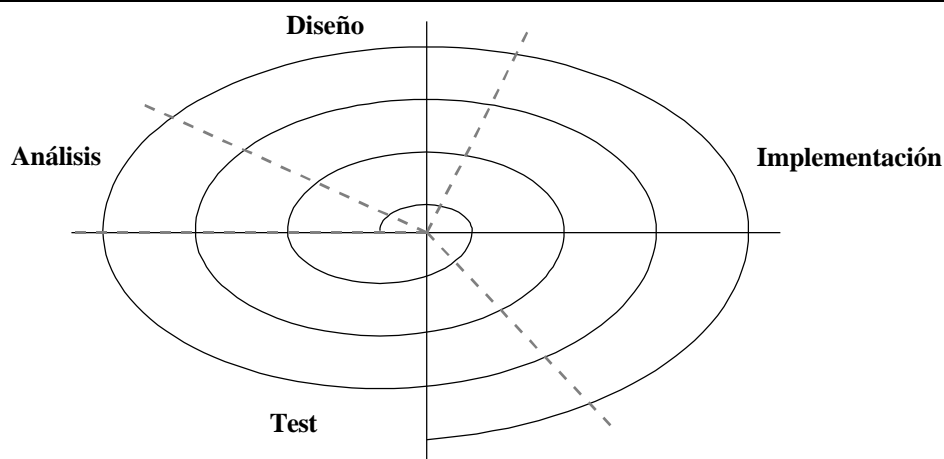
Debido a los nuevos requerimientos en el desarrollo de software, surgieron muchos otros modelos que trataban de solucionar los problemas existentes, que se basaron en el modelo en Cascada. Por ejemplo, el Modelo en Espiral, en el cual el sistema se desarrolla incrementalmente (Fig. 2).

Los modelos propuestos poseen básicamente las mismas etapas, pero varían en:

- ✓ los métodos y herramientas utilizadas en cada actividad
- ✓ los controles requeridos, paralelismo en las actividades
- ✓ las salidas de cada etapa

No es aconsejable elegir un modelo y seguirlo al detalle sino que se debe adaptar a las características del proyecto que está siendo desarrollado.

Los métodos de desarrollo de software pueden dividirse en dos grupos: *función/dato* y *orientados a objetos*.



**Fig. 2: Modelo de Ciclo de Vida en Espiral**

Orientado a Función/Dato	Orientado a Objetos
✓ Énfasis en la transformación de datos.	✓ Énfasis en la abstracción de datos.
✓ Funciones y datos tratados como entidades separadas.	✓ Funciones y datos encapsulados en entidades fuertemente relacionadas.
✓ Difícil de entender y modificar.	✓ Facilidades de mantenimiento.
✓ Funciones, usualmente, dependientes de la estructura de los datos.	✓ Mapeo directo a entidades del mundo real.

**Orientado a Función/Dato:** Aquellos métodos en los cuales las funciones y/o los datos son tratados como entidades independientes. Estos sistemas resultan difíciles de mantener. El mayor problema es que las funciones generalmente dependen de la estructura de los datos. A menudo diferentes tipos de datos tienen distintos formatos y se necesita verificar el tipo del dato (con sentencias If-Then o CASE), produciendo programas difíciles de leer y modificar. Si se desea hacer alguna modificación en la estructura de los datos se debe modificar en todos los lugares donde es utilizado.

Otro problema es que una persona no piensa naturalmente en términos de una estructura. La especificación de requerimientos se hace en lenguaje común, se especifica la funcionalidad que debe tener el sistema y no en cómo se deben estructurar los datos.

**Orientado a Objetos:** Son aquellos métodos en los cuales datos y funciones están altamente relacionados. El énfasis está centrado en la abstracción de datos. Se piensa en forma natural, los objetos son mapeados a entidades del mundo real. Los programas son fácilmente mantenibles y extensibles por medio de la construcción de subclases.

Varios métodos de desarrollo de software han sido propuestos para cada uno de estos grupo, algunos de los cuales son descriptos en la Fig. 3.

Donde:

<b>SADT:</b>	Structured Analysis and Design Technique	[Ross85]
<b>RDD:</b>	Requirement Driven Design	[Alford85]
<b>SA/SD:</b>	Structured Analysis and Structured Design	[Yourdon&Constantine79]
<b>OOSE:</b>	Object-Oriented Software Engineering	[Jacobson94]
<b>OOA:</b>	Object-Oriented Analysis	[Goldberg]
<b>OMT:</b>	Object Modeling Technique	[Rumbaugh93]
<b>UP:</b>	Unified Process	[Booch&Jacobson&Rumbaugh98]
<b>Catalysis:</b>	Catálisis	[D'Souza98]

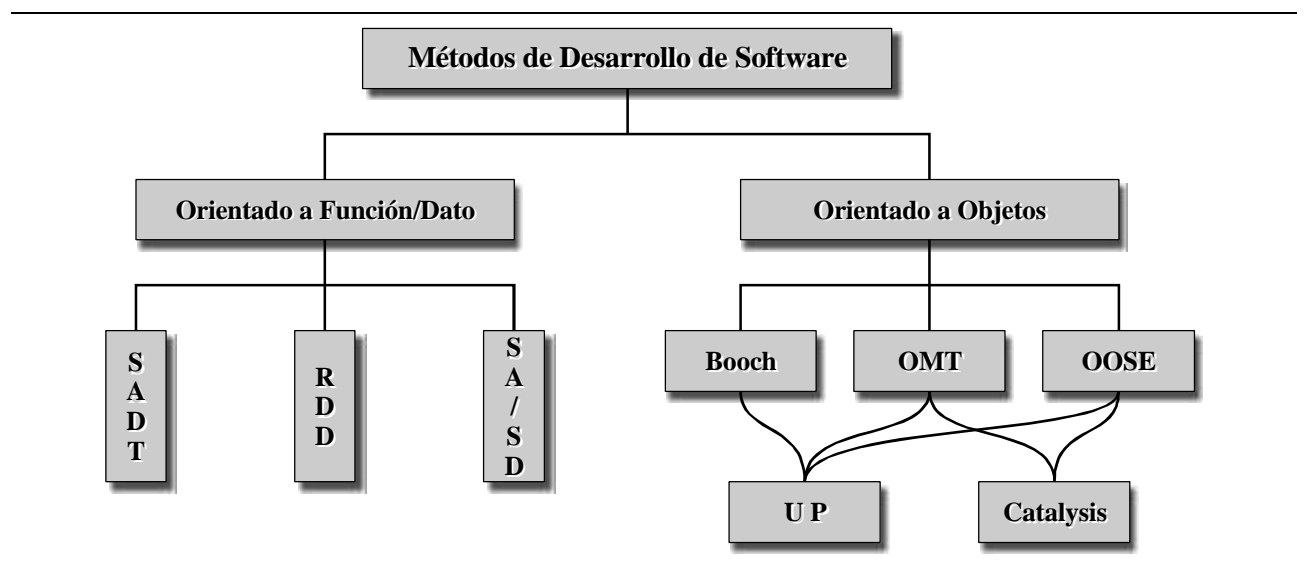
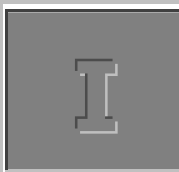


Fig. 3: Métodos de Desarrollo de Software

En el corriente curso: *Metodologías de Desarrollo de Software I* nos centraremos en los métodos de desarrollo de software orientados a función/datos y en las herramientas propuestas para el modelado de los diferentes aspectos de un sistema: datos, control y funciones. Adicionalmente, se presentaran métodos formales, específicamente Z, y métodos orientados a objetos.



# Capítulo I. ASML: A System Modeling Language

## Contenido

<b>Estructura de la Metodología .....</b>	<b>5</b>
El Modelo Esencial .....	5
El Modelo de Implementación .....	6
<b>Secuencia de Creación de los Modelos .....</b>	<b>9</b>

ASML es una metodología de desarrollo estructurado de sistemas que cubre todo el ciclo de vida de desarrollo. Esta metodología integra las principales ideas del *Análisis Estructurado* [DeMarco 79; Gane 79] y el *Diseño Estructurado* [Constant74; Yourdon 78] en un marco conceptual único y consistente.

Si bien la definición original de la metodología puede reconocerse en los libros de Ward [Ward 83; 86] y MacMenamim [MacMenam84], le cabe una mejor definición al ser interpretada como: la conjunción del *Análisis Estructurado Moderno* [Yourdon 89] y el *Diseño Estructurado* [Constant74; Yourdon 78], con extensiones para el modelado de sistemas de tiempo real [Ward 86].

## I.1. Estructura de la Metodología

ASML es una metodología que integra todas las ideas involucradas en el análisis y diseño estructurado. Conjuga las técnicas y herramientas de modelado usadas en el *Análisis Estructurado Moderno* y en el *Diseño Estructurado* dentro de una organización que destaca los diferentes modelos necesarios para: obtener una buena comprensión del problema, y diseñar una solución de buena calidad (mantenible, adaptable, etc.). Separa el modelado de un sistema en una jerarquía de modelos necesarios para comprender diferentes propiedades del mismo. Dicha jerarquía de modelos se presenta en la Fig. I-1.

El *Modelo del Sistema* está dividido en dos modelos generales:

- ✓ El *Modelo Esencial* [McMenam84; Yourdon89]: Representa la etapa de *Análisis Estructurado*. Construcción de un modelo libre de detalles tecnológicos.
- ✓ El *Modelo de Implementación* [Page88; Ward86; Yourdon78; Yourdon89]: Representa la etapa de *Diseño Estructurado*. Instanciación de un *Modelo Esencial* con una tecnología dada.

### I.1.1. El Modelo Esencial

Puede ser considerado como la aplicación de la metodología de *Análisis Estructurado Moderno* de Yourdon. La idea fundamental con la que el modelo esencial es concebido es la de *Tecnología Perfecta* en la cual no hay restricciones de cantidad de memoria, tamaño del disco o velocidad del procesador. Dos modelos componen el modelo esencial:

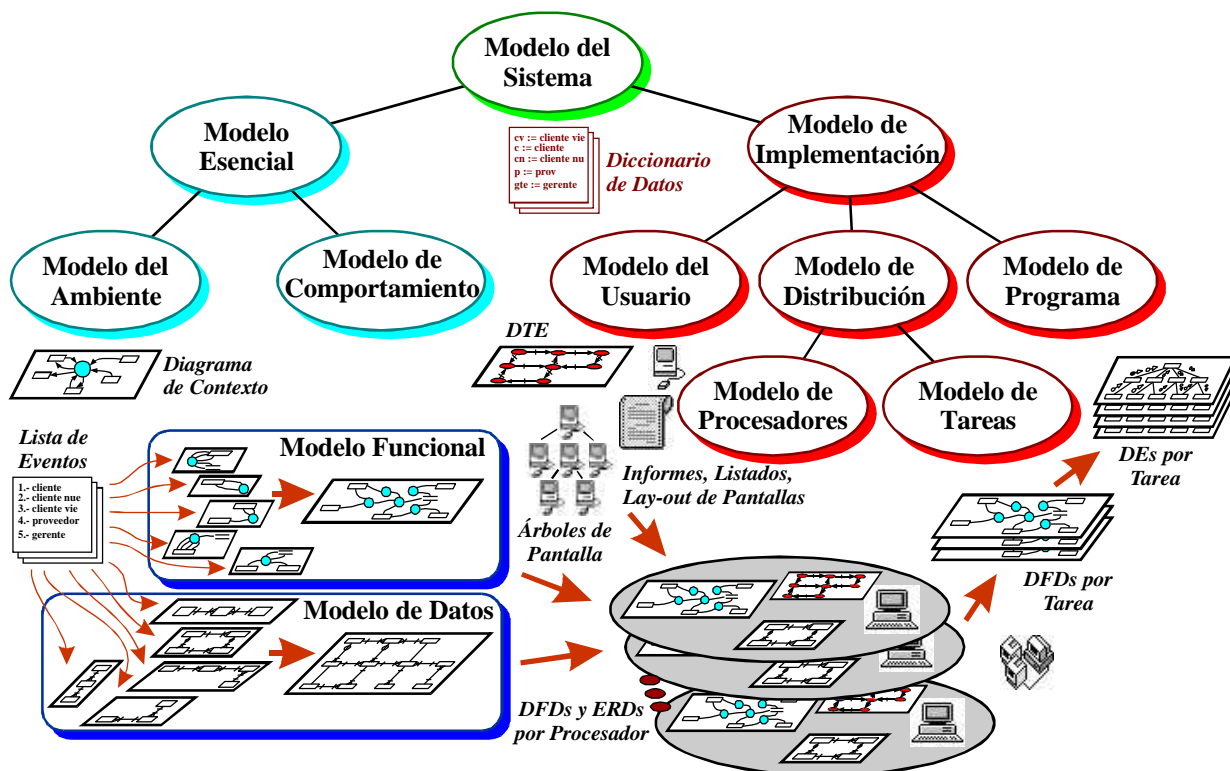


Fig. I-1: Jerarquía de Modelos

- ✓ El *Modelo del Ambiente*: Declaración de los objetivos. Creación de un *Diagrama de Contexto* y de una *Lista de Eventos*, describe los estímulos que recibe el sistema y las respuestas generadas por los estímulos. Definición del *Diccionario de Datos* inicial. Tabla de Estímulo-Respuesta.
- ✓ El *Modelo de Comportamiento*: Creación de un DFD, y un ERD por cada uno de los eventos de la *Lista de Eventos*. Los DFDs por eventos se unen en un único DFD (el *Modelo Funcional*) y los ERDs por eventos se unen en un único ERD (el *Modelo de Datos*). Se acostumbra, también, modelar el comportamiento externo del sistema con DTE, árboles de pantallas o menús, etc. La creación simultánea del modelo de datos, modelo funcional y modelo de interfaz o comportamiento externo, ayuda en la validación y completitud del modelo esencial (descubriendo, por ejemplo, eventos no considerados).

Todos los criterios de modelado y, principalmente de validación, descritos en la metodología de *Análisis Estructurado Moderno* pueden (y deben) ser aplicados en esta etapa para obtener un modelo esencial de calidad y que sea consistente.

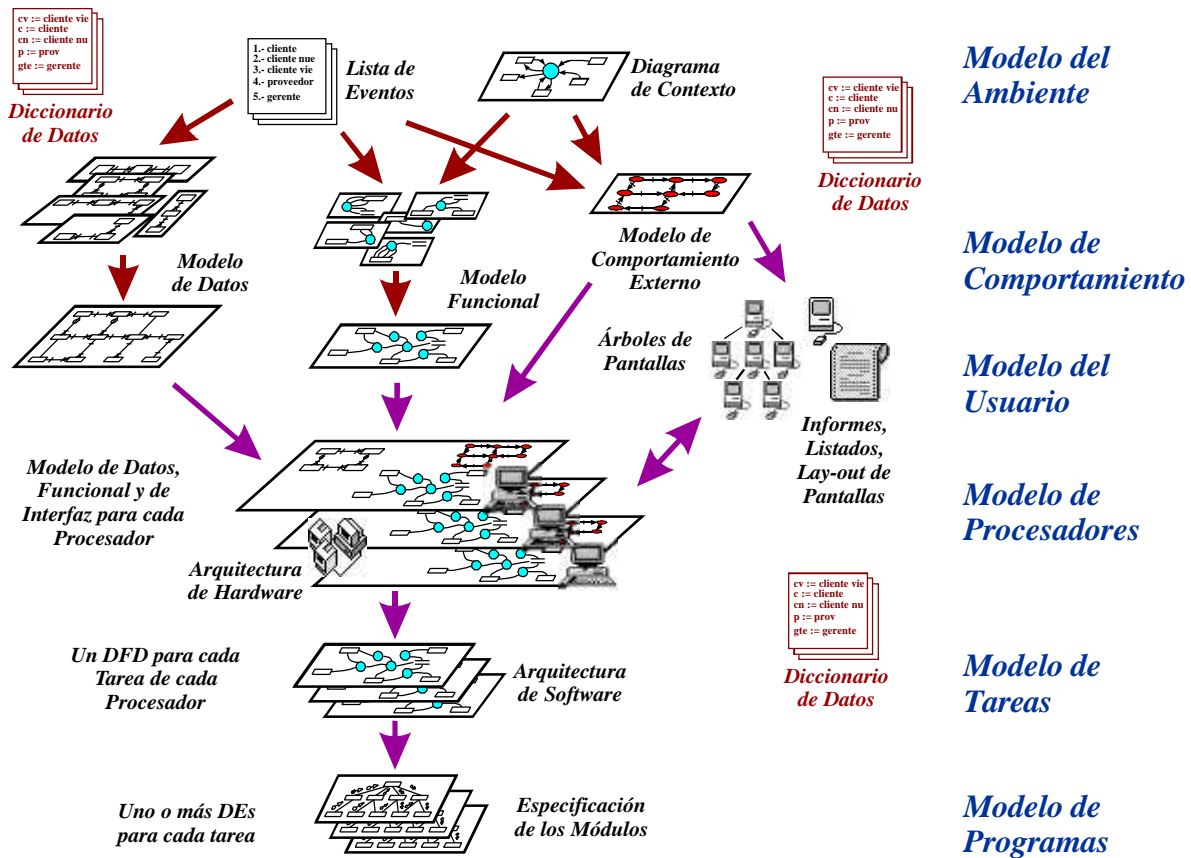
## I.1.2. El Modelo de Implementación

A partir de esta etapa, el modelo esencial es instanciado en una tecnología dada. Se debe considerar ahora, las imperfecciones de la tecnología y determinar: la cantidad de procesadores necesarios, las cualidades de estos procesadores, el tamaño de disco necesario de acuerdo al volumen de la información a ser almacenada, etc. Luego se diseña la solución sobre la base de esas restricciones tecnológicas.

La creación del modelo de implementación se fundamenta en la creación de tres modelos, uno de ellos en forma independiente (el modelo del usuario o de la interfaz hombre-máquina) y los otros dos en forma encadenada en un proceso incremental de refinamiento e incorporación de detalles:

- ✓ El *Modelo del Usuario*: La interfaz hombre-máquina es modelada en todos sus detalles, estilo (árboles de menús, lenguajes de comandos, manipulación directa, etc.), lay-out y formato de pantallas, formato de informes y listados, diseño de pantallas para el ingreso de datos y presentación de resultados, estilo de mensajes de error, secuencialidad, etc. La creación de este modelo es independiente del resto de los modelos que conforman el de implementación, y puede ser desarrollado en paralelo. Las interfaces deben ser diseñadas para cada uno de los procesadores (del modelo de procesadores) y para cada una de las tareas (del modelo de tareas).
- ✓ El *Modelo de Distribución*: Describe todas las decisiones relativas a la arquitectura de hardware (modelo de procesadores) y a la estructuración general de la arquitectura de software (modelo de tareas). Se incorporan, en los modelos creados hasta este punto algunas *Distorsiones* destinadas a optimizar el uso de esa tecnología. El criterio fundamental es: *Minimizar todo lo posible las distorsiones agregadas*.
- ✓ El *Modelo de Procesadores*: El modelo comportamental (modelo de datos, modelo funcional y modelo de comportamiento externo o de interfaz) es subdividido por procesadores. Se aplican criterios *cualitativos* (por ejemplo: necesidad de monitores de alta resolución gráfica) y *cuantitativos* (por ejemplo: velocidad del procesador, volumen de información almacenada, etc.) para seleccionar los procesadores, sistemas operativos, software y hardware de red, etc. Las *distorsiones* agregadas corresponden a la partición del DFD, ERD, DTE en procesadores, refinamiento de procesos y entidades o depósitos de datos (para asociar parte en un procesador y parte en otro) y a la incorporación de procesos para el control de la comunicación entre procesadores (siempre que la tecnología no solucione el problema de manera transparente).
- ✓ El *Modelo de Tareas*: Los modelos resultantes de la creación del modelo de procesadores son estudiados por separado (un procesador por vez), para determinar tareas diferentes (que serán programas diferentes de manera tal que se pueden ejecutar concurrentemente o no). La *distorsión* agregada en esta etapa representa la subdivisión del modelo funcional de un procesador (el DFD) en distintos DFDs (uno por tarea) agrupando procesos batch, interactivos o de tiempo real, partes del DFD aisladas del resto (comunicación solamente a través de depósitos de datos), etc. Además, es probable que sea necesario agregar procesos de control de concurrencia y sincronización para el acceso a recursos compartidos (como por ejemplo los depósitos de datos).
- ✓ El *Modelo de Programas*: La estructura del programa que implementa cada una de las tareas resultantes de las etapas de modelado de procesadores y tareas, es diseñada mediante la aplicación de las técnicas y estrategias descriptas por el *Diseño Estructurado* (por ejemplo: Análisis de Transformaciones y Transacciones) y mejorada con la aplicación de criterios de calidad (por ejemplo: Cohesión, Acoplamiento, etc.).

## I.2. Secuencia de Creación de los Modelos





## Contenido

<b>Introducción .....</b>	<b>9</b>
El Modelo Esencial .....	10
Dificultades en la Construcción del Modelo Esencial .....	10
Componentes del Modelo Esencial.....	11
¿Cuándo es Necesario Desarrollar un Modelo del Sistema Actual?.....	11
<b>El Modelo Ambiental.....</b>	<b>11</b>
La Declaración de Objetivos.....	13
El Diagrama de Contexto.....	13
Construcción del Diagrama de Contexto.....	14
La Lista de Eventos.....	16
Construcción de la Lista de Eventos .....	18
¿Qué va primero: el Diagrama de Contexto o la Lista de Eventos? .....	18
El Diccionario de Datos Inicial.....	19
<b>El Modelo Comportamental .....</b>	<b>19</b>
Creación del DFD .....	19
Creación del Modelo de Datos.....	22
Subdivisión en Niveles.....	23
El Diccionario de Datos .....	24

El análisis estructurado fue la metodología de análisis de sistemas que tuvo uno de los impactos más grandes en el ambiente de producción del software. Desde sus orígenes a mediados de los años setenta, con los libros de Tom Demarco [Demarco 79] y, Chris Gane y Trish Sarson [Gane 79], luego durante la década siguiente se extendió rápidamente hasta volverse en una metodología *standard*. En los ochenta, muchos autores trabajaron en su modernización y en la incorporación de nuevas herramientas y estrategias de modelamiento [McMenam 84; Ward 85a, 86b; y otros].

La metodología presentada en este capítulo, Análisis Estructurado Moderno [Yourdon 89], es una propuesta de Edward Yourdon que, establece el *modelo standard* para el análisis estructurado e incorpora las ideas, criterios y herramientas presentados por los libros que lo precedieron, en una estructura de trabajo muy organizada y consistente.

## II.1. Introducción

El propósito del Análisis de Sistemas es producir una declaración de requisitos esenciales del sistema que deben llevarse a cabo. Un requisito esencial es una característica que el sistema debe presentar, cualquiera sea la tecnología que llegara a ser usada para implementarlo.

Frecuentemente, los analistas de sistemas incluyen falsos requisitos en las especificaciones. Un requisito falso es una característica irrelevante o necesaria sólo para el empleo de una cierta tec-

nología. También pueden estar creándose requisitos falsos en términos de la implementación de un requisito esencial.

Al realizar un análisis de sistemas que confunde requisitos falsos por esenciales, o dejando de incluir características esenciales, debido al exceso de requisitos falsos que confundieron las consideraciones de los analistas, se pueden producir resultados desastrosos y retardar el desarrollo del proyecto considerablemente.

Según McMenamim [McMenam 84], el análisis tradicional intentó resolver este problema instruyendo a los analistas a que separasen los requisitos *lógicos* de los aspectos *físicos* del sistema pero, sus proponentes eran incapaces de definir los términos *lógicos* y *físicos* apropiadamente, y ellos no proporcionaron los procedimientos detallados para distinguir los requisitos esenciales.

Para buscar una solución es aconsejable la construcción de un *Modelo Esencial* del sistema, evitando el modelamiento del sistema actual y las características que dependen de la tecnología, desarrollando un modelo del nuevo sistema deseado por el usuario.

### II.1.1. El Modelo Esencial

El modelo esencial del sistema indica lo que el sistema debe hacer para satisfacer los requisitos del usuario y debe mencionar el mínimo posible (preferentemente nada) de *como* el sistema se llevará a cabo. Eso significa que el modelo del sistema presupone tecnología perfecta (la capacidad ilimitada de almacenamiento, velocidad infinita del procesador, etc.) y que eso pueda ser obtenida a costo cero.

De una manera específica, cuando el analista de sistemas deba discutir con los clientes y usuarios sobre los requisitos del sistema, debe evitar consideraciones sobre las implementaciones específicas de los procesos del sistema.

Por supuesto, si el objetivo final es la implementación de un grupo de programas que ejecutan y atiendan los propósitos y la conducta requerida, deben ser incluidas las propias características de una tecnología en particular, pero eso se pospone para una fase posterior: la fase de diseño. Si las características de una tecnología son consideradas como requisitos en el análisis, el modelo resultante estará viciado con esa tecnología y la complejidad (medida en la cantidad de características que el analista tiene que considerar en un momento dado) puede ser tan grande que es posible olvidarse de algunos de los requisitos esenciales (el árbol esconde el bosque). Sin embargo, una vez desarrollado el modelo esencial, éste puede ser instanciado con cualquier tecnología.

### II.1.2. Dificultades en la Construcción del Modelo Esencial

A pesar que las pautas podrían parecer simples y obvias, muchas veces se torna bastante difícil eliminar completamente del modelo esencial todos los detalles de implementación. Algunos ejemplos comunes de detalles de implementación pueden ser:

- ✓ *Las secuencias arbitrarias de actividades en un proceso de transformación de datos:* La funcionalidad del sistema se planea en base a las transformaciones de los datos. Así, la secuencia de actividades debe ser aquella exigida por los datos (una actividad **B** puede exigir un elemento de los producidos por la actividad **A** y no podrá comenzar su trabajo hasta que la actividad **A** lo haya producido).
- ✓ *Archivos innecesarios:* (Depósitos de datos que no serían necesarios si estuviera disponible la tecnología perfecta). Los archivos temporales o intermedios pueden aparecer en el modelo de implementación porque los procesos son escalonados en el tiempo para trabajar en momentos diferentes. También se introducen en dicho modelo para el resguardo de información (backup) y recuperación, debido a que la tecnología de implementación es propensa a fallas, así como las personas que operan las computadoras son propensas a cometer errores.

- ✓ *La verificación y validación innecesaria de errores:* Las actividades de validación son necesarias en un modelo de implementación por ser necesario trabajar con procesos (humanos y programas) propensos a errores, debido a los canales ruidosos en la transferencia de datos entre tales procesos.
- ✓ *Datos redundantes o derivados:* A veces ciertos datos redundantes son incluidos en depósitos con el objetivo de mejorar la eficiencia; aunque esto es normalmente razonable, debe hacerse durante la fase de diseño y no durante el modelamiento de las funciones y datos esenciales. Por otra parte, un analista de sistemas desprevenido puede incluir elementos de datos que pueden ser derivados, o calculados a partir de los valores de otros datos.

### II.1.3. Componentes del Modelo Esencial

El modelo esencial está compuesto de dos componentes principales:

- ✓ *El Modelo Ambiental:* este modelo define la *frontera* entre el sistema y el resto del mundo (es decir, el ambiente donde el sistema reside). El *modelo ambiental* está compuesto de un *Diagrama de Contexto*, una *Lista de Eventos* y una descripción pequeña del propósito del sistema, la *Declaración de Objetivos*.
- ✓ *El Modelo Comportamental:* describe la conducta, del interior del sistema, necesaria para interactuar exitosamente con el ambiente. Esta compuesto de Diagramas de Flujo de Datos, Lista de Eventos, Diagramas de Entidades y Relaciones, Diagramas de Transición de Estados, Diccionario de Datos y Especificación de Procesos.

### II.1.4. ¿Cuándo es Necesario Desarrollar un Modelo del Sistema Actual?

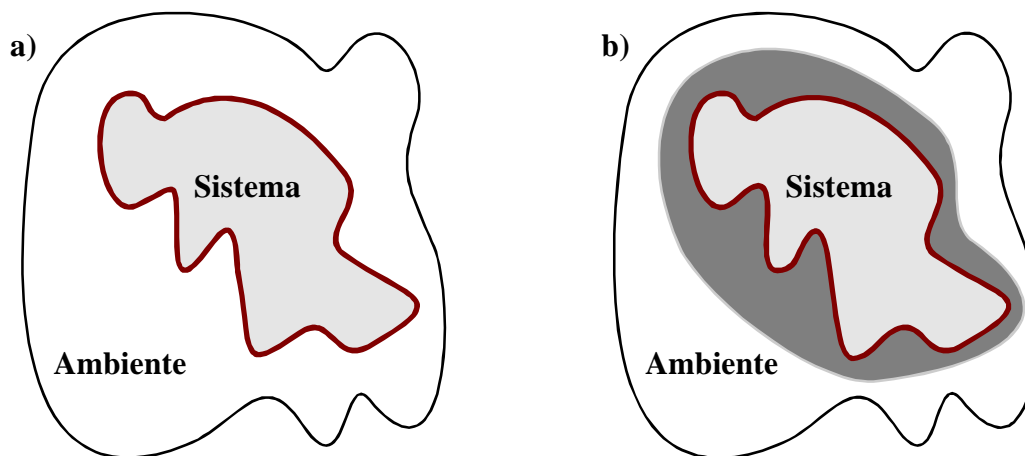
Existen circunstancias en que el analista de sistemas puede encontrar necesario elaborar un modelo del sistema actual antes de construir el Modelo Esencial del sistema. Normalmente esto es necesario debido a que el usuario (o el propio analista) no tiene suficiente conocimiento del problema actual.

Lo primero a considerar, si el modelo del sistema actual es construido, es que su objetivo principal es obtener el conocimiento y una visión general del sistema existente. El objetivo no es documentar el sistema actual detalladamente.

## II.2. El Modelo Ambiental

Para el analista de sistemas, la tarea más difícil en la especificación de un sistema es, muchas veces, la determinación de qué forma parte del sistema y qué no. Todos los sistemas, no importando cuan ambiciosos o grandiosos sean, todavía serán parte de un sistema más grande. De esta manera, el primer modelo importante por desarrollar es el que define las interfaces entre el sistema y el resto del universo, es decir, el ambiente. El Modelo Ambiental describe los límites del sistema, los estímulos que recibe y como reacciona a ellos.

Además de determinar lo que está dentro del sistema y lo que está fuera de él (definiendo la frontera entre el sistema y el ambiente), también es de fundamental importancia definir las interfaces entre el sistema y el ambiente, es decir, que información penetra en el sistema proveniente del ambiente externo, y que información del sistema es transmitida al ambiente externo.



**Fig. II-1: Frontera entre el ambiente externo y el sistema.**

La parte a) muestra los límites del sistema, la porción a ser estudiada. La parte b) muestra el área gris entre el sistema y el ambiente.

Naturalmente, no se producen entradas y salidas por casualidad: ningún sistema de información incluye a todos los datos disponibles del universo, ni cualquier sistema real emite información de forma aleatoria para el consumo del ambiente externo. Los sistemas que construimos son racionales y objetivos; específicamente, producen salidas como respuesta a un evento, o a un estímulo del ambiente. Así, otro aspecto básico del modelo ambiental es la identificación de los eventos que ocurren en el ambiente a los que el sistema debe reaccionar (los eventos que ocurren en el exterior y que exigen una respuesta del sistema).

Como se presenta en la Fig. II-1-a, la frontera entre un sistema y su ambiente es arbitraria. Puede ser establecida por una decisión de la gerencia, como resultado de una negociación política, o simplemente por accidente. Es algo en que el analista de sistemas, habitualmente, tiene alguna oportunidad de influenciar.

En general, el cliente tiene una idea buena de los límites generales pero, como mostró en la Fig. II-1-b, existe muchas veces un *Área Nebulosa*, abierta para las negociaciones, en el que el cliente no está muy seguro, en que todavía no pensó, tiene algunas ideas que necesitan ser analizadas, o todo esto junto.

Si el analista de sistemas escoge, en ese área gris, un ámbito demasiado pequeño para un proyecto, este se destina al fracaso, porque el cliente puede haber estado identificando el *síntoma* del problema en lugar de la causa. Si el analista de sistemas, escoge un ámbito demasiado grande para un proyecto, estará destinado a fallar por exceso de confianza o ingenuidad, porque estará lidiando con una situación política mucho más compleja, y estará intentando desarrollar un sistema que será probablemente demasiado grande de ser desarrollado bajo cualquier circunstancia, o puede estar tratando con problemas que al cliente no le interesan o no pueden modificarse de ninguna manera. De esta manera, es importante dedicar el tiempo necesario y obtener del usuario bastante participación en la determinación de un límite del sistema apropiado.

Algunos de los factores más importantes en la elección del ámbito del proyecto se listan a continuación:

- ✓ *El deseo del cliente de obtener una cierta porción del mercado para un servicio, o para aumentarlo más allá de su nivel actual.* Esto podría ser hecho por la oferta de un nuevo servicio o por el aumento de la funcionalidad de un servicio existente (por ejemplo, mayor funcionalidad ofrecida por los sistemas de cajeros automáticos y sistemas bancarios en línea). O el usuario podría intentar aumentar su participación en el mercado a través de la oferta de un servicio mejor o más rápido.

- ✓ *Legislación promulgada por gobiernos.* La mayoría de esos sistemas son del tipo de sistemas generadores de informe, por ejemplo, sistemas que informan el empleo (o desempleo) de obreros con base en la edad, sexo, nacionalidad y así sucesivamente. O un nuevo sistema podría construirse para acomodar alteraciones incluidas en nuevas leyes de impuestos.
- ✓ *Deseo del cliente de minimizar los gastos operacionales de algún área de su compañía.* Eso era especialmente común en los años 60/70 y en la actualidad se cumple en muchas compañías pequeñas que están instalando su primera computadora hoy. Aun así la mayoría de las organizaciones que tienen computadoras instaladas hace 10 años ya se beneficiaron de las oportunidades evidentes de reducir el exceso de empleados.
- ✓ *El deseo del cliente de obtener alguna ventaja estratégica para una línea de productos o área de negocio en que está operando.* Esto podría hacerse organizando y manejando información sobre el mercado para que pueda producirse el género de producto de una manera más oportuna y económica. Un buen ejemplo de esto se da en las líneas aéreas donde una mejor información sobre las tendencias del mercado y las preferencias de los pasajeros puede manejar las escalas de los vuelos con mayor eficiencia y determinar precios más bajos.

El modelo ambiental está compuesto de tres partes: la *Declaración de Objetivos*, el *Diagrama de Contexto* y la *Lista de Eventos*. Cada uno de esos componentes y las herramientas que se usan para su construcción, se describirán a continuación.

## II.2.1. La Declaración de Objetivos

El primer componente del modelo ambiental es una declaración textual y concisa de los objetivos del sistema. Tal declaración se destina para la gerencia (o el cliente) y otras personas que no están involucradas directamente en el desarrollo del sistema. Un posible ejemplo de una declaración de objetivos es:

*El propósito es manipular todos los detalles de las ordenes de compra de libros, así como los remitos, facturación y recibos. La información sobre las demandas de libros también debe estar disponible para otros sistemas, como "Marketing", "Ventas" y "Contabilidad".*

La declaración de objetivos puede tener una, dos o varias frases, en un único párrafo simplemente, puesto que no esta destinada a dar una descripción detallada y abarcativa del sistema. Semejante esfuerzo sería inútil: esa tarea pertenece a los modelos ambiental y comportamental.

Como resultado, la declaración de objetivos será deliberadamente vaga respecto a muchos detalles. Una buena declaración de objetivos es la que establece el ámbito del sistema, sin margen de duda y, con la menor cantidad posible de palabras. Tiene que ser formulado de una manera abstracta y conceptual.

Muchos analistas de sistemas también consideran que la declaración de objetivos debe cuantificar los beneficios que serán obtenidos por el nuevo sistema; por ejemplo: "el propósito del sistema es reducir el tiempo necesario para procesar una demanda de tres semanas en un día". Aunque esto puede ser muy útil en pequeños proyectos localizados, no es fácil de conseguir en grandes proyectos. En estos casos es necesario un análisis Costo/Beneficio en forma particular.

## II.2.2. El Diagrama de Contexto

El diagrama de contexto realza varias características importantes del sistema:

- ✓ Las personas, organizaciones o sistemas con los que el sistema se comunica. Esos elementos son conocidos como Agentes Externos, Entidades Externas o Terminadores.

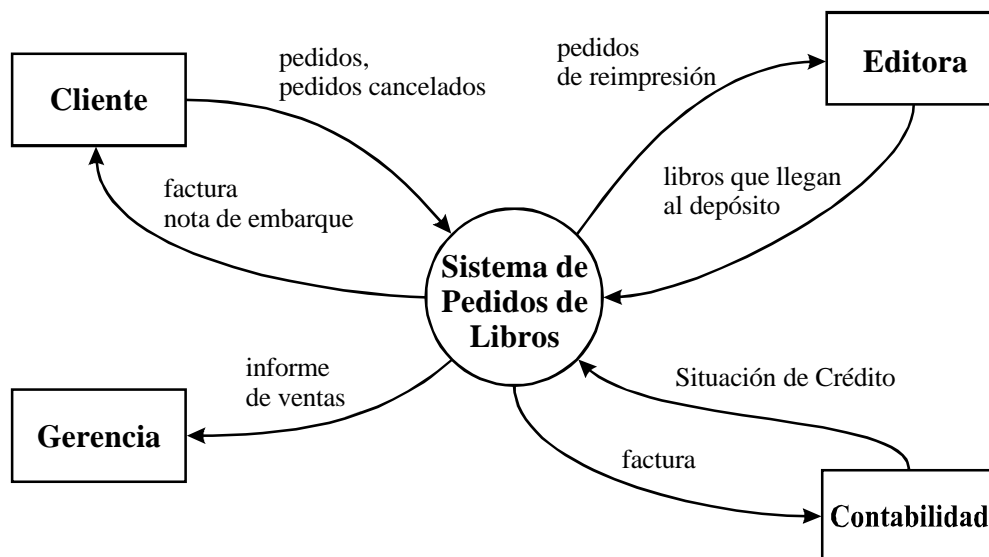


Fig. II-2: Un ejemplo de Diagrama de Contexto para el Sistema de pedidos de libros

- ✓ Los datos que el sistema recibe del mundo externo y que debe procesar de algún modo.
- ✓ La información producida por el sistema y enviada hacia el mundo externo.
- ✓ Los límites entre el sistema y el resto del mundo.

La Fig. II-2 muestra un ejemplo de Diagrama de Contexto.

### II.2.2.1. Construcción del Diagrama de Contexto

En el diagrama de contexto una única burbuja (proceso) representa el sistema. El nombre de ese proceso normalmente es el nombre del sistema o una sigla. Observe que, en el caso más extremo, el nuevo sistema puede representar una organización entera; en ese caso, en el nombre del proceso estaría, normalmente, el propio de la organización.

Los *Agentes Externos* (o *Terminadores*) son representados por un cuadro rectangular y se comunican directamente con el sistema a través de los *Flujos de Datos* o de *Control*.

Observe que los *Agentes Externos* no se comunican directamente entre sí. De hecho, los *Agentes Externos* tienen comunicaciones con el sistema, pero por definición, los *Agentes Externos*

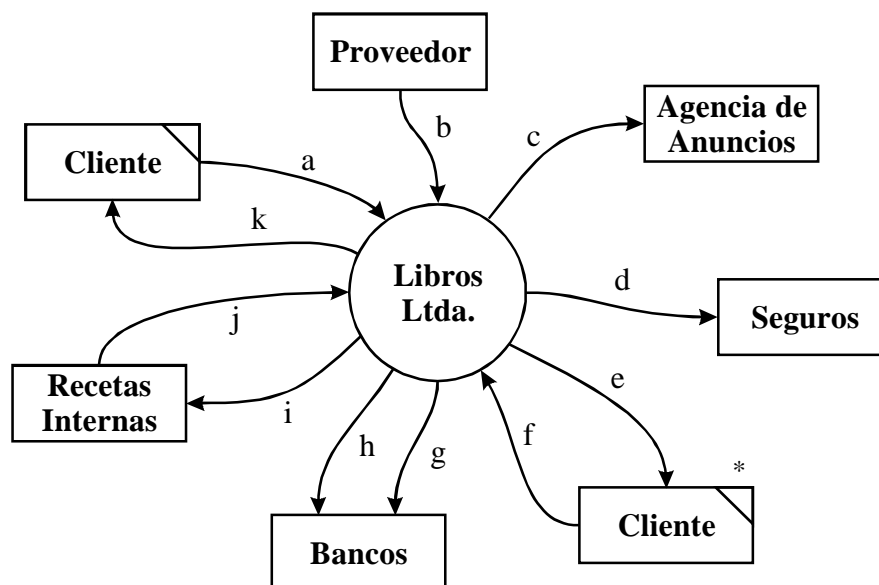


Fig. II-3: Ejemplo de Diagrama de Contexto con agentes externos duplicados

no son parte del sistema, la naturaleza y el volumen de las interacciones entre ellos son irrelevantes. Si durante las charlas con los usuarios se decidió que es esencial saber cuando, porque o como un *Agente Externo* se comunica con otro, entonces estos son parte del sistema y deben ponerse dentro del diagrama de contexto que representa el sistema completo.

Deben hacerse otras observaciones sobre los agentes externos:

- ✓ Algunos agentes externos pueden tener un gran número de entradas y salidas. Para evitar un diagrama muy congestionado, puede ser conveniente dibujarlos más de una vez. Los agentes externos duplicados están marcados para distinguirlos del otro. En la Fig. II-3, en la parte superior se presenta el agente con una línea diagonal.
- ✓ En el modelado de los roles llevados a cabo por los agentes externos se debe tener cuidado de no confundírseles con los agentes físicos que los llevan a cabo. Por ejemplo, la misma compañía puede llevar a cabo el rol del banco y agencia de seguros, pero tendrán flujos de información diferente y como otras compañías pueden especializarse en un solo rol, es conveniente especializar los roles en propios agentes.
- ✓ Los agentes externos pueden ser organizaciones, unidades o personas, que es la situación normal cuando se está modelando aplicaciones de comercio típicas. Pero los agentes también pueden estar generando y recibiendo señales de control, como cuando se modelan aplicaciones de automatización.

Como estamos principalmente interesados en el desarrollo del modelo esencial del sistema, es importante que se distinga entre *fuentes* y *manipuladores* cuando se dibuja a los agentes externos en el diagrama de contexto. Un *manipulador* es un mecanismo, dispositivo, o el medio físico para transportar datos dentro o fuera del sistema.

Cuando un *manipulador* es conocido y visible a los usuarios de la implementación actual del sistema, existe una tendencia a mostrar al manipulador, en lugar de la verdadera *fuentes* de los datos. Sin embargo, como el nuevo sistema normalmente tendrá la opción de modificar la tecnología con que los datos se traen dentro y se envían fuera del sistema, el manipulador no debe mostrarse. Es preferible hablar de "Cliente" y no de "Correo" cuando las demandas se hacen a través de una agencia de correo.

Como un compromiso, principalmente si el usuario insiste, un agente externo puede etiquetarse para mostrar al verdadero origen y al manipulador que maneja los datos (por ejemplo para etiquetar con el nombre "Cliente vía Correo").

Los *flujos* mostrados en el diagrama de contexto modelan los datos que llegan y dejan el sistema. Estos serán incluidos en el diagrama de contexto si son necesarios para determinar un evento del ambiente al que el sistema debe contestar, o si son necesarios (como datos) para que se produzca una respuesta. También pueden mostrarse los flujos de datos en el diagrama del contexto cuando los datos son producidos por el sistema para responder a un evento.

Sin embargo, habrá ocasiones en que un agente externo no iniciará entradas, porque este no conoce si el sistema necesita su entrada. Similarmente, hay ocasiones en que el sistema no comienza una salida, porque no conoce si el agente externo la necesita (o la requiere). En esos casos una solicitud es una parte esencial del modelo.

A veces es conveniente mostrar flujos de entrada y salida como un flujo de diálogo (una flecha de dos puntas). Esa concentración de dos flujos en uno puede clarificar el diagrama de contexto y describir la calidad de estar relacionados fuertemente de dos flujos.

Hay también una clase de flujos que no representan entrada o salida de datos del sistema sino que establecen la necesidad de ejecutar una función dada. Estos flujos se denominan flujos de control y son mostrados con una línea punteada.

En la Fig. II-4 se muestra parte de un diagrama de contexto, que incluye algunas de las clases de flujos mencionadas.

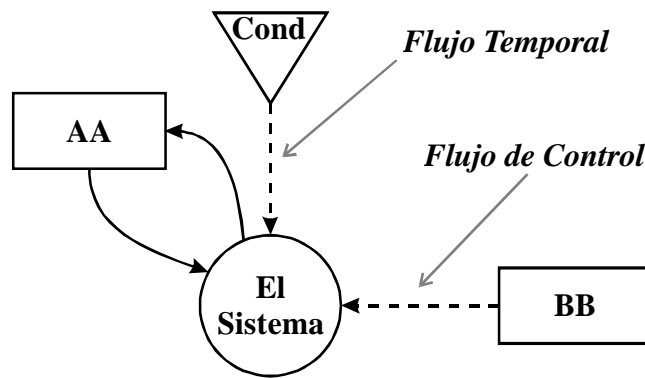


Fig. II-4: Un diagrama de contexto parcial

Una descripción detallada de los tipos de flujos en un diagrama de contexto se presentará en la sección siguiente con los tipos de eventos asociados.

### II.2.3. La Lista de Eventos

La lista de eventos es una lista narrativa de los *estímulos que ocurren* en el mundo externo, y al que el sistema debe responder. El ejemplo que sigue presenta una lista de eventos para el sistema de demandas de libros:

1. Cliente entrega demanda. (*F*)
2. Cliente cancela demanda. (*F*)
3. La dirección recibe informe de ventas. (*T*)
4. La demanda de reimpresión llega al depósito. (*C*)

Observe que cada evento se etiqueta con *F*, *T* o *C* que son para mostrar que el evento es *guiado a través de flujo* (*F*), un *evento temporal* (*T*) o un *evento de control* (*C*).

✓ Un *evento dirigido por flujo* se asocia a un flujo de datos; es decir, el sistema percibe la ocurrencia del evento cuando un grupo de datos llega (o varios grupos de datos).

Sin embargo, no todos los flujos de datos en el diagrama de contexto son necesariamente eventos dirigidos por flujos. Considere el ejemplo (parcial) de la Fig. II-5. A primera vista podría deducirse que los flujos de datos *A*, *B* y *C* son todos los indicadores de eventos separados y discretos. Sin embargo, puede ser que el flujo de datos *A* este asociado a un evento (por ejemplo, el flujo de datos es iniciado por el agente externo *X*). Para procesar el evento, puede ser que el sistema exige explícitamente a los otros agentes externos *Y* y *Z* las entradas relativas a los flujos de datos *B* y *C* para producir una respuesta.

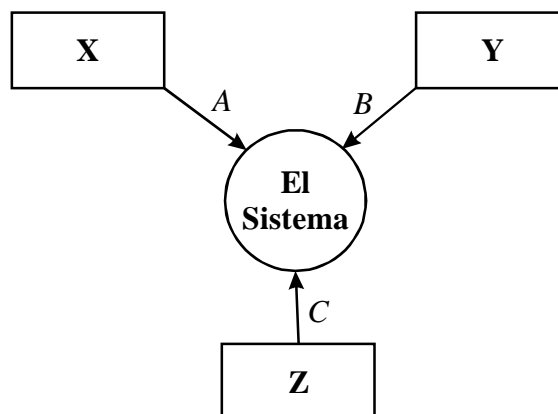


Fig. II-5



De ese modo, no existe necesariamente una correspondencia de uno a uno entre los flujos de datos del diagrama de contexto y los eventos de la lista de eventos. En general, cada flujo de datos o es un evento, o es exigido por el sistema con el propósito de procesar un evento.

✓ Los *Eventos Temporales* se descargan en un cierto momento. Estos son algunos ejemplos de eventos temporales:

- Un informe diario de todas las demandas de libros que se emite a las 9:00 horas.
- Las facturas deben generarse a las 15:00 horas.
- Deben generarse los informes para la dirección cada hora.

Observe que los eventos temporales no se descargan ni son representados por cualquier flujo de datos de entrada; se puede imaginar que el sistema tiene un reloj interno con el que puede controlar el pasaje del tiempo. Recuerde, sin embargo, que un evento temporal puede exigir que el sistema pida entradas de uno de los agentes externos. Así, pueden asociarse uno o más flujos de datos a un evento temporal, aunque los flujos de datos no representen el propio evento.

✓ Los *eventos de control* son estímulos externos que ocurren en momentos imprevistos. Un evento de control no se relaciona con el paso normal del tiempo, así, el sistema no lo puede prever por uso del reloj interno. Y al contrario de un evento orientado por flujo de datos, un evento de control no marca su presencia por la llegada de datos. Un evento de control está representado por un flujo de control (trazado) en el diagrama de contexto. Ese flujo no transporta datos, simplemente informa al sistema que debe ejecutar una acción inmediata. Estos son muy comunes en los sistemas de tiempo real [Ward 85b, 86].

Tipos de Eventos		
<b>Evento de Flujo de Datos</b>		El flujo de datos <i>flujo de inicio</i> es la entrada de datos para la transacción <i>Trans x</i> , tiene, implícitamente la función de activar la transacción, a causa de un <i>Evento de Flujo</i> (esto es modelado por el flujo de control que viene implícito con el <i>flujo de inicio</i> ).
<b>Evento Temporal</b>		La condición <i>Ct</i> es una condición temporal que activa a la transacción <i>Trans x</i> a causa de un <i>Evento Temporal</i> .
<b>Evento de Control</b>		El agente externo <i>Ag 2</i> activa a la transacción <i>Trans x</i> a causa de un <i>Evento de Control</i> .
<b>Múltiples Eventos, Simple Respuesta</b>		Hay múltiples condiciones temporales ( <i>Ct1</i> y <i>Ct2</i> ) que activan a la transacción <i>Trans x</i> . Cada una de las condiciones se corresponde con un <i>Evento Temporal</i> .
<b>Eventos de Múltiples Respuestas</b>		Hay múltiples transacciones ( <i>Trans x</i> y <i>Trans y</i> ) activadas por la condición temporal <i>Ct</i> . Dichas transacciones son activadas por un <i>Evento Temporal</i> que precisa de múltiples respuestas.

### II.2.3.1. Construcción de la Lista de Eventos

La lista de eventos es una simple lista textual de eventos del ambiente a los cuales el sistema debe responder. Al construirla asegúrese de distinguir entre un evento y un flujo relacionado a un evento. Por ejemplo, la sentencia de abajo probablemente no es un evento:

*"Un pedido de un cliente es recibido por el sistema"*

Sino que probablemente describe un flujo de datos de llegada por el cual el sistema toma conocimiento de la ocurrencia del evento. Un nombre más apropiado para el evento podría ser:

*"Cliente entrega pedido"*

Esto puede parecer un ejercicio semántico, pero no lo es. Si describiéramos un evento desde el punto de vista del sistema (esto es, visto desde adentro), podríamos identificar erróneamente los flujos de llegada que no son eventos por sí mismos, pero que son necesarios para procesar algún otro evento. De ese modo, siempre deseamos describir los eventos desde el *punto de vista del ambiente* (esto es, desde afuera del sistema).

En la mayoría de los casos, el medio más fácil para identificar los eventos relevantes de un sistema es visualizar al sistema en acción: lo que implica examinar cada agente externo y preguntar cuál es el efecto que sus acciones pueden tener en el sistema. Esto se hace habitualmente en combinación con los usuarios del sistema, que pueden desempeñar el rol de agentes externos. Sin embargo, debemos tener la precaución de distinguir entre los eventos discretos que hayan sido accidentalmente *empaquetados* juntos, como si fueran un único evento; esto sucede casi siempre con los eventos Orientados por Flujos.

Debemos examinar un evento candidato y preguntarnos si todas las instancias del evento envuelven los mismos datos; si los datos estuvieran presentes en algunas instancias y ausentes en otras, podemos tener de hecho dos eventos distintos. Por ejemplo, si examináramos cuidadosamente el evento *"cliente entrega pedido"*, veremos que algunas instancias del evento incluyen al elemento de datos *"vendedor"* y otros no; y veremos que la respuesta del sistema será diferente cuando un vendedor estuviera involucrado. De este modo sería más apropiado tener los dos eventos separados: *"cliente entrega pedido"*, y *"vendedor entrega pedido de cliente"*, lo que también muestra que posiblemente hay dos agentes externos (*vendedor* y *cliente*).

Además de esto, conviene aclarar que la lista de eventos debe incluir no solamente las interacciones normales entre el sistema y sus agentes externos, sino que también, situaciones de fallas. Como estamos creando un modelo esencial, no tenemos que preocuparnos por las fallas del sistema; pero debemos tener en consideración las posibles fallas por errores causados por los agentes externos. Como los agentes externos no son parte del sistema no es posible modificar su tecnología con el propósito de impedir errores, solo es posible responder con un mensaje e impedir que esos errores generen dificultades en el interior del sistema [Ward 85b].

### II.2.4. ¿Qué va primero: el Diagrama de Contexto o la Lista de Eventos?

Se puede comenzar con la lista de eventos o con el diagrama de contexto. Realmente, esto no importa, a medida que los componentes del modelo ambiental son generados se debe confirmar que haya consistencia entre ellos.

Que hacer primero depende en gran medida del interlocutor, con quien el analista procura conocimiento sobre el sistema. Si el interlocutor es un programador de mantenimiento de una versión actualmente en funcionamiento, es muy probable que sea más conveniente el desarrollo del diagrama de contexto. El programador de mantenimiento dará información focalizada sobre las entradas y salidas del sistema. En tanto que si el interlocutor es un usuario o algún funcionario de nivel intermedio o alto en la organización, será más fácil la construcción de la lista de eventos.

Cuando ambos componentes del modelo ambiental estuvieren terminados, estaremos en condiciones de afirmar lo siguiente:

- ✓ Cada flujo de entrada en el diagrama de contexto será necesario en el sistema para reconocer que un evento ocurre o para la generación de una respuesta a un evento, o ambos.
- ✓ Cada flujo de salida debe ser una respuesta a un evento.
- ✓ Para cada evento (no temporal ni de control) de la lista de eventos, deberá haber un flujo de datos de entrada a partir del cual el sistema pueda detectar que dicho evento ha ocurrido.
- ✓ Cada evento debe generar una salida inmediata como respuesta, o almacenar datos para ser emitidos como salida posteriormente (como respuesta o parte de una respuesta para algún otro evento), o intervenir con que el cambio de estado del sistema (lo cual será indicado en el diagrama de transición de estados que será elaborado más adelante).

## II.2.5. El Diccionario de Datos Inicial

Un sistema de mediano porte acostumbra tener algunas decenas de flujos de datos que entran y salen; un gran sistema puede tener, literalmente, centenas. Ahora, esos flujos de datos serán eventualmente definidos en gran detalle en el modelo comportamental, puede ser útil iniciar rápidamente la preparación de un diccionario de datos. Esto puede ser importante si las interfaces entre el sistema y los diversos agentes externos estuvieran sujetas a cambios y negociaciones; cuanto más rápido se comiencen a definir formalmente estas interfaces más rápido podrán ser finalizadas.

## II.3. El Modelo Comportamental

Una vez construido el Modelo Ambiental, tenemos una buena idea del ambiente en el cual el sistema debe trabajar. Conocemos los eventos que lo estimulan (*Lista de Eventos*) y las respuestas que tienen que generar el tratamiento de cada evento, como también que agentes externos están involucrados (*Diagrama de Contexto*). Al final de la etapa de construcción del modelo ambiental también se dispone de una primera versión del *Diccionario de Datos* con una descripción de cada uno de los flujos de datos del *Diagrama de Contexto*.

Con el *Modelo Comportamental* tenemos que descubrir y modelar la manera en la cual el sistema realiza los eventos, para generar las respuestas deseadas por los agentes externos y, también, los depósitos persistentes. Es decir, tenemos que modelar todo lo que acontece en el interior de la burbuja del *Diagrama de Contexto*. Para describir lo que sucede cuando un evento es recibido por el sistema se utiliza: *Diagrama de Flujo de Datos* preliminar, *Diagrama de Entidades y Relaciones* y *Diccionario de Datos* (Fig. II-6).

### II.3.1. Creación del DFD

Para la creación del DFD básicamente existen dos enfoques: *Partición en Eventos* [McMenam 84; Yourdon 89] (Fig. II-6) y *Enfoque de Análisis Estructurado Clásico* [DeMarco 79; Gane 79] (Fig. II-7).

En el enfoque de partición por eventos para la construcción del DFD preliminar (Fig. II-6), se agrega una burbuja por cada evento definido en la lista de eventos. Por cada evento, se especifican los flujos (control y datos), agentes externos y depósitos de datos. Al final, es validado en relación con el diagrama de contexto.

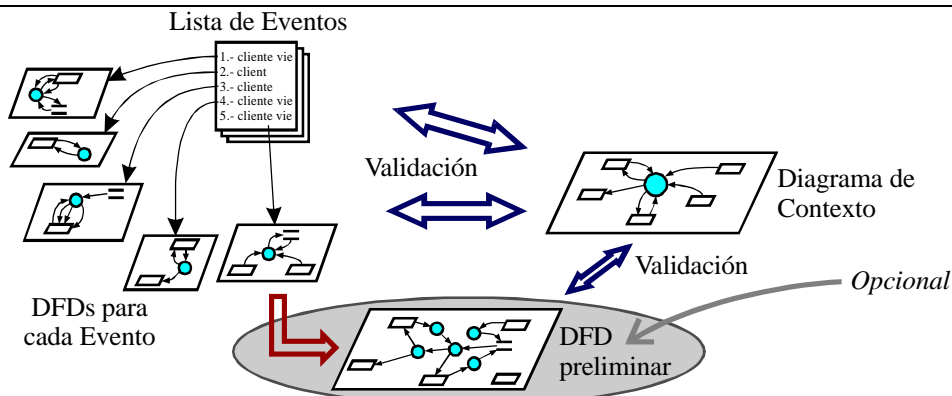


Fig. II-6: Partición en Eventos [McMenam 84; Yourdon 89]

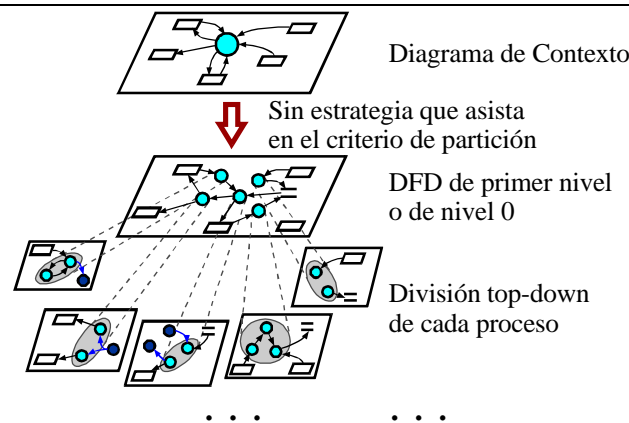


Fig. II-7: Enfoque de Análisis Estructurado Clásico [DeMarco 79; Gane 79]

Este enfoque, al contrario del enfoque clásico de análisis estructurado [DeMarco 79; Gane 79] (ilustrado en la Fig. II-7), presenta una manera sistemática para desarrollar el DFD preliminar, en base al estudio de cada uno de los eventos por separado, y fue llamado por McMenamim y Palmer de partición en eventos [McMenam 84].

En el enfoque de análisis estructurado clásico (Fig. II-7) para la construcción del DFD de primer nivel (o nivel 0), el analista (o el grupo de analistas) estudia el diagrama de contexto y crea un DFD de nivel 0 sin una estrategia que lo asista. Sobre la base de su conocimiento del sistema, o del tipo de aplicación, divide en "*Burbujas Importantes*" (por ejemplo, que representen subsistemas). Este enfoque tiene los siguientes problemas:

- ✓ *Parálisis del análisis:* En muchos sistemas grandes y complejos, simplemente no existe algo que pueda orientar al analista de sistemas en el diseño del DFD de nivel 0 adecuado a partir del diagrama de contexto. De esta forma, el se sienta en su mesa, contemplando el diagrama de contexto, esperando por la inspiración divina, o por alguien que le diga que el proyecto sobrepasó el tiempo de análisis y que ya es hora de iniciar su codificación.
- ✓ *El fenómeno de los seis analistas:* En un sistema grande y complejo muchas veces hay más de un analista de sistemas contemplando el diagrama de contexto. Para dividir el trabajo sin que uno interfiera con el trabajo de los otros, ellos crean arbitrariamente un DFD de nivel 0 con una burbuja para cada uno de ellos. De este modo, si hubiera seis analistas, el DFD de nivel 0 tendría seis burbujas.
- ✓ *La subdivisión física arbitraria:* En muchos casos un nuevo sistema se basa en otro sistema ya existente o es la informatización de una organización. Muchas veces es utilizada la subdivisión del sistema actual (por ejemplo, las áreas actuales de la organización o los sistemas de procesamiento de datos existentes) para la subdivisión del sistema nuevo, entonces se obtiene

una subdivisión desde el punto de vista funcional, lo que perpetua el sistema actual, con sus vicios y virtudes.

Por esta razón el enfoque de partición por eventos es el mas utilizado. Este enfoque no establece una actividad puramente top-down ni bottom-up. Una vez que el DFD preliminar está listo, puede ser preciso crear algunos niveles superiores (abstracción de funciones) y/o algunos niveles inferiores (descomposición de funciones).

El DFD preliminar puede ser representado como un único diseño o por un conjunto de DFDs separados (uno por cada evento). En un sistema de poca o moderada cantidad de eventos (ej.: 20 o 30 eventos), la construcción de un único DFD compuesto permite una mejor comprensión global y facilita la actividad de abstracción pero, en un sistema con muchos eventos, esa posibilidad será impracticable.

Para la construcción del diagrama de flujo de datos preliminar, con un enfoque de partición por eventos, la metodología de análisis estructurado moderna propone la siguientes cuatro etapas:

1. Se diseña una burbuja (proceso) para cada evento de la *Lista de Eventos*.
2. La burbuja recibe un nombre de acuerdo con la respuesta que el sistema debe dar al evento asociado.
3. Se diseñan los flujos de entrada y salida apropiados de modo que cada burbuja sea capaz de emitir una respuesta necesaria, y se diseñan los depósitos para la comunicación entre las burbujas.
4. El *DFD preliminar* resultante es verificado en relación con el *Diagrama de Contexto*, la *Lista de Eventos* y el *Modelo de Datos* para confirmar si esta completo y consistente.

La primera etapa es mecánica por naturaleza. Si hubiera 25 eventos en la lista de eventos, habrá 25 diagramas comportamentales (25 burbujas en el DFD preliminar). Para facilitar consultas, se le asigna un número de evento para la burbuja a la que está asociado. Así, el evento 13 corresponderá a la burbuja 13.

La segunda etapa también es mecánica, pero es necesario cierto análisis: cada burbuja recibe un nombre apropiado, en relación a la respuesta que genera. Es necesario examinar el evento y preguntar *¿Qué respuesta el sistema debe dar a este evento?*. El objetivo es procurar un nombre lo más específico posible. Por ejemplo, para el evento *Cliente Efectúa Pago*, un nombre adecuado para la burbuja asociada sería *Actualizar Cuenta* (si es esta la única respuesta exigida), en vez de *Procesar Pago de Cliente*, que no dice nada sobre la naturaleza de la respuesta.

La tercera etapa no es mecánica pero, normalmente, es bastante simple. Para cada evento se detallan sus diagramas comportamentales, es decir para cada burbuja diseñada se identifican: los flujos de entrada que la burbuja necesita para ejecutar su función, las salidas (si hubiera alguna) que la burbuja produce y los depósitos de datos a los que la burbuja debe tener acceso. Esto normalmente se hace mediante entrevistas con los usuarios apropiados y focalizando cada evento a la vez o, más precisamente, a su burbuja asociada. Entonces se debe preguntar: *¿Qué necesita esta burbuja para llevar a cabo su función?* y *¿Qué salidas genera?*.

La cuarta etapa es una actividad de verificación de consistencia:

- ✓ Se debe certificar que cada entrada en el diagrama de contexto este asociada a una entrada en uno de los diagramas comportamentales y que cada salida producida por un proceso en el DFD preliminar (una transacción) sea enviada a un depósito, o a una salida externa de las mostradas en el diagrama de contexto. Existen dos casos especiales:
  - Eventos únicos que provocan múltiples salidas: Por cada una de las respuestas, es incluida una burbuja en el DFD Preliminar.
  - Eventos múltiples que causan la misma respuesta: En este caso, una burbuja está asociada a más de un evento. Esta situación solo es válida si los flujos de datos de entrada debidos a los eventos son idénticos, y las respuestas de la burbuja también.

- ✓ Cada entidad del modelo de datos debe ser referenciada por lo menos por un proceso (transacción), para escritura y uno para lectura.
- ✓ Cada atributo de una entidad del modelo de datos debe ser componente de por lo menos un flujo de escritura y uno de lectura.
- ✓ Los flujos de escritura y lectura de un depósito de datos deben contener solamente un subconjunto de los atributos de la entidad asociada y ninguna otra cosa.

## II.3.2. Creación del Modelo de Datos

La creación del modelo de datos es una actividad que puede ser hecha en paralelo o también anteceder a la creación del DFD Preliminar, los dos modelos son independientes y ninguno de ellos puede ser considerado como predominante en la construcción del otro. Para ver esto tenemos que:

- ✓ *Son complementarios en la creación:* Uno agrega información que puede ser usada en la construcción del otro. Por ejemplo, las entidades del ERD, son depósitos en el DFD.
- ✓ *Son complementarios en la validación:* Pueden ser usados para verificaciones cruzadas.
- ✓ *Son complementarios en el modelamiento:* El diagrama de entidad-relación permite el modelamiento de los datos y las interrelaciones entre ellos (no incluidas en el DFD). Por otro lado, los DFDs permiten el modelado de funcionalidad de creación, mantenimiento, vinculación, análisis y derivación de datos. Esto es, permiten el modelamiento de las características dinámicas del sistema (no incluidas en el DER).

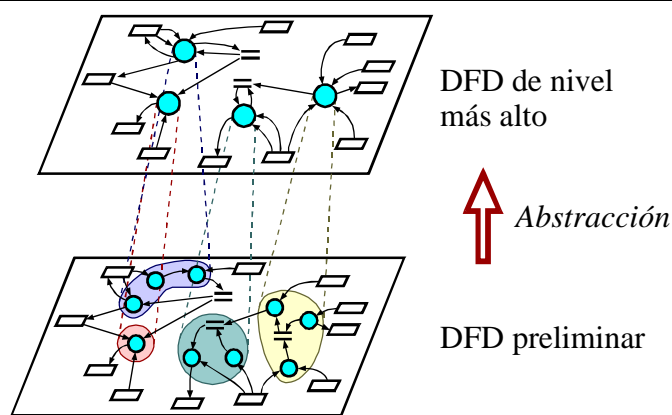
Hasta ahora, no existe un orden establecido por la metodología para la construcción del DFD y del modelo de datos, es fuertemente recomendable elaborar un DER preliminar antes de detallar los diagramas comportamentales de cada evento. Con esto se evita el riesgo de diseñar un banco de datos como colección de archivos de intercambio entre procesos.

El modelo ambiental también puede ser usado en la elaboración del DER inicial. Los *sustantivos* en la lista de eventos representan entidades en el DER; por ejemplo, en el evento *Cliente introduce pedido*, identificamos inmediatamente **Cliente** y **Pedido** como entidades candidatas. También los *verbos* son útiles pues pueden representar relaciones que se desean mantener registradas entre los sustantivos.

En el ejemplo de evento *Cliente introduce pedido* identificamos el verbo **Introduce** posiblemente como una transacción que deberá ser modelada con un diagrama comportamental, pero también como una relación que tiene que ser registrada entre las entidades **Cliente** y **Pedido**.

Por último, los *adjetivos* describen cualidades de los sustantivos y pueden representar atributos de las entidades o la necesidad de una clasificación (o especialización) de ellas. Por ejemplo, si tuviéramos el evento *Cliente joven introduce pedido*, el adjetivo **joven** puede sugerir la posibilidad de existencia de una entidad **Cliente** y otra **Cliente joven**, que tienen características semejantes a la entidad **Cliente**, pero que agregan propiedades de juventud como atributos y relaciones específicos. La edad de un cliente puede ser interpretada de maneras diferentes en sistemas diferentes o puede ser irrelevante. Si el sistema es uno de ventas en un área comercial, la edad puede ser irrelevante pero, si el área se dedica a la venta de seguros de vida, la edad es una cualidad importante para la venta. Hay que tener mucho cuidado con una clasificación de entidades basada sobre un atributo que tiene muchas probabilidades de cambiar, pues puede producir más dificultades que los beneficios que modela.

Podemos usar a lista de eventos como medio de verificación cruzada del DER inicial: todos los sustantivos de la lista de eventos deben corresponder a entidades del DER.



**Fig. II-8: Abstracción**

Los procesos del *DFD Preliminar* que pueden ser asociados a una abstracción de mayor nivel, son agregados para crear una única burbuja en el DFD de nivel superior.

### II.3.3. Subdivisión en Niveles

El DFD preliminar se compone de un solo nivel con una burbuja para cada uno de los eventos. Para un sistema mediano o grande (50 o más eventos), el DFD preliminar contiene demasiadas burbujas. Para mejorar la comprensión, ahora, precisamos subdividirlo en niveles superiores (abstracción). Esto quiere decir que deseamos agrupar los procesos (o burbujas) *relacionados* en funciones de más alto nivel de abstracción, cada uno representando una burbuja en el diagrama de más alto nivel (Fig. II-8).

Existen tres reglas que nos ayudan en el proceso de abstracción:

1. Cada agrupamiento de procesos debe involucrar respuestas estrictamente relacionadas (véase que cada burbuja en el DFD preliminar tiene un nombre relativo a la respuesta de un evento de la lista de eventos). Esto habitualmente significa que los procesos trabajan con datos estrechamente relacionados.
2. Agrupar procesos que trabajen con los mismos almacenamientos. Así, si usted encontrara un grupo de procesos con flujos para el mismo depósito, sin que otros procesos (que no son del grupo) se refieran a este depósito, entonces puede crear una burbuja, en el nivel más alto, que oculte el depósito. La figura 6 presenta un ejemplo de un grupo de procesos más a la derecha.
3. Se debe notar que la persona que examinara sus diagramas de flujo de datos, será un usuario o un analista de sistemas o un diseñador, que quiere ver todo de una sola vez. Un buen criterio de agregación o agrupamiento no tiene más de siete más o menos dos ( $7 \pm 2$ ) fragmentos de información (considerando como fragmentos de información a los *Procesos* y los *Depósitos de Datos*).

La tercera regla no es un criterio dominante. No quiere decir que tenemos que construir grupos de siete burbujas en la abstracción. Las reglas están listadas en orden de dominancia, la tercera ayuda para controlar las complejidades de los DFDs resultantes.

Aplicando estos criterios, usted tiene que generar abstracciones para la obtención de un DFD de nivel 0 de complejidad adecuada, esto es, uno de no más de  $7 \pm 2$ , fragmentos de información.

El DFD Preliminar también puede tener burbujas muy complejas, situación que será evidente a la hora de detallar su especificación. Si la especificación de una burbuja es muy larga (una página o más), la función es muy compleja y debe ser subdividida. Por ello, puede ser preciso *refinar* el DFD Preliminar creando niveles más bajos. Algunas reglas a seguir para la subdivisión en niveles de detalle son las siguientes:

- ✓ En algunos casos, el enfoque de descomposición funcional pura es adecuado. Esto es, si usted encuentra una burbuja de proceso que ejecute una función compleja, intente identificar subfunciones, cada una de la cuales pueda ser un círculo de nivel inferior. Suponga, por ejemplo, que tengamos un proceso llamado *Ajustar trayectoria de misil*; este podría ser una burbuja responsable por el mantenimiento de un evento temporal en un sistema de dirección de misiles en tiempo real. La funcionalidad general de ajustar la trayectoria del misil podría ser descompuesta en varias subfunciones:
  - Calcular la variación de la coordenada X.
  - Calcular la variación de la coordenada Y.
  - Calcular la variación de la coordenada Z.
  - Calcular un nuevo factor de *Arrastre Atmosférico*.
  - Calcular la nueva *Velocidad del Viento*.
  - ...
- ✓ En otros casos, los flujos de datos que llegan y salen de la burbuja darán una mejor indicación para la subdivisión en niveles descendentes.

Un criterio muy importante para la validación del trabajo de subdivisión (sea *abstracción* o *refinamiento*) es el *balanceo*. Es preciso verificar si las entradas y salidas a una burbuja de un determinado nivel corresponden a las entradas y salidas del diagrama de nivel inmediatamente inferior.

Para la subdivisión del DFD en niveles inferiores pueden ser aplicados los siguientes criterios:

- ✓ Si la burbuja es muy compleja, un DFD de nivel más bajo (más detallado) puede ser creado, explotando en varias subfunciones.
- ✓ Si la burbuja representa una transacción de generación de informes o unión de archivos, puede ser modelada a través de diagramas de Jackson.
- ✓ Si la burbuja es muy simple, basta una especificación del proceso usando *Diagramas de Acción*, o *Diagramas Nassi-Shneiderman*, o un *Pseudo-Lenguaje Estructurado*, o cualquier otra técnica simple de especificación de procesos.
- ✓ Si la burbuja es una transacción de manipulación de la base de datos, no muy simple, un *Diagrama de Navegación* puede ser creado.
- ✓ Si la burbuja representa un diálogo interactivo con un agente externo, un *Diagrama de Transición de Estados* puede ser creado.
- ✓ Si la burbuja involucra varias condiciones de activación de varias posibles acciones, las *Ta-blas de Decisión* pueden ser convenientes.

## II.3.4. El Diccionario de Datos

El diccionario de datos es una herramienta fundamental en el modelamiento de sistemas. Las herramientas gráficas como los diagramas de flujo de datos, los diagramas de entidad-relación, los diagramas de transición de estados, etc., son de mucha importancia para el modelamiento estructural de los sistemas (estructuras funcionales, estructuras de información, estructuras de comportamiento, etc.) y permiten una adecuada interpretación general de las ideas modeladas pero, no son completos. Para contar con una especificación completa es preciso tener una descripción textual de los detalles que no pueden ser especificados en los diagramas.

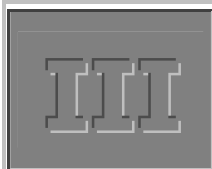
La importancia del diccionario de datos queda mucho más clara si usted trata de recordar los días de escuela primaria, en las aulas de lengua cuando usted era constantemente asediado por nuevas palabras. Recuerde también, los cursos de lengua extranjera, especialmente aquellos que exigían que leyera libros y revistas o hiciera traducciones. Sin un diccionario, usted estaría perdido. Lo mismo puede ser dicho en relación al diccionario de datos en el análisis de sistemas: sin él, usted



estará perdido, y el usuario no tendrá certeza de que usted haya entendido completamente los detalles de la aplicación.

El diccionario de datos es una lista organizada de todos los elementos de datos pertinentes al sistema (todos los nombres de las componentes de los diagramas), con definiciones precisas y rigurosas para que el usuario y el analista de sistemas puedan conocer todas las entradas, salidas, componentes de depósitos y cálculos intermedios. El diccionario de datos define los elementos de datos de la siguiente manera:

- ✓ Describiendo el significado de los flujos y depósitos mostrados en los diagramas de flujo de datos.
- ✓ Describiendo la composición de paquetes agregados de datos que se mueven por los flujos y que pueden ser divididos en ítems más elementales.
- ✓ Describiendo la composición de las estructuras de datos de los depósitos.
- ✓ Especificando los valores y unidades relevantes de partes elementales de información de los flujos de datos y depósitos de datos.
- ✓ Describiendo los detalles de las relaciones entre los depósitos de los diagramas de entidades y relaciones.



## Capítulo III. Herramientas de Análisis Estructurado

### Contenidos

<b>Diagramas de Flujo de Datos .....</b>	<b>27</b>
Flujos de Datos .....	28
Procesos.....	29
Depósitos de Datos .....	29
Agentes Externos.....	30
Refinamiento de Procesos en un DFD.....	30
Reglas de Validación.....	31
Álgebra de Descomposición de Procesos.....	32
Construcción Sistemática del DFD .....	36
Caso de Estudio – Administración Hotelera.....	36
<b>Diccionario de Datos (DD).....</b>	<b>44</b>
Notación .....	44
<b>Modelo Entidad Relación (ERD) .....</b>	<b>45</b>
Entidades y Atributos .....	46
Relaciones.....	47
Relación Uno-a-Uno.....	48
Relación Uno-a-Muchos.....	49
Relación Muchos-a-Muchos.....	50
Relaciones Indefinidas.....	50
Mecanismos de Abstracción.....	51
Clasificación.....	51
Agregación de Atributos ( <i>atributos compuestos</i> ).....	51
Especialización ( <i>es-un o es-subtipo-de</i> ) .....	51
Agregación de Entidades ( <i>compuesto-por</i> ) .....	51
Entidades Relacionantes .....	52
Construcción de un Diagrama Entidad-Relación .....	52
<b>Especificación de Procesos.....</b>	<b>53</b>
Lenguaje Estructurado.....	53
Pre/Pos Condiciones .....	54
Tabla de Decisión .....	56
Árboles de Decisión .....	56
Diagramas de Nassi-Shneiderman.....	57
<b>Diagramas de Transición de Estados (DTE).....</b>	<b>58</b>
Estados.....	58
Transiciones.....	59
Representación en Tabla de Transición.....	59
Representación en Diagramas de Grade.....	60

## III.1. Diagramas de Flujo de Datos

Los diagramas de flujo de datos (DFD) son utilizados para modelar la funcionalidad de un sistema. Tal como es descripto por DeMarco [DeMarco 79] y Gane & Sarson [Gane 79], un DFD permite representar un sistema como una red de procesos de transformación de datos que intercambian información por medio de flujos de datos.

Un proceso en un DFD puede representar funcionalidad en distintos niveles de abstracción, desde unidades funcionales de una organización (por ejemplo: departamento de recursos humanos, sección de ventas, etc.) hasta expresiones simples (por ejemplo: cálculo de la tasa nominal anual de un préstamo). La Fig. III-1 presenta un ejemplo no necesariamente completo, pero que ilustra las distintas componentes de un DFD.

El diagrama de flujo de datos describe cómo los datos fluyen a través del sistema, pero no proveen información acerca de estructuras de control o de secuencias de ejecución. La única secuencia que puede ser reconocida en un DFD es la determinada por la necesidad de información: el proceso *Generar Pedido del Cliente* puede completar su funcionalidad sólo en el caso que los flujos de datos *Datos del Cliente Validados*, *Información de Embarque* e *Información de las Tarifas* estén disponibles (fig.1). Por otra parte, los procesos *Validar Cliente* y *Validar Existencia* no tienen un orden definido de ejecución relativo entre ellos, puesto que ninguno de ellos recibe flujos de salida del otro proceso.

Podemos considerar al diagrama de flujo de datos como un lenguaje gráfico, útil para describir la funcionalidad de un sistema, en un cierto grado de detalle. La sintaxis de dicho lenguaje comprende los siguientes símbolos:

✓ *Flujos de Datos*: Información pasada de una componente a otra. Son representados por fle-

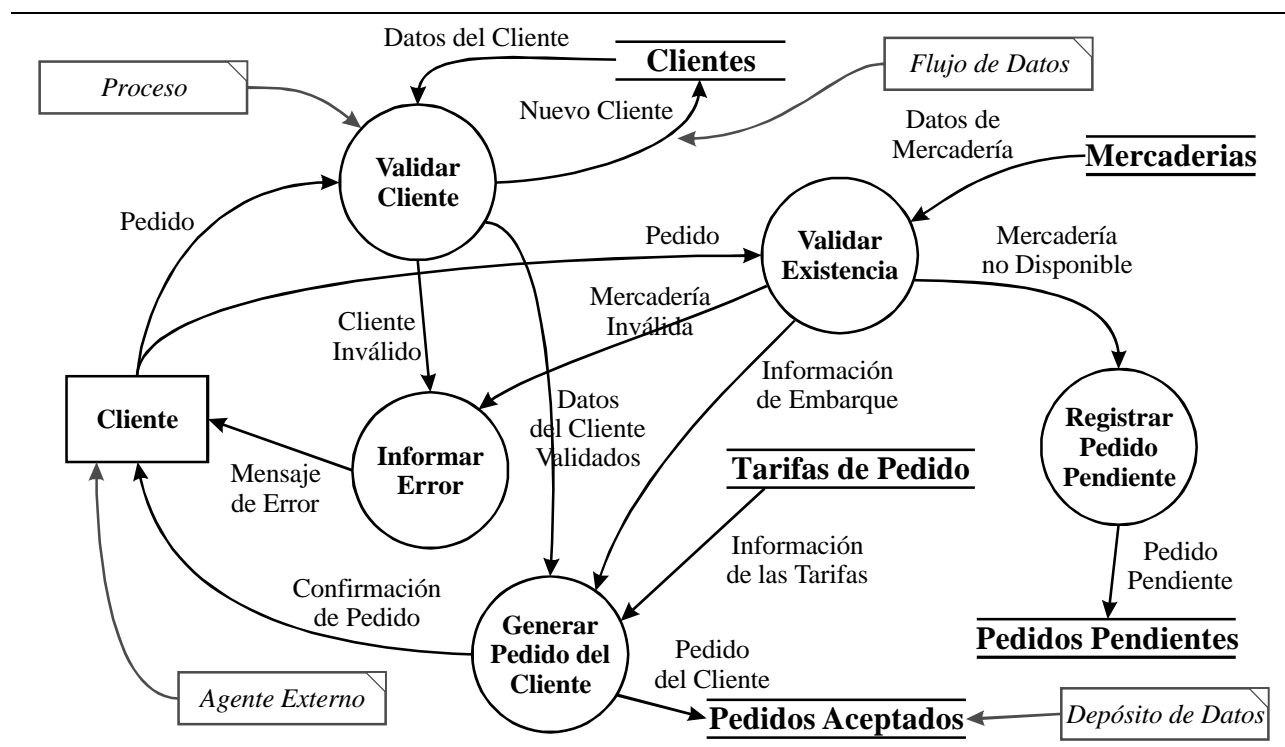


Fig. III-1: Ejemplo de DFD – “Procesar Pedido de Clientes”

Cuando un pedido es ingresado, se consultan los datos del cliente y se valida su estado de cuenta. Luego, se verifica la existencia en stock de la mercadería pedida. Si hay existencia suficiente, se registra como Pedido Aceptado y se genera una confirmación del pedido. Si no hay existencia suficiente, el pedido se registra como pendiente. Si los datos ingresados no son válidos, un mensaje de error es generado.

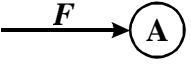
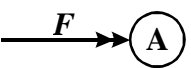
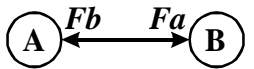


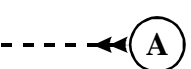


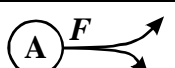
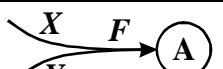
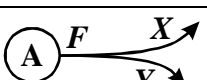
chas rotuladas.

- ✓ *Procesos*: Porciones de funcionalidad del sistema. Son representados por burbujas o círculos con un nombre descriptivo de dicha funcionalidad.
- ✓ *Depósitos de Datos*: Representan un archivo, área de memoria compartida o cualquier mecanismo de almacenamiento de datos. Son representados por dos líneas paralelas.
- ✓ *Agentes Externos*: Es una caja negra que genera flujos hacia el sistema o recibe respuestas de él. Representa alguna cosa o entidad externa que interactúa con el sistema.

### III.1.1. Flujos de Datos

Los flujos de datos son representados por arcos o flechas rotuladas. La flecha apunta en la dirección del flujo de la información, los datos fluyen en esa dirección. El nombre con el que se rotula el arco debe ser representativo de los datos contenidos en él. En algunos casos, cuando el nombre es obvio, puede ser omitido (por ejemplo: un flujo que entra o sale de un depósito de datos representando un registro completo) pero, en general, esa práctica no es recomendable.

Se han propuesto extensiones a la notación utilizada por DeMarco y Gane & Sarson [Ward 85, 86; Gane 79; Yourdon 89] algunas de ellas destinadas a hacer mas descriptivo el DFD, incorporando información adicional por medio de convenciones de diseño de los flujos. En la tabla siguiente se presenta un resumen de las notaciones mas utilizadas.

Flujos de Datos – Notaciones mas utilizadas		
<b>Flujo Discreto</b>		La información contenida en <i>F</i> solamente esta disponible para el proceso <i>A</i> , en un momento dado y con periodicidad discreta.
<b>Flujo Continuo</b>		La información contenida en <i>F</i> esta disponible, para el proceso <i>A</i> , en forma continua en un intervalo de tiempo. Es utilizado para modelar cantidades medibles (ej.: temperatura) en sistemas de tiempo real.
<b>Flujo de Diálogo</b>		El <i>flujo de datos de diálogo</i> es una simplificación, de dos flujos de datos relacionados (uno es consecuencia del otro).
<b>Flujo de Control</b>		Un flujo con línea de trazos es utilizado para modelar la ocurrencia de un evento que precisa que se ejecute una acción del sistema. Este flujo no transporta datos.
<b>Activación</b>		Un <i>flujo de control</i> que representa la necesidad de activación de un proceso. Es utilizado en protocolos de sincronización entre procesos.
<b>Suspensión</b>		Un <i>flujo de control</i> que representa la necesidad de desactivación o suspensión de un proceso. Es utilizado en protocolos de sincronización entre procesos.
<b>Flujo Temporal</b>		Un <i>flujo de control</i> que modela la ocurrencia de un evento temporal especificado por la condición temporal <i>Ct</i> (ej.: fin del dia, etc.). Típicamente se rotulan con un nombre con el prefijo: “es hora de ”
<b>Fuente Múltiple</b>		El <i>flujo de datos F</i> es provisto por una de dos fuentes. El proceso <i>A</i> precisa de los datos contenidos en el flujo pero no tiene mayor importancia la fuente.
<b>Destino Múltiple</b>		El <i>flujo de datos F</i> es generado por el proceso <i>A</i> y es enviado a dos destinatarios diferentes (simultáneamente).
<b>Conjunción</b>		El <i>flujo de datos F</i> es la unión de los flujos <i>X</i> e <i>Y</i> generados por fuentes diferentes.
<b>División</b>		Dos subconjuntos diferentes del <i>flujo de datos F</i> ( <i>X</i> e <i>Y</i> ) son enviados hacia dos destinatarios diferentes.

### III.1.2. Procesos

Un proceso representa una componente funcional del sistema. Un proceso transforma, distribuye o genera datos. Por ejemplo, los procesos pueden realizar operaciones aritméticas o lógicas sobre los datos que recibe para producir algún resultado. Los procesos en un DFD son representados por un círculo (en la notación de DeMarco) o como una caja con bordes redondeados (en la notación de Gane & Sarson) con el nombre del proceso en su interior. Deben utilizarse nombres significativos, conteniendo por lo menos un verbo, para definir la operación desempeñada. A continuación se muestran ambas notaciones:

**Notación de DeMarco**



**Notación de Gane & Sarson**



*Referencia al proceso  
comúnmente un número compuesto  
representando niveles de refinamiento*

Respecto de los procesos, un DFD describe únicamente los nombres y los flujos de entrada y salida, sin aportar ninguna otra información sobre las actividades internas de los procesos. Para describir con mayor detalle, y especificar la funcionalidad por la que es responsable el proceso, se utilizan técnicas de especificación de procesos, que serán descritas mas adelante.

Ocasionalmente, un proceso tendrá por función la de controlar la ejecución de otros procesos del DFD, en lugar de tener por funcionalidad la de transformar datos. Esos procesos son denominados *procesos de control*. Los procesos de control son representados por líneas de trazo en su contorno. La Fig. III-2 muestra un ejemplo.

### III.1.3. Depósitos de Datos

Un *depósito de datos* es incluido en un DFD para modelar la necesidad de almacenar datos. Un depósito de datos puede representar un archivo en el disco de la computadora o un área de memoria global a los procesos. En la literatura es posible encontrar que este mismo concepto puede recibir otros nombres como por ejemplo: Archivo, Almacenamiento de Datos o Repositorio.

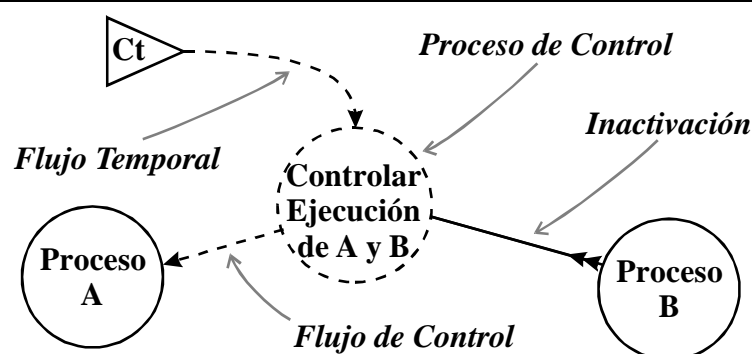
**Notación de DeMarco**



**Notación de Gane & Sarson**



Las lecturas que se realizan a un depósito de datos son representadas por flujos de salida (del depósito), y las actualizaciones de información se representan por flujos de entrada (al depósito). Comúnmente, el nombre de un depósito de datos es un sustantivo y puede estar compuesto también



**Fig. III-2**

por adjetivos. Los verbos no son válidos como parte del nombre, puesto que los almacenamientos de datos en los DFDs representan una entidad estática, carente de funcionalidad o comportamiento alguno. La estructura de datos contenida en el archivo es documentada en un *diccionario de datos*, como se mostrará más adelante.

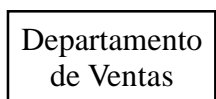
### III.1.4. Agentes Externos

Un *agente externo* establece el origen o fuente de los datos utilizados por el sistema o el receptor de los datos producidos por él.

#### Notación de DeMarco



#### Notación de Gane & Sarson



Los agentes externos (también denominados *entidades externas*) no son parte del sistema. Son modelados como *cajas negras* que generan o reciben datos del sistema. Su funcionalidad y comunicación con otros agentes externos son irrelevantes, desde el punto de vista del sistema siendo desarrollado.

Un agente externo puede representar algún área funcional de una organización, o el cargo de un funcionario, una agencia del gobierno, un dispositivo generador de datos continuos u otro sistema que debe interactuar con el sistema modelado.

### III.1.5. Refinamiento de Procesos en un DFD

Un DFD es una herramienta comúnmente utilizada para análisis *top-down*, es decir que permite realizar un análisis que va de lo general a lo particular del problema. Los DFDs son utilizados para modelar tanto vistas detalladas como de alto nivel de un sistema o programa [Yourdon 78]. La funcionalidad de un proceso puede llegar a ser tan compleja que para comprenderlo sea necesario detallar sus actividades de manera separada. Como ejemplo, consideremos un proceso representando el trabajo de toda un área funcional de una organización. En ese caso, el proceso complejo puede ser especificado con otro DFD mas detallado. Así, un árbol de DFDs puede ser desarrollado presentando diferentes niveles de abstracción en el modelado de un sistema.

La Fig. III-3 presenta la especificación de un proceso utilizando otro DFD. El proceso P resulta muy complejo y debe ser *refinado* para obtener una mejor comprensión de su funcionalidad.

El refinamiento de DFDs tiene una regla de validación cruzada para garantizar consistencia en el modelado de los procesos, en los diferentes niveles de abstracción:

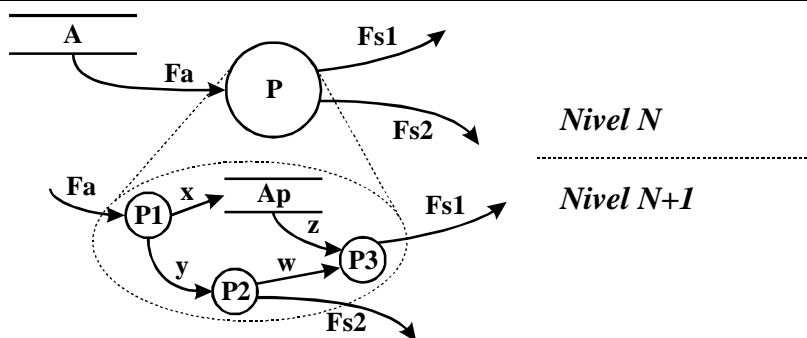
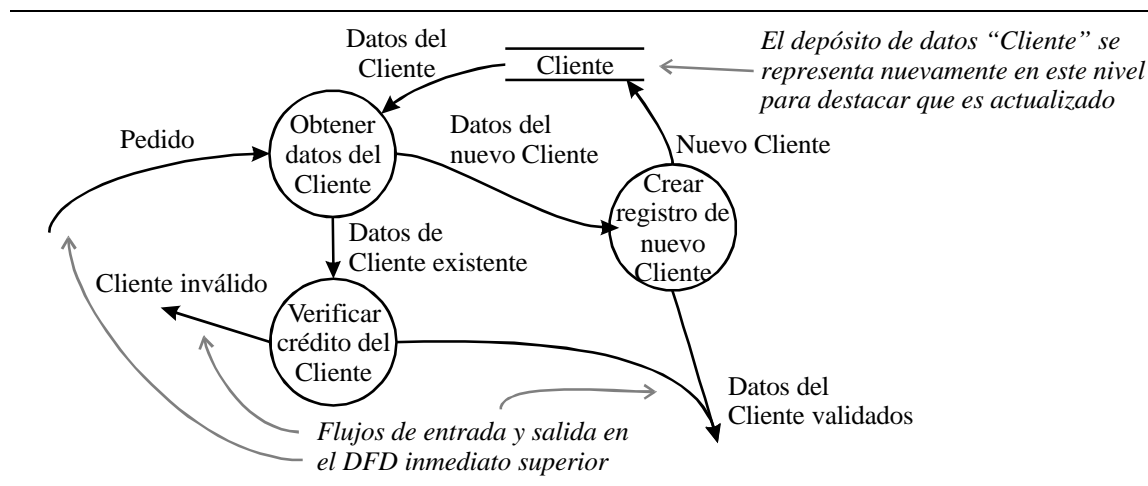


Fig. III-3: Especificación del proceso P con un DFD más detallado (Refinamiento)



**Fig. III-4: Refinamiento del proceso “Validar Cliente”**

Los flujos de entrada y salida de un proceso deben ser preservados en el refinamiento. Es decir, todos los flujos de entrada y de salida de un proceso deben ser los mismos flujos de entrada y salida del DFD de nivel inmediatamente inferior, en el árbol de niveles generado por el proceso de refinamiento.

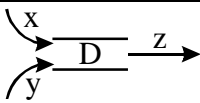
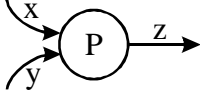
La Fig. III-4 presenta un ejemplo de refinamiento. Este ejemplo representa una vista más detallada del proceso *Validar Cliente* del DFD de *Procesar Pedido de Clientes* presentado en la Fig. III-1.

### III.1.6. Reglas de Validación

El DFD es una herramienta de modelado muy simple de utilizar. La notación incluye solamente cuatro tipos de símbolos, lo que permite una buena y rápida comprensión de los modelos. Si bien las reglas de construcción son simples y flexibles, existen algunas combinaciones de símbolos inválidas, que constituyen errores estructurales.

Errores Estructurales		
<b>Depósitos Activos</b>		Dos <i>depósitos de datos</i> no pueden estar unidos por un <i>flujo de datos</i> sin un <i>proceso</i> que oficie de intermediario.
<b>Depósitos Mágicos</b>		Debe existir al menos un <i>flujo</i> de entrada y un <i>flujo</i> de salida en todos los <i>depósitos de datos</i> .
<b>Depósitos Sumideros</b>		Los <i>depósitos</i> no pueden generar “ <i>mágicamente</i> ” ni ser “ <i>sumideros</i> ” de datos.
<b>Depósitos Externos</b>		Un <i>agente externo</i> no puede comunicarse directamente con un <i>depósito de datos</i> .
<b>Agentes Vinculados</b>		Las vinculaciones entre <i>agentes externos</i> no son relevantes para el sistema. Ellos son <i>cajas negras</i> .
<b>Procesos Mágicos</b>		En todos los <i>procesos</i> debe existir al menos un <i>flujo</i> de entrada, sin considerar flujos de lectura de depósitos de datos, y un <i>flujo</i> de salida.
<b>Procesos Sumideros</b>		Los <i>procesos</i> no pueden generar “ <i>mágicamente</i> ”, ni ser “ <i>sumideros de</i> ”, datos.

Además, deberán tenerse en cuenta reglas de balance en los flujos de entrada y de salida de procesos y depósitos, con el objetivo de mantener el modelo del sistema consistente y completo.

Balance de Entradas y Salidas		
<b>Depósitos de Datos</b>		<p>Los flujos de entrada y salida de un depósito sólo pueden contener un subconjunto de la estructura de datos almacenada en él. Así, observando el ejemplo de la izquierda, la estructura de datos de <b>d</b> debe ser tal que:</p> $((x \cup y) \subseteq d) \wedge (z \subseteq d)$ <p>En general, un principio a tener en cuenta en el diseño de la estructura de un depósito de datos y de los flujos de entrada y salida de este, es el de:</p> <div><p>“Todo lo que se ingresa en un depósito debe ser extraído en algún momento, de lo contrario no tiene sentido almacenarlo. Y todo lo que se extrae de él debe haber sido ingresado antes, o no podría encontrarlo.”</p></div>
<b>Procesos</b>		<p>Los flujos de salida de un proceso deben poder ser definidos como una función de los flujos de entrada:</p> $z = \mathbf{p}(x, y) = f(\mathbf{p}'(x), \mathbf{p}''(y), \xi)$ <p>en la especificación provista para el proceso <b>p</b>, debe resultar natural y simple de interpretar esa correspondencia (<i>f</i>), principalmente <math>\xi</math> (que puede representar, por ej. , estados locales).</p>

### III.1.7. Álgebra de Descomposición de Procesos

Los procesos del DFD pueden ser refinados en otra red de procesos conectados por flujos de datos, constituyendo un DFD de menor nivel de abstracción. Esta forma de especificar un proceso, por medio de otro DFD completo se denomina *refinamiento*, *descomposición* o *explosión*. El problema es definir cual es el criterio mas adecuado para hacer esto y, determinar hasta dónde bajar en la jerarquía de DFDs, es decir cuando parar de realizar explosiones.

Una de las formas más sistemáticas de realizar el refinamiento de un proceso es el que propone Mike Adler [Adler 88], con el Álgebra de Descomposición de Procesos. Se aplica un álgebra, definida por un conjunto de operadores, a cada uno de los procesos del DFD por separado. En los procesos que generan dudas con relación a su funcionalidad sirve como una guía para su descomposición, mientras que para los procesos que generan muy pocas dudas con relación a su funcionalidad, puede ser utilizado como un método de validación.

El álgebra es aplicada a un único proceso del DFD por vez, y precisa como entrada adicional

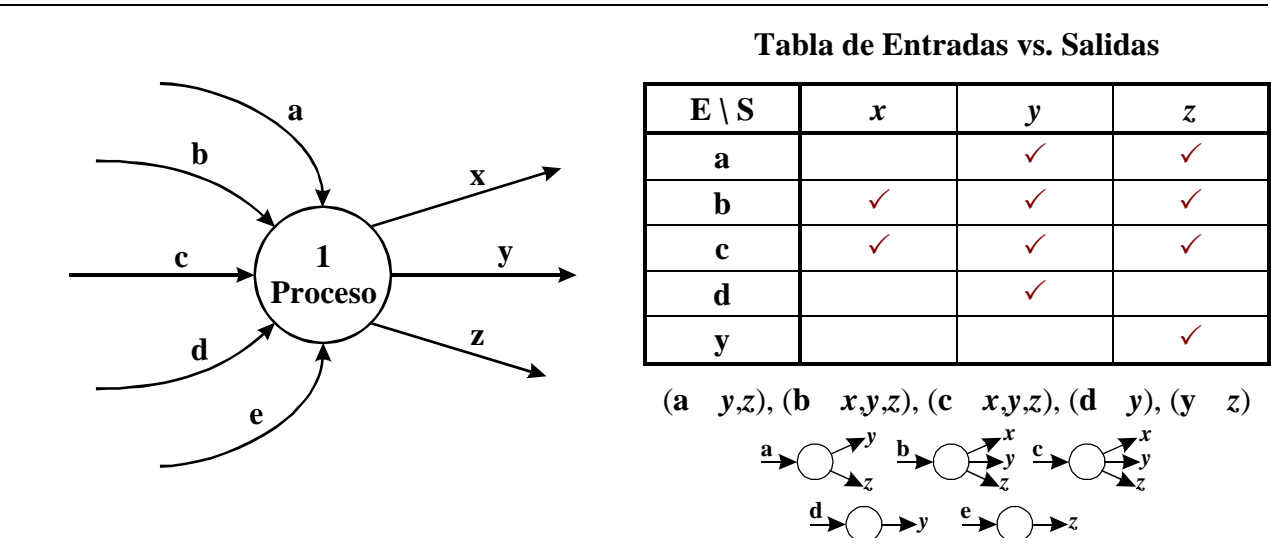


Fig. III-5: Ejemplo de argumentos del álgebra de descomposición de procesos



una tabla relacionando cada salida del proceso con las entradas usadas para generarla. Se presenta un ejemplo en la Fig. III-5.

La notación  $(\alpha \rightarrow \beta, \xi, \dots, \psi)$  será utilizada como alternativa frente a la tabla de entradas vs. salidas, donde el símbolo "  $\rightarrow$  " debe ser interpretado como "es utilizado para generar". Así, en la tabla ejemplo, la expresión  $(a \rightarrow y, z)$  es interpretada como "a es utilizado para generar y y z".

La relación "es utilizado para generar" puede ser representada gráficamente como se observa al pie de la figura , donde las flechas de entrada a una burbuja representan los flujos de entrada al proceso, y las flechas de salida de la burbuja representan los flujos generados utilizando los flujos de entrada. La burbuja representa un proceso generado (que será parte de la descomposición del proceso original).

La transformación se realiza intentando aplicar en orden cada uno de los operadores indicados a continuación.

	Oper	Antes	Después		Oper	Antes	Después
1	absorb	$(a \rightarrow z, (b \rightarrow z))$ 	$(a \rightarrow (b \rightarrow z))$ 	4	subst subset	$(a \rightarrow x, z), (b \rightarrow z)$ 	$(a \rightarrow x, (b \rightarrow z))$ 
2	collect	$(a \rightarrow (b \rightarrow z))$ 	$(a, b \rightarrow z)$ 	5	subst weak	$(a \rightarrow x, z), (b \rightarrow y, z)$ 	$(a \rightarrow x, (b \rightarrow y, z))$ 
3	subst exact	$(a \rightarrow x, z), (b \rightarrow x, z)$ 	$(a \rightarrow (b \rightarrow x, z))$ 	6	connect	$(a \rightarrow z), (b \rightarrow y)$ 	$(a \rightarrow z, (b \rightarrow y))$ 

Los operadores son aplicados para producir una expresión mínima. Cuando un término nuevo es generado, los operadores son re-aplicados a él. La expresión subrayada (burbuja llena en la representación gráfica) representa los parámetros para la operación siguiente.

El proceso de transformación del ejemplo de la Fig. III-5 es presentado a continuación.

	Oper	Expresión	Representación Gráfica
0	Expr Inicial	$(a \rightarrow y, z), (b \rightarrow x, y, z), (c \rightarrow x, y, z), (d \rightarrow y), (y \rightarrow z)$	
1	subst exact	$(a \rightarrow y, z), (b \rightarrow (c \rightarrow x, y, z)), (d \rightarrow y), (y \rightarrow z)$	
2	collect	$(a \rightarrow y, z), (b, c \rightarrow x, y, z), (d \rightarrow y), (y \rightarrow z)$	
3	subst subset	$(b, c \rightarrow x, (a \rightarrow y, z)), (d \rightarrow y), (y \rightarrow z)$	

	Oper	Expresión	Representación Gráfica
4	connect	$(b, c \rightarrow x, (d \rightarrow y), (a \rightarrow y, z)), (y \rightarrow z)$	
5	subst subset	$(b, c \rightarrow x, (a \rightarrow z, (d \rightarrow y))), (y \rightarrow z)$	
6	connect	$(b, c \rightarrow x, (y \rightarrow z), (a \rightarrow z, (d \rightarrow y)))$	
7	subst subset	$(b, c \rightarrow x, (a \rightarrow (y \rightarrow z), (d \rightarrow y)))$	

En este caso, la aplicación del álgebra de descomposición generó un DFD con cuatro procesos y tres nuevos flujos ( $l_1$ ,  $l_2$  y  $l_3$ ).

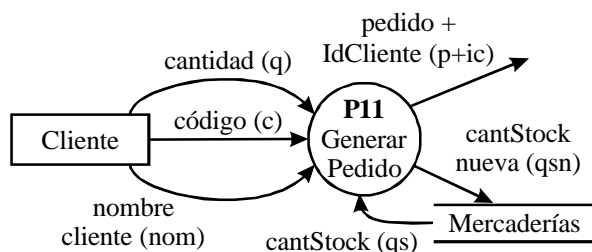
La Fig. III-6 presenta un ejemplo más concreto de la aplicación del álgebra de descomposición. El proceso *Generar Pedido* es parte de un DFD que modela la generación de facturas por la venta de mercaderías.

La tabla siguiente detalla la aplicación del álgebra en el ejemplo de la figura Fig. III-6.

	Oper	Expresión	DFD
0	Expr Inicial	$(nom \rightarrow ic), (c \rightarrow p, qsn), (q \rightarrow p, qsn), (qs \rightarrow qsn)$	
1	Subst exact	$(nom \rightarrow ic), (c \rightarrow (q \rightarrow p, qsn)), (qs \rightarrow qsn)$	

Tabla de Entradas vs. Salidas

E \ S	ic	p	qsn
nom	✓		
c		✓	✓
q		✓	✓
qs			✓



$(nom \rightarrow ic), (c \rightarrow p, qsn), (q \rightarrow p, qsn), (qs \rightarrow qsn)$

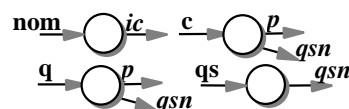


Fig. III-6: Ejemplo de Argumentos del Álgebra de Descomposición de Procesos

	Oper	Expresión	DFD
2	collect	$(nom \quad ic), (c, q \quad qsn, p), (qs \quad qsn)$	
3	subst subset	$(nom \quad ic), (c, q \quad (qs \quad qsn), p)$	

La figura Fig. III-7 presenta el resultado de la aplicación del álgebra, una descomposición del proceso de la figura. Resta solamente determinar los nombres de los tres nuevos procesos y el nombre del nuevo flujo  $l_1$  que, en este caso son bastante evidentes. El proceso P11.1 podría llevar el nombre “Armar Pedido” y el proceso P11.2 “Actualizar Stock”, mientras que el flujo  $l_1$  podría llevar el nombre “cantidad requerida”.

El proceso P11.3 debe obtener un identificador de cliente a partir de un nombre, y podría llevar por nombre “Obtener Identificador de Cliente”. Este proceso, probablemente deberá recuperar el identificador de un almacenamiento donde se codifican los clientes, por lo tanto habrá que agregar a nuestro DFD el almacenamiento y el flujo de lectura necesarios.

El proceso “Armar Pedido” es factible que deba ser descompuesto en el caso que entendamos al flujo de salida “pedido” como un flujo compuesto por un número de pedido además de la cantidad y código del artículo requeridos (esto puede determinarse con el diccionario de datos y el destino del flujo). En tal caso, dicho número de pedido debe ser generado, con seguridad teniendo en cuenta el último número de pedido existente. Esto presupone la existencia de un depósito de Pedidos, que habrá que agregar al DFD junto con el correspondiente flujo de lectura para obtener el último pedido pendiente.

Además, se observa que el pedido es generado sin tener en cuenta la cantidad en existencia del artículo requerido, esto se debe a que no consideramos al flujo “qs” como necesario para generar “p”. Esta hubiera sido una visión más realista del problema, y queda como ejercicio para el lector intentar una nueva descomposición introduciendo esta dependencia. A priori, es razonable pensar que introducir esta dependencia hará que el proceso “Armar Pedido” sea aún más factible de ser descompuesto en otros sub-procesos (digamos Validar pedido, Obtener número de pedido y Armar pedido, siguiendo el orden de suposiciones expuesto).

Por otra parte, el DFD de la Fig. III-6 preveía un flujo de salida compuesto por pedido e identificador de cliente, que puede modelarse por medio de la conjunción de los flujos “p” e “ic”.

Con frecuencia no será tan simple darles nombre a los nuevos procesos y flujos generados al aplicar el álgebra de descomposición. En ese caso, para darles un nombre adecuado es necesario analizar la narrativa que describe el problema, y aplicar sentido común para aportar claridad y evitar ambigüedad.

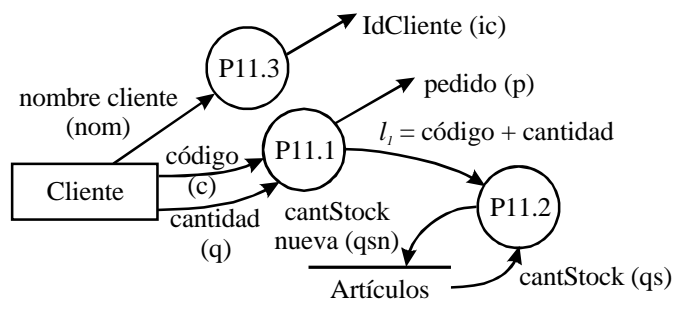


Fig. III-7: Resultado de la Aplicación del Álgebra de Descomposición

## III.1.8. Construcción Sistemática del DFD

Al desarrollar sistemas grandes y complejos, en general, no existen medios que guíen al analista de sistemas en el diseño de un DFD. El Analista se sienta en su mesa, contemplando una hoja de papel, esperando por un momento de inspiración divina o saturándose de ideas que le son imposibles de expresar todas a la vez. Esta incertidumbre puede eliminarse intentando aplicar un método de construcción sistemático, asistido por herramientas complementarias que ayudan a desglosar el problema en partes y tratarlas una por vez en una manera organizada.

A continuación se desarrollará un ejemplo concreto de construcción sistemática de DFDs. El método que aplicaremos es descripto de manera informal, a fin de presentar algunas otras herramientas que asisten en la construcción de un modelo funcional de un sistema, expresado en una jerarquía de DFDs. El método, así como los modelos, herramientas y criterios que apoyan las decisiones que involucra, será descripto en mayor detalle en el contexto de la metodología de Análisis Estructurado Moderno.

Al desarrollar un sistema, cualquiera fuere su tamaño, es necesario contar en primer término, con una narrativa textual y una declaración concisa de los objetivos del sistema (la funcionalidad que se requiere, es decir lo que se espera que el sistema haga), por supuesto validada con el usuario del sistema (las personas a las que él esta destinado).

A continuación, se presenta una narrativa textual, y su correspondiente declaración de objetivos, referente al caso de estudio que abordaremos: el desarrollo de un sistema de información para la administración de un hotel.

### III.1.8.1. Caso de Estudio – Administración Hotelera

Un hotel acepta reservas de habitaciones y exige el pago de un adelanto del 50% de la tarifa (precio de la habitación \* cant. de días). En la operación de reservas, un pasajero, consulta sobre sus necesidades de alojamiento. El recepcionista, para poder responder, arma un código según lo que el cliente demande. Con este código verifica la disponibilidad, y se la comunica al pasajero junto con el precio, o si no tiene lo requerido, pide alternativas. Al confirmar el pasajero su reserva, el empleado toma los datos personales y le da un número de reserva. Ante el pago de la reserva, se la registra junto con la fecha de pago y se envía un recibo a vuelta de correo.

Este hotel tiene un concesionario en el servicio de bar y restaurante, cuyos comensales no necesariamente están alojados. En el caso que sí lo estén, los vales firmados por los clientes son procesados por la administración del hotel, que agregará el importe de esas consumiciones a la factura que emita cuando el pasajero se retire del hotel. Ante el pago de los clientes se confeccionan y entregan recibos. Una vez por semana la administración confecciona un informe para el concesionario del bar con el detalle de las consumiciones realizadas por los clientes, acompañado por el importe correspondiente.

La gerencia, semanalmente recibe un informe de la facturación emitida. A pedido de la misma se confecciona un informe estadístico de ocupación de habitaciones.

#### III.1.8.1.1. Objetivos Funcionales

- ✓ Administración de Información sobre Reservas.
- ✓ Administración de Información sobre Pasajeros.
- ✓ Administración de tarifas y ocupación de habitaciones.
- ✓ Facturación en línea.
- ✓ Generación de Informe Semanal de Servicios.
- ✓ Generación de Informe Semanal de Facturación.
- ✓ Generación de Estadísticas de Ocupación de Habitaciones.

A partir de esta narrativa, se debe obtener una descripción de los hechos, que ocurren en el entorno o ambiente en el que el sistema funcionará, y a los que el sistema debe dar una respuesta preplaneada. Es decir, podemos ver al sistema como un agente que reacciona ante determinados estímulos que ocurren en su mundo exterior. Una vez conocido lo que estimula al sistema, nuestra tarea consistirá en planificar sus reacciones acorde con los objetivos.

Utilizaremos, para describir y enumerar los hechos o eventos que estimulan al sistema y que hacen a este reaccionar, una herramienta denominada *lista de eventos*.

#### III.1.8.1.2. Construcción de la Lista de Eventos

Para detectar los eventos se deben analizar todas las oraciones de la narrativa, analizando fundamentalmente, los diferentes sustantivos que aparecen. A partir de ellos podremos reconocer sujetos externos, es decir entidades que pueden generar estímulos al sistema, y otros objetos candidatos de los cuales el sistema mantenga información, es decir que constituirán su memoria esencial.

En la mayoría de los casos, el medio más fácil para identificar los eventos relevantes para un sistema es visualizar al sistema en acción: implica examinar cada sujeto (entidad, agente) externo y preguntar cual es el efecto que sus acciones pueden tener en el sistema.

Al extraer los eventos de la narrativa y construir la lista de eventos, es necesario tener en cuenta que un evento:

- ✓ Ocurre en el ambiente del sistema (es generado por algún sujeto externo al sistema).
- ✓ Genera una respuesta, del sistema, preplaneada.
- ✓ Ocurre en un punto del tiempo.

Los eventos detectados se redactan de la siguiente forma:

**<sujeto> <verbo> <objeto>**

Para los sujetos utilizaremos el artículo indefinido en forma singular (un, una).

#### III.1.8.1.3. Lista de Eventos construida para el Caso de Estudio

1. Un pasajero realiza un pedido de reserva
2. Un pasajero acepta la reserva
3. Un pasajero paga la reserva
4. Un pasajero cancela la reserva
5. Un pasajero se presenta para alojarse
6. Un pasajero informa que se retira
7. Un pasajero paga la factura
8. El concesionario entrega factura por consumición
9. Es hora de confeccionar el informe para el concesionario y pagar (C.t.: ha pasado una semana desde el último informe)
10. Es hora de confeccionar el informe de facturación para la Gerencia (C.t.: ha pasado una semana desde el último informe)
11. La Gerencia realiza un pedido de estadísticas de ocupación
12. La Gerencia envía nuevas tarifas para habitaciones

Además de una descripción de los estímulos a los que el sistema responde, es necesaria también, una descripción de los límites que separan al sistema de su medio ambiente. Con esta descripción tendremos una buena comprensión de los alcances que tiene el sistema. Utilizaremos para describir esto, el *Diagrama de Contexto*. Este diagrama es un caso especial del diagrama de flujo de datos, en el cual una única burbuja representa al sistema entero. El nombre que se le da a dicha burbuja (proceso) es normalmente, el nombre del sistema o una sigla patrón o modelo.

#### III.1.8.1.4. Construcción del Diagrama de Contexto

La construcción del diagrama de contexto involucra los siguientes pasos:

- ✓ Para cada sujeto de la lista de eventos se dibuja una entidad externa.
- ✓ Para cada evento, buscar un nombre para el paquete de datos que sirve de estímulo.
- ✓ Para cada evento dibujar un flujo de la entidad externa al sistema, colocándole el nombre y el número de evento que lo genera.
- ✓ Dibujar la respuesta del sistema a cada estímulo y colocarle el número de evento correspondiente. (Si la respuesta es interna, es decir no sale del sistema, no se dibuja. Las externas se dibujan todas, y pueden ser más de una por evento).
- ✓ Por último, se debe controlar la falta de estímulos necesarios, observando otras respuestas en la narrativa.

Otra herramienta utilizada para describir los estímulos y respuestas del sistema es la tabla de estímulo-respuesta, que generalmente se construye junto con el diagrama de contexto. Esta tabla asocia cada estímulo que se produce en el ambiente con las respuestas que el sistema produce, describiendo además las respuestas internas o actividades que el sistema realiza ante cada evento.

#### III.1.8.1.5. Diagrama de Contexto y Tabla de Estímulo-Respuesta para el Caso de Estudio

Hasta aquí lo que se ha logrado es comprender mejor el problema, es decir, el sistema que debemos desarrollar. Conocemos los eventos que lo estimulan (*Lista de Eventos*) y las respuestas que se generan por cada evento, como así también qué agentes externos están involucrados (*Diagrama de Contexto*, Fig. III-8). También tenemos una idea, aunque poco precisa, de las actividades a desarrollar ante cada evento (respuestas internas en Tabla Estímulo-Respuesta). Los modelos contruidos hasta aquí se denominan comúnmente, en su conjunto, *Modelo Ambiental*.

Al final de la etapa de construcción del modelo ambiental también se dispone de una primera versión del *Diccionario de Datos* (DD) conteniendo, al menos, una descripción de cada uno de los flujos de datos del diagrama de contexto. El DD será omitido por simplicidad, y a los efectos de no saturar la exposición en desarrollo. La construcción del diccionario de datos será objeto de una sección posterior.

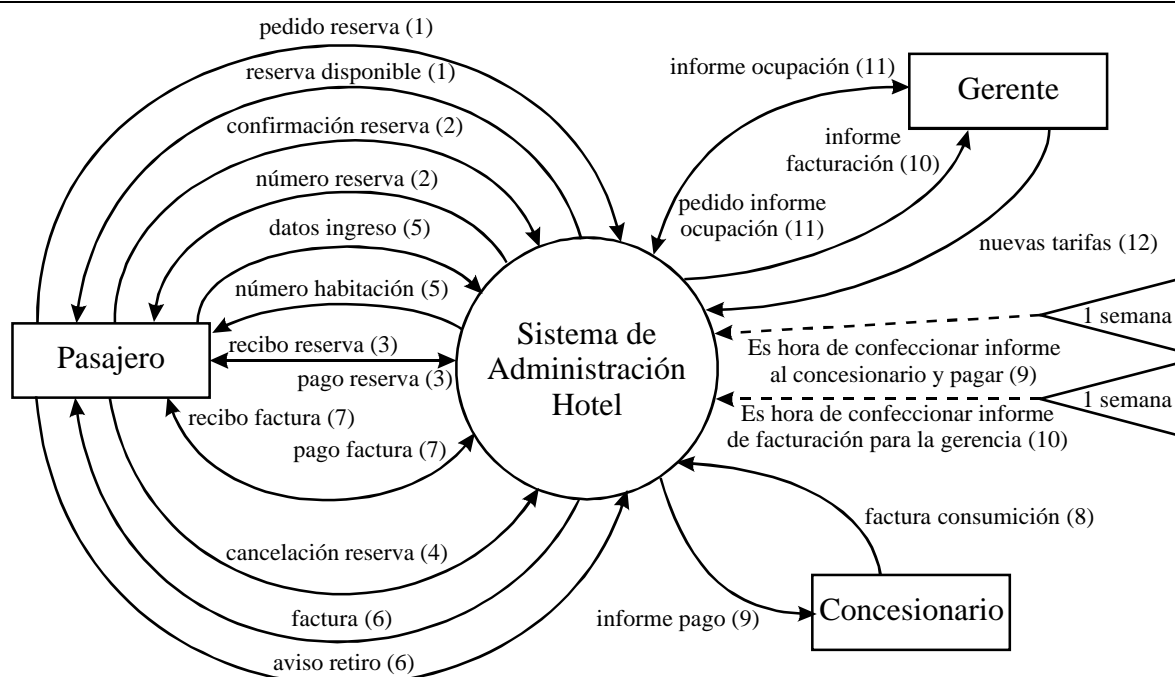


Fig. III-8: Diagrama de Contexto del caso de estudio

A partir del modelo ambiental tendremos que descubrir y modelar la manera en que el sistema trata los diferentes eventos que recibe para generar las respuestas deseadas por los agentes externos y, también, se deben descubrir y modelar los depósitos persistentes que contendrán la información esencial a ser manejada por el sistema. Esto es, tendremos que modelar todo lo que acontece en el interior del único proceso del diagrama de Contexto, que representa al sistema.

En este punto, podríamos aplicar un enfoque clásico de análisis estructurado para descomposición descendente o top-down [DeMarco 79; Gane 79]. Este enfoque propone la construcción de una jerarquía de DFDs, cada una representando un nivel de abstracción diferente. Se comienza con la construcción de un DFD de primer nivel (o nivel 0), que constituye la explosión del diagrama de contexto. Para la construcción del DFD de primer nivel, el analista (o el grupo de analistas) estudia el diagrama de contexto y crea un DFD (de nivel 0) sin una estrategia que lo asista. Utilizando su propio conocimiento del problema, o del tipo de aplicación, y su sentido común, divide al sistema en "*Burbujas Importantes*" (representando por ejemplo subsistemas). Estas burbujas importantes, o subsistemas principales, son particionadas a su vez en otras, representando un nuevo nivel de descripción acerca del detalle de las transformaciones que el sistema produce sobre los datos que recibe. Este proceso de descomposición se aplica a cada burbuja en cada nivel, describiéndola con un nuevo DFD, hasta alcanzar una burbuja que denominaremos "atómica" y que no requiere de mayor descomposición, y cuyo funcionamiento pueda ser descripto por medio de una técnica de especificación complementaria (estas se verán en detalle más adelante).

Estímulos				Respuestas	
Evento	Entidad Externa Origen	Estímulo (Flujo de datos)	Externa (Flujo de datos)	Entidad Externa Destino	Interna (Actividades o Procesos que involucra)
1	Pasajero	Pedido_Reserva	Reserva_Disponible	Pasajero	<ul style="list-style-type: none"> <li>✓ Codificar las necesidades del pasajero</li> <li>✓ Verificar disponibilidad</li> <li>✓ Informar al pasajero precio y disponibilidad</li> </ul>
2	Pasajero	Confirmación_Reserva	Número_Reserva	Pasajero	<ul style="list-style-type: none"> <li>✓ Registrar reserva</li> <li>✓ Registrar pasajero</li> </ul>
3	Pasajero	Pago_Reserva	Recibo_Reserva	Pasajero	✓ Registrar pago reserva
4	Pasajero	Cancelación_Reserva	-----	-----	✓ Registrar cancelación reserva
5	Pasajero	Datos_Ingreso	Número_Habitación	Pasajero	<ul style="list-style-type: none"> <li>✓ Completar datos pasajero</li> <li>✓ Asignarle habitación</li> <li>✓ Actualizar reservas</li> <li>✓ Abrir cuenta y factura</li> </ul>
6	Pasajero	Aviso_Retiro	Factura	Pasajero	✓ Cerrar cuenta y factura
7	Pasajero	Pago_Factura	Recibo_Factura	Pasajero	<ul style="list-style-type: none"> <li>✓ Registrar pago</li> <li>✓ Registrar consumiciones pagadas</li> </ul>
8	Concesionario	Factura_Consumición	-----	-----	✓ Registrar consumiciones para cada pasajero
9	Evento Temporal		Informe_Pago	Concesionario	<ul style="list-style-type: none"> <li>✓ Seleccionar consumiciones pagadas</li> <li>✓ Emitir informe concesionario</li> <li>✓ Pagar concesionario</li> </ul>
10	Evento Temporal		Informe_Facturación	Gerencia	✓ Confeccionar informe semanal de facturación
11	Gerencia	Pedido_Informe_Ocupación	Informe_Ocupación	Gerencia	✓ Confeccionar informe semanal de ocupación
12	Gerencia	Nuevas_Tarifas			✓ Registrar nuevas tarifas

Aunque el enfoque clásico de descomposición descendente constituye el pilar fundamental en el que se apoya el análisis estructurado, en la práctica presenta varios problemas: parálisis e incertidumbre en el análisis, partición física arbitraria del sistema, etc. Estos problemas se deben fundamentalmente a la falta de una estrategia robusta que conduzca la descomposición.

Podríamos entonces, utilizar un enfoque más sistemático para hacer frente a los problemas mencionados, intentando explotar la burbuja del diagrama de contexto utilizando el álgebra de descomposición de procesos, descrita en la sección anterior. Utilizaremos sin embargo, otro enfoque, con el objeto de presentarlo quedando su descripción detallada para la sección que cubre la metodología de Análisis Estructurado Moderno. El enfoque que utilizaremos aquí se denomina comúnmente Enfoque Medio, o como fuera llamado por McMenamim y Palmer, “de partición por eventos” [McMenam 84].

El enfoque de derivación del DFD por partición de eventos propone desarrollar un *Diagrama de Flujo de Datos Preliminar* y al nivel de detalle dado por los eventos en la lista de eventos, que describirá las transformaciones que el sistema produce sobre los datos como respuesta a los eventos. Este enfoque, suele denominarse Enfoque Medio debido a que no es una actividad puramente top-down ni tampoco es puramente bottom-up. Una vez que el DFD preliminar está listo, puede ser necesario crear algunos niveles superiores (abstracción de funciones) y/o algunos niveles inferiores (descomposición de funciones).

El DFD construido con este enfoque (DFD preliminar) presenta una burbuja por cada evento existente en la lista de eventos, y estas burbujas no se comunican directamente unas con otras, sino que la comunicación se da a través de depósitos de datos. Esto último se debe a que las burbujas o procesos del DFD preliminar representan funciones que generan las respuestas que el sistema da ante cada uno de los eventos, y los eventos que ocurren en el medio ambiente externo son, en general, asincrónicos. Es decir, no hay forma de garantizar que dos eventos ocurrirán en el mismo instante, o con segundos de diferencia, o con algún otro intervalo específico de tiempo. Los eventos ocurren cuando tienen que ocurrir, por lo tanto, como la respuesta a un evento puede requerir de datos producidos por otro proceso atendiendo otro evento, la única manera de sincronizar los múltiples procesos interdependientes del DFD preliminar es mediante depósitos de datos.

#### III.1.8.1.6. Derivación del DFD preliminar por eventos

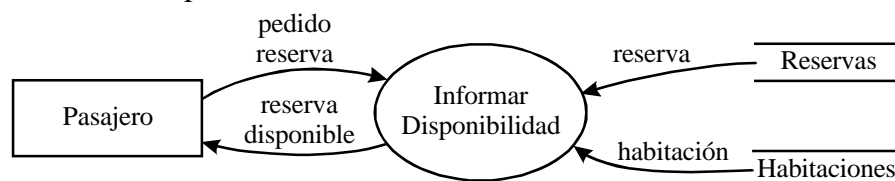
Para cada evento:

- ✓ Dibujar una burbuja que se ocupe de él.
- ✓ Ponerle un nombre acorde con la transformación que el sistema realiza con el estímulo y observando la respuesta que debe dar.
- ✓ Añadir los flujos existentes en el Diagrama de Contexto, asociados al evento.
- ✓ Contestar para cada burbuja la pregunta: ¿Qué datos necesita para producir la respuesta? y agregar los flujos que se necesiten para aportar estos datos.
- ✓ Contestar para cada burbuja la pregunta: ¿Qué otros datos produce? y agregar los flujos que se necesiten para producir y responder estos datos.

Estas últimas preguntas pueden ser contestadas apoyándose en la narrativa, y en la tabla de estímulo-respuesta.

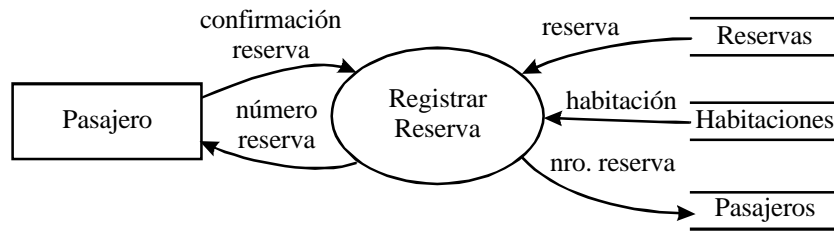
A continuación se presenta el resultado de aplicar estos pasos a los eventos en nuestro caso de estudio.

1. Un pasajero realiza un pedido de reserva

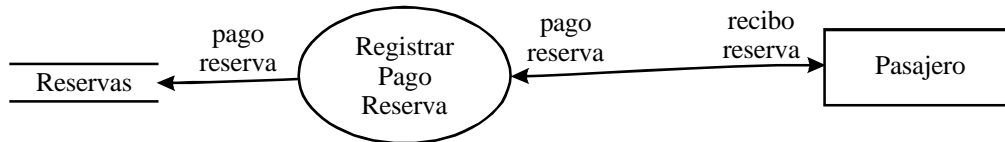




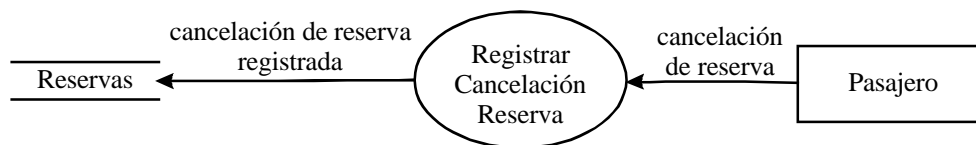
2. Un pasajero acepta la reserva



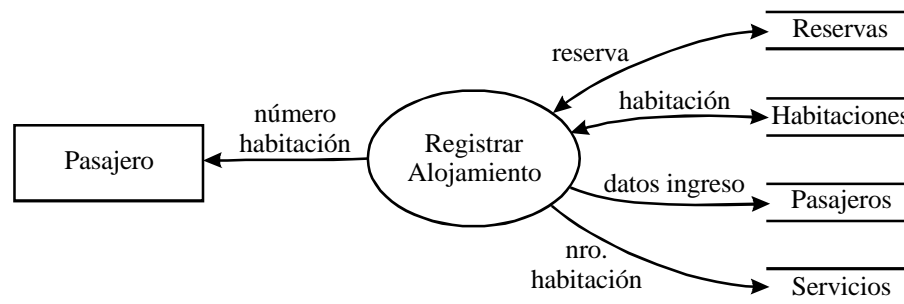
3. Un pasajero paga la reserva



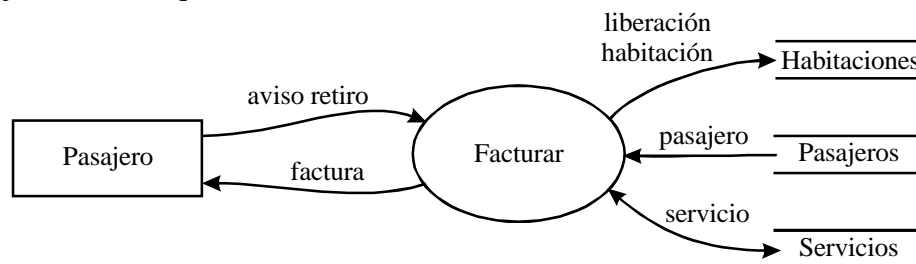
4. Un pasajero cancela la reserva



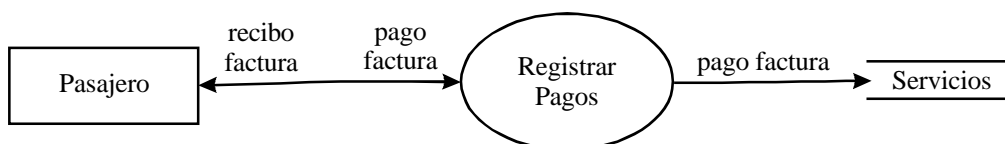
5. Un pasajero se presenta para alojarse



6. Un pasajero informa que se retira



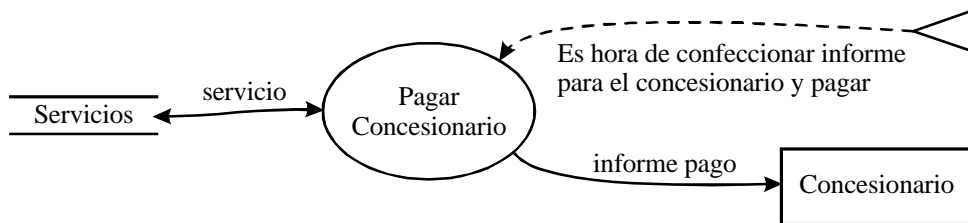
7. Un pasajero paga la factura



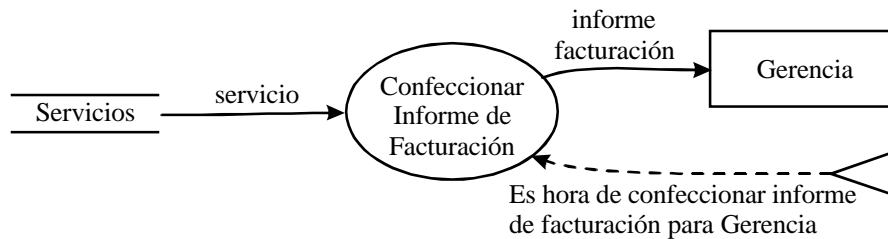
8. El concesionario entrega factura por consumición



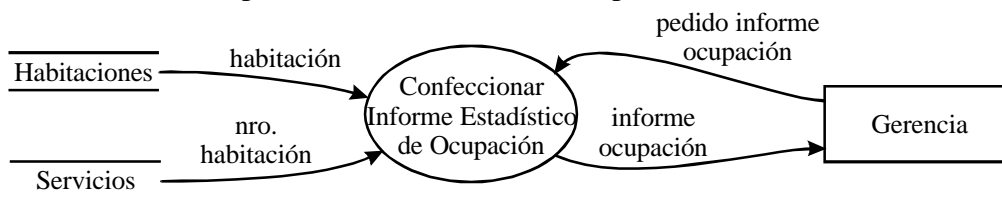
9. Es hora de confeccionar el informe para el concesionario y pagar (C.t.: ha pasado una semana desde el último informe)



10. Es hora de confeccionar el informe de facturación para la Gerencia (C.t.: ha pasado una semana desde el último informe)



11. La Gerencia realiza un pedido de estadísticas de ocupación



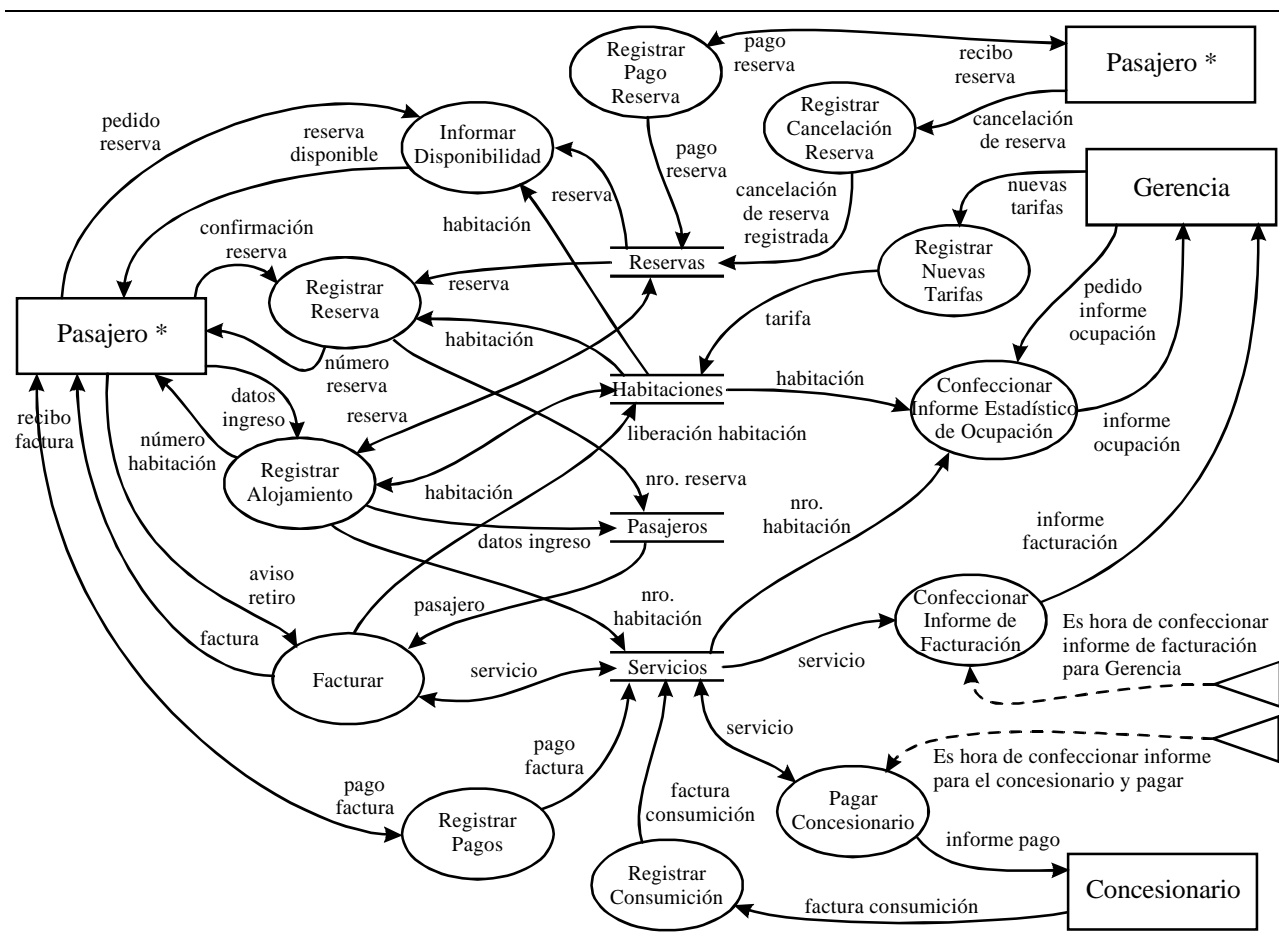
12. La Gerencia envía nuevas tarifas para habitaciones



Luego de desarrollar un diagrama por cada evento se deben conectar los diagramas en uno único, agregando los repositorios necesarios entre los datos que una burbuja produce y que otra consume. Conviene tener en cuenta en este paso, que toda información entrante a un proceso que no proviene del medio ambiente externo, debe provenir necesariamente de un almacenamiento. Por otra parte, toda información generada que no se emita directamente al medio ambiente, deberá almacenarse. Este paso puede apoyarse también en la construcción de un modelo de datos (objeto de estudio de una sección posterior) y en los objetos candidatos a memoria esencial observados en la lista de eventos, para identificar los repositorios de datos. En el caso de estudio, los repositorios identificados son: Reservas, Habitaciones, Pasajeros, y Servicios.

Por último, habrá que refinar el diagrama verificando que no tiene errores estructurales y corregir las fallas de comunicación (agregar o refinar eventos).

La Fig. III-9 muestra el DFD preliminar completo de nuestro caso de estudio.



**Fig. III-9: DFD Preliminar – Administración Hotelera**

Es necesario observar que el DFD resultante (DFD preliminar) se compone de un solo nivel con un proceso por cada uno de los eventos. Para un sistema mediano o grande (50 o más eventos), el DFD preliminar contendrá demasiados procesos y se presentará probablemente muy desnivelado, estando representados diferentes niveles de abstracción de manera simultánea. Para mejorar su comprensión, precisamos subdividirlo realizando abstracciones. Esto quiere decir que deseamos agrupar los procesos estrechamente relacionados en funciones de más alto nivel de abstracción, en un diagrama de más alto nivel. Se deben generar abstracciones para la obtención de un DFD de nivel 0 de complejidad adecuada, esto es, uno de no más de  $7 \pm 2$  procesos. Este número no es arbitrario como tampoco debe ser rígido, y representa un límite para la comprensión humana, que han encontrado expertos en ciencias de la comunicación.

Una vez obtenido el DFD de nivel 0 se procede a realizar nivelaciones descendentes o explosiones, las que se consideren adecuadas para lograr especificaciones adecuadas de las funciones del sistema. Es decir, posiblemente los procesos identificados en el DFD de nivel 0, resulten no ser procesos atómicos o primitivos y requieran particiones descendentes en DFDs de nivel inferior, esto significa que dichos procesos pueden resultar demasiado complejos para ser descriptos adecuadamente en una especificación de proceso de una página.

Queda como ejercicio para el lector realizar abstracciones y explosiones adecuadas para lograr un modelo funcional completo del caso de estudio, compuesto por una jerarquía de DFDs.

Además, sería interesante completar el sistema con la funcionalidad que considere faltante, agregando nuevos eventos, y determinar el impacto que producimos en nuestro análisis al incorporar nuevos requerimientos funcionales. Como sugerencia, mínimamente debiera atenderse lo siguiente: Todos los días acumular ocupación y servicios; y Registrar habitación fuera de servicio.

## III.2. Diccionario de Datos (DD)

El diccionario de datos es una herramienta fundamental en el modelado de sistemas. Las herramientas gráficas, como los diagramas de flujo de datos, los diagramas de estructura, los diagramas de transición de estados, etc., son de mucha importancia al modelar la estructura de los sistemas (estructuras funcionales, estructuras de módulos, estructuras de comportamiento, etc.) y permiten una interpretación general de las ideas modeladas pero, no son completos. Para contar con una especificación completa es preciso tener una descripción textual de los detalles que no pueden ser especificados en el diagrama.

El diccionario de datos es una lista organizada de todos los elementos de datos que le son pertinentes al sistema (todos los nombres de las componentes de los diagramas), con definiciones precisas y rigurosas para que el usuario y el analista de sistemas puedan conocer todas las entradas, salidas, componentes de depósitos y estructuras intermedias existentes en el sistema. El diccionario de datos describe:

- ✓ El significado de los flujos y depósitos presentes en los DFDs.
- ✓ La composición de los paquetes agregados de datos (paquetes compuestos o ítems compuestos) que son transportados por los flujos de datos y que pueden ser divididos en ítems más elementales.
- ✓ La composición de las estructuras de datos en los depósitos.
- ✓ Los valores y unidades relevantes de los ítems elementales de información de los flujos de datos y depósitos de datos.
- ✓ Los detalles de las relaciones entre los depósitos de datos.

### III.2.1. Notación

Existen muchas propuestas para la notación a ser utilizada en el diccionario de datos. La que se presenta a continuación es una de las más comunes, que utiliza un conjunto reducido y simple de símbolos:

Símbolo	Se lee	Ejemplo de la Sintaxis	Interpretación
$:=$	“Se define por” o “Se compone de”	$I := Y$	El ítem <b>I</b> está definido por la expresión <b>Y</b>
$+$	“Junto con” o “Y”	$I := A + B$	El ítem <b>I</b> está compuesto de <b>A</b> y <b>B</b> (la concatenación de <b>A</b> con <b>B</b> )
$()$	“Opcional”	$I := A + ( B )$	El ítem <b>I</b> está compuesto de <b>A</b> y <b>B</b> , o de <b>A</b> sólo ( <b>B</b> es opcional)
$\{ \}$ $i \{ \} s$	“Repeticiones de” o “Iteraciones de” o “Secuencia de”	$I := \{ A \}$ $I := I \{ A \} IO$	El ítem <b>I</b> está compuesto de una secuencia de <b>As</b> (iteración) El ítem <b>I</b> está compuesto de una secuencia de <b>As</b> (mínimo <b>I</b> y máximo <b>IO</b> ).
$[   ]$	“Uno entre” u “O”	$I := [ A   B   C ]$	El ítem <b>I</b> está compuesto de <b>A</b> o <b>B</b> o <b>C</b> . Sólo uno de ellos. (o exclusivo)
$**$	“Comentario”	$* \text{ Texto } *$	El <b>Texto</b> entre asteriscos es un comentario
$@$	“Campo Clave”	$@ A$	El elemento <b>A</b> es uno de los campos clave de un depósito de datos.

## Ejemplos

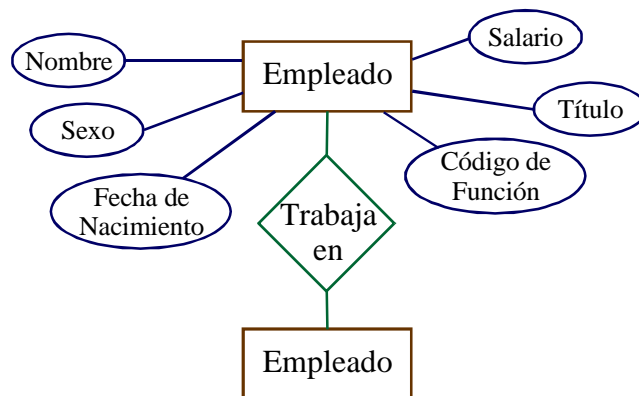
CLIENTE	$\text{:= } \{ \text{cliente} \} * \text{el archivo de Clientes} *$
cliente	$\text{:= } @ \text{nro\_cliente} + \text{nombre\_cliente} + \text{dirección\_para\_remito} + \text{crédito}$
nro_cliente	$\text{:= } * \text{identificador interno de un cliente, campo clave del depósito} \\ \text{CLIENTES} * * [ 1 \dots 999 ] * * \text{un número entre 1 y 999} *$
crédito	$\text{:= } [ \text{Positivo} \mid \text{Negativo} ]$
nombre_cliente	$\text{:= } \text{título\_de\_cortesía} + \text{primer\_nombre} + (\text{nombre-intermedio}) + \text{apellido}$
título_de_cortesía	$\text{:= } [ \text{Sr.} \mid \text{Srta.} \mid \text{Sra.} \mid \text{Dr.} \mid \text{Prof.} \mid \text{Don} \mid \text{Doña} ]$
primer_nombre	$\text{:= } 1 \{ \text{carácter\_válido} \} 30$
nombre_intermedio	$\text{:= } 1 \{ \text{carácter\_válido} \} 30$
apellido	$\text{:= } 1 \{ \text{carácter\_válido} \} 30$
carácter_válido	$\text{:= } [ \text{letra} \mid \text{dígito} \mid ' \mid - \mid ]$
dígito	$\text{:= } [ 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 ]$
letra	$\text{:= } [ \text{letra\_en\_mayúscula} \mid \text{letra\_en\_minúscula} ] * [ A \dots Z / a \dots z ] *$
dirección_para_remito	$\text{:= } \text{calle} + \text{número\_dir} + (\text{departamento}) + (\text{localidad}) * \text{si la localidad} \\ \text{no se detalla, la dirección es de Tandil} *$
calle	$\text{:= } \{ \text{carácter\_válido} \}$
número_dir	$\text{:= } \{ \text{dígito} \}$
localidad	$\text{:= } [ \text{Tandil} \mid \text{Villa Cacique} \mid \text{Barker} \mid \text{Juárez} \mid \text{Lobería} \mid \text{Posadas} ] * \\ \text{localidades en las que se entregan pedidos} *$
pedido	$\text{:= } \text{nro\_cliente} + \text{nombre\_cliente} + \text{dirección\_para\_remito} + 1 \\ \{ \text{item\_pedido} \} 10 * \text{un pedido puede contener hasta 10 items} *$
item_pedido	$\text{:= } \text{código\_artículo} + \text{nombre\_artículo} + \text{cantidad}$
código_artículo	$\text{:= } 1 \{ \text{dígito} \} 3 * \text{identificador interno de un artículo, un número de} \\ \text{hasta tres dígitos} *$

## III.3. Modelo Entidad Relación (ERD)

La construcción del modelo entidad relación (ERD) es el paso previo a la creación y uso de bases de datos en un desarrollo. El proceso de generación de la base de datos comienza desde la etapa de análisis y se va completando hasta llegar a la etapa de implementación.

El modelo entidad relación es una herramienta que permite especificar la estructura estática de la aplicación, modela dónde se encontrarán y cuál será la estructura de los datos. Los datos deben estar bien organizados ya que si datos que se refieren a algún objeto específico son almacenados en diferentes lugares la búsqueda de estos datos resulta muy difícil. Este modelo tiene los siguientes requisitos:

- ✓ *Accesibilidad:* Si los datos no son fáciles de acceder es muy difícil que sean utilizados.
- ✓ *Oportunidad:* Los datos deben reflejar un pasado relativamente inmediato. Los datos que no reflejan la situación presente con suficiente validez no tienen valor para tomar decisiones.
- ✓ *Precisión:* Cada valor almacenado debe estar dentro de un rango 'aceptable' de precisión alrededor del valor 'real'.
- ✓ *Consistencia:* Los datos deben representar fielmente la realidad.



**Fig. III-10: Ejemplo simple de diagrama de Entidad/Relación**

- ✓ **Disponibilidad:** Un dato que se necesita pero que no puede ser accedido es un síntoma de mala organización.

El modelo entidad relación permite describir la información involucrada en un sistema como un conjunto de entidades y las relaciones existentes entre ellas.

La Fig. III-10 presenta un ejemplo de diagrama entidad relación. En esta figura se pueden distinguir tres tipos de componentes diferentes:

- ✓ **Entidades:** También llamado *Tipo de Objetos* o *Clase de Objetos*. Es diseñada como una caja y representa un conjunto de objetos, llamados *instancias*, que tienen características comunes. Por ejemplo, en la figura 9 la entidad Empleado representa el conjunto de todos los empleados que trabajan en una organización.
- ✓ **Atributos:** Los óvalos vinculados a una entidad son llamados atributos. Representan características comunes a todas las instancias de una entidad.
- ✓ **Relaciones:** Son diseñadas como un rombo y representan la relación entre algunas instancias de una entidad con instancias de otra. Por ejemplo, en la figura 9 la relación *Trabaja en* indica que un empleado (instancia de una entidad *Empleado*) trabaja en un *Departamento*. La notación 1 del lado del *Departamento* y N del lado del *Empleado* indica que la relación es *uno a muchos, 1:N*, y es interpretado como: *varios empleados trabajan en un departamento, o en un departamento trabajan varios empleados*.

### III.3.1. Entidades y Atributos

Una entidad representa la información que es necesario almacenar, pudiendo esa necesidad de información abarcar personas o cosas tangibles como un empleado, un cliente o materiales. Puede ser intangible como el título de una función, una asociación, un préstamo, una compra o un pedido de seguro.

Una entidad tiene varios atributos que describen la información que se desea mantener: tamaño, valor, código, fecha de nacimiento, dirección. Generalmente, en el procesamiento de datos se almacena una colección de objetos semejantes tales como los empleados y se registra la misma información para cada uno de ellos.

Comúnmente, el programador mantiene un registro sobre cada instancia de una entidad, y un ítem de dato relacionado a cada atributo en cada uno de los registros. Los registros similares son agrupados en archivos y pueden presentarse como una tabla de dos dimensiones como la que aparece en la Fig. III-11.

En el cuadro hay un conjunto de ítems de datos y es mostrado el valor de cada uno. Cada línea contiene los valores de los atributos de una instancia en particular de la entidad. Cada columna con-

#### Estructura del Registro (Atributos de la Entidad)

Número de Empleado	Nombre	Sexo	Fecha de Nacimiento	Depto	Código de Función	Título	Salario
<b>Ocurrencia de un registro (Instancia de una entidad)</b>							
<b>Archivo lógico o relación (Entidad)</b>							
53730	Perez José	M	10/03/3	044	73	Contado	2.000
28719	Balanagan	M	10/10/1	172	43	Abogado	1.800
53550	Lawrence	F	09/09/3	044	02	Escribano	1.100
79632	Rockefeller	M	01/11/3	090	11	Consulta	5.000
15971	Horseradish	F	25/02/6	172	07	Ingeniero	2.500
...	...	...	...	...	...	...	...

Identificador registro (entidad)      Conjunto de valores de un atributo o tipo de ítem de      Algunos atributos son si identificadores de otro registro (entidad)      Valores atributo

Fig. III-11

tiene un tipo específico de ítem de datos, relativo a un tipo de atributo dado. La columna de la izquierda contiene los ítems de datos que identifican a la entidad. En este ejemplo, la entidad es un empleado y el atributo designado como identificador de las instancias es el número de empleado.

En un modelo de entidad relación bien definido, las entidades y las relaciones deben estar en tercera forma normal [Chen 76], sin embargo, frecuentemente las entidades no están bien definidas e incluyen características de otras entidades.

### III.3.2. Relaciones

Una relación representa un conjunto de vínculos lógicos entre instancias de dos o más entidades. Cada una de las relaciones en un diagrama entidad relación tiene una semántica propia que es definida por el tipo de vínculo existente en el dominio del problema modelado. Desde un punto de vista matemático puede ser definido como:

Una relación entre entidades simples es una lista ordenada de entidades y una entidad dada puede aparecer una o más veces en la lista [Ullman 82].

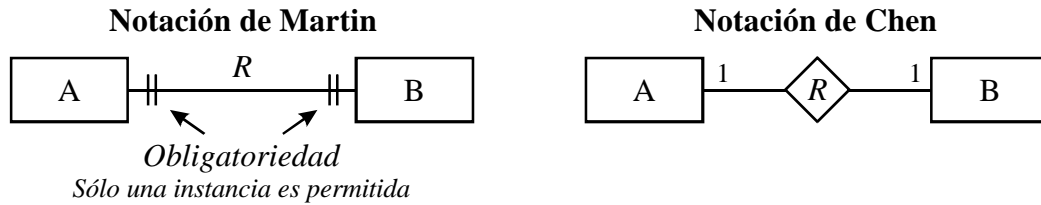
Si en un diagrama entidad relación hay una relación  $R$  entre las entidades  $E_1, E_2, \dots, E_n$ , representa un conjunto compuesto por las listas  $(e_1, e_2, \dots, e_n), (e_1', e_2', \dots, e_n') \dots$ ; donde las componentes  $e_i, e_i', \dots$  son instancias diferentes de la entidad  $E_i$ . La cantidad de entidades que participan en una relación es arbitraria, sin embargo, se recomienda la utilización de relaciones entre dos entidades, es decir, relaciones binarias.

Una entidad dada puede participar en más de una relación. Se pueden clasificar las relaciones binarias en diferentes tipos como base en la cantidad de participantes de cada una de las entidades.

En las siguientes secciones se definen los diferentes tipos de relaciones. Existen diferentes convenciones para la notación gráfica de las relaciones. En las secciones siguientes se utilizan las más usadas: la notación original de Chen [Chen 76] y la notación utilizada por James Martin [Martin 81], denominada también diagrama de Bachmann.

### III.3.2.1. Relación Uno-a-Uno

Una línea uniendo las entidades **A** y **B** representa una relación uno-a-uno. La barra corta, más interna, cruzando la línea de la relación (notación de Martin) indica la obligatoriedad de la relación, es decir, una ocurrencia de la entidad tiene que existir para que la relación tenga sentido.

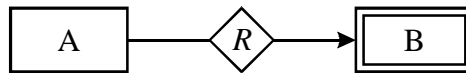


La figura representa gráficamente la siguiente regla:

- ✓ Cada ocurrencia de la entidad **A** esta relacionada a una y solo una ocurrencia de la entidad **B**.
- ✓ Cada ocurrencia de la entidad **B** esta relacionada a una y solo una ocurrencia de la entidad **A**.

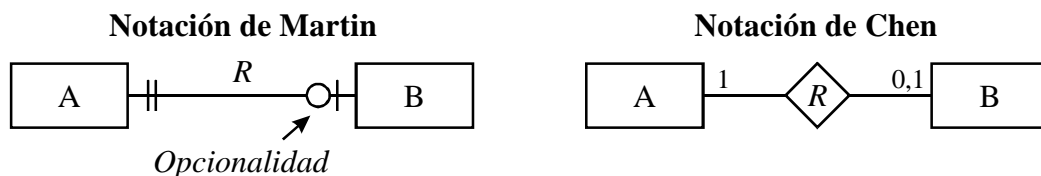
Por lo tanto, una ocurrencia, ni más ni menos, de la entidad **A** puede existir con una, ni más ni menos, ocurrencia de la entidad **B**. Esta relación es denominada relación *uno-a-uno obligatoria*. Las ocurrencias de las entidades **A** o **B** no pueden existir independientemente, una depende de la otra para existir.

Chen no considera la interpretación de la obligatoriedad en las relaciones. La notación en la parte superior de la relación es interpretada como la cantidad permitida de participantes. Sin embargo, considera una notación diferente para la dependencia de existencia por medio del uso de una flecha apuntando a la entidad dependiente. Por ejemplo, si la existencia de la entidad **B** depende de la existencia de la entidad **A**, se diseña una flecha apuntando a la entidad **B**.



#### III.3.2.1.1. Opcionalidad

Un círculo sobre la línea de la relación de lado de la entidad **B** indica una relación de opcionalidad. Mientras que la relación de **B** a **A** es obligatoria, la relación de **A** para **B** es opcional.



Esto es interpretado de la siguiente manera:

- ✓ Cada ocurrencia de la entidad **A** esta relacionada con cero o una ocurrencia de la entidad **B**.
- ✓ Cada ocurrencia de la entidad **B** (si existe) esta relacionada a una y solamente una ocurrencia de la entidad **A**.

Una entidad **A** puede existir sin la presencia de una entidad **B**. Mientras que si la entidad **B** existe, no puede haber mas que una ocurrencia de la entidad **A** relacionada. La entidad **B** no puede existir sin la presencia de la entidad **A**. La figura siguiente presenta una relación uno-a-uno opcional en los dos sentidos:





- ✓ Cada ocurrencia de la entidad **A** esta relacionada con cero o una ocurrencia de la entidad **B**.
- ✓ Cada ocurrencia de la entidad **B** esta relacionada con cero o una ocurrencia de la entidad **A**.

Tanto la ocurrencia de la entidad **A** como de la entidad **B** puede existir sin la presencia de la otra. Si la relación existe, cada ocurrencia de **A** puede estar relacionada solamente a una ocurrencia de la entidad **B** y viceversa.

### III.3.2.2. Relación Uno-a-Muchos

Las relaciones con varias instancias de una entidad se representa por medio del *signo menor*. Los siguientes ejemplos utilizan este tipo de relación:



Este ejemplo representa una relación uno-a-muchos obligatoria, debido a que las barras cortas cruzan a la línea de la relación. Este diagrama es interpretado de la siguiente manera:

- ✓ Cada ocurrencia de la entidad **A** esta relacionada a una o varias ocurrencias de la entidad **B**.
- ✓ Cada ocurrencia de la entidad **B** esta relacionada a uno y solamente una ocurrencia de la entidad **A**.

Ninguna de las entidades **A** o **B** pueden existir sin la presencia de la otra. La relación debe existir entre ocurrencias específicas de las entidades **A** y **B**. Una ocurrencia de la entidad **A** en particular puede estar relacionada a varias ocurrencias de la entidad **B**, debe haber por lo menos una ocurrencia de la entidad **B**. Por otro lado, una ocurrencia de la entidad **B** debe estar relacionada, siempre, a una y solo una ocurrencia de la entidad **A**.

#### III.3.2.2.1. Opcionalidad

La siguiente relación indica una relación *uno-a-muchos opcional* con **B**:

- ✓ Cada ocurrencia de la entidad **A** esta relacionada con cero, una o varias ocurrencias de la entidad **B**.
- ✓ Cada ocurrencia de la entidad **B**, si existe, será relacionada con una y solamente una ocurrencia de la entidad **A**.



Siempre que existe una ocurrencia de la entidad **B** ella debe estar relacionada a una ocurrencia de la entidad **A** y no más ni menos que una. Si una ocurrencia en particular de la entidad **A** esta relacionada a cero ocurrencias de la entidad **B**, la relación no existe para esa ocurrencia de la entidad **A**. Por otro lado, si la relación existe, ella puede ser con una o varias ocurrencias de la entidad **B**.



La relación anterior indica una relación *uno-a-muchos opcional* entre **A** y **B**:

- ✓ Cada ocurrencia de la entidad **A** esta relacionada con cero, una o varias ocurrencias de la entidad **B**.

- ✓ Cada ocurrencia de la entidad **B** esta relacionada con cero o una ocurrencia de la entidad **A**.

Las entidades **A** o **B** no necesitan existir. Si existen, deben estar relacionadas. Si existe una relación entre ellas, una ocurrencia específica de la entidad **A** puede estar relacionada con cero, una o varias ocurrencias de la entidad **B**. Cada una de las ocurrencias de la entidad **B** pueden estar relacionadas a solamente una ocurrencia de la entidad **A**. Por lo tanto, las ocurrencias de la entidad **B** relacionadas a una ocurrencia de la entidad **A** no pueden estar relacionadas a ninguna otra ocurrencia de la entidad **A**.

### III.3.2.3. Relación Muchos-a-Muchos

Dos relaciones *uno-a-muchos* para ambos lados pueden existir entre entidades, ellas se convierten en una sola relación *muchos-a-muchos* y es representada de la siguiente manera:



- ✓ Cada ocurrencia de la entidad **A** esta relacionada con una o varias ocurrencias de la entidad **B**.
- ✓ Cada ocurrencia de la entidad **B** esta relacionada con una o varias ocurrencias de la entidad **A**.

#### III.3.2.3.1. Opcionalidad



- ✓ Cada ocurrencia de la entidad **A** esta relacionada con cero, una o varias ocurrencias de la entidad **B**.
- ✓ Cada ocurrencia de la entidad **B**, si existe, esta relacionada con una o varias ocurrencias de la entidad **A**.



- ✓ Cada ocurrencia de la entidad **A**, si existe, esta relacionada con cero, una o varias ocurrencias de la entidad **B**.
- ✓ Cada ocurrencia de la entidad **B**, si existe, esta relacionada con cero, una o varias ocurrencias de la entidad **A**.

### III.3.2.4. Relaciones Indefinidas

Se ha descrito cómo se representan gráficamente las relaciones *uno-a-uno* y *uno-a-muchos*, *obligatoria* y *opcional*. Sin embargo, cuando se esta desarrollando un modelo entidad relación puede suceder que no se conozca el tipo de relación existente y que el tipo de relación no este hasta el momento definida. En estos casos la relación es descrita de la siguiente manera:



### III.3.3. Mecanismos de Abstracción

En la construcción de diagramas entidad relación existen mecanismos que permiten modelar diversos tipos de abstracción, muy útiles en la organización conceptual de los modelos de datos.

#### III.3.3.1. Clasificación

El mecanismo de clasificación fue introducido intuitivamente, puesto que los tres conceptos básicos en los que se basan los diagramas entidad relación fueron desarrollados como una aplicación de abstracciones de clasificación:

- ✓ *Entidad*: Una entidad es una clasificación que representa un conjunto de objetos con características comunes.
- ✓ *Atributos*: Un atributo es una clasificación que representa un conjunto de valores de una propiedad atómica de una entidad o una relación.
- ✓ *Relación*: Una relación es una clasificación que representa el conjunto de vínculos entre objetos integrantes del mismo conjunto de entidades.

#### III.3.3.2. Agregación de Atributos (atributos compuestos)

Un atributo de una entidad o relación puede ser una estructura compuesta por ítems que se desean identificar. La Fig. III-12 presenta una entidad *Cliente* con un atributo compuesto *Dirección*.

#### III.3.3.3. Especialización (es-un o es-subtipo-de)

La relación *es-un* o *es-subtipo-de* es una relación muy común en una clasificación de entidades. Es útil si existen entidades con la mayoría de la características comunes, pero con algunas características diferentes. La Fig. III-13 presenta un ejemplo.

Una especialización también puede ser útil si solamente un sub-conjunto de ocurrencias de las entidades, a ser relacionadas, participan en la relación.

#### III.3.3.4. Agregación de Entidades (compuesto-por)

La relación *compuesto-por* es una relación que permite describir composición, por ejemplo la

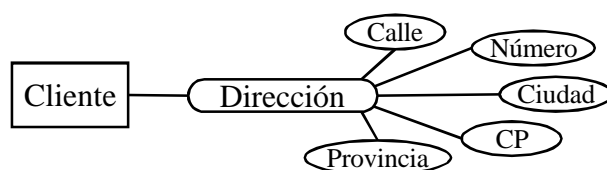


Fig. III-12: Ejemplo de atributos compuestos

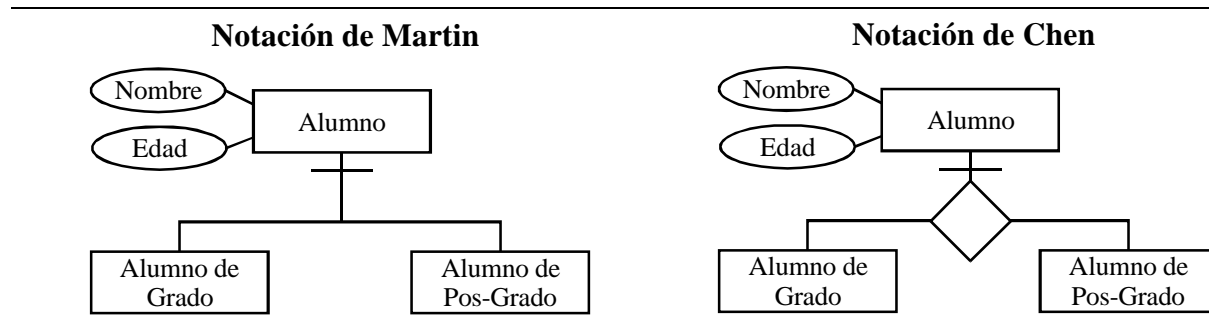


Fig. III-13

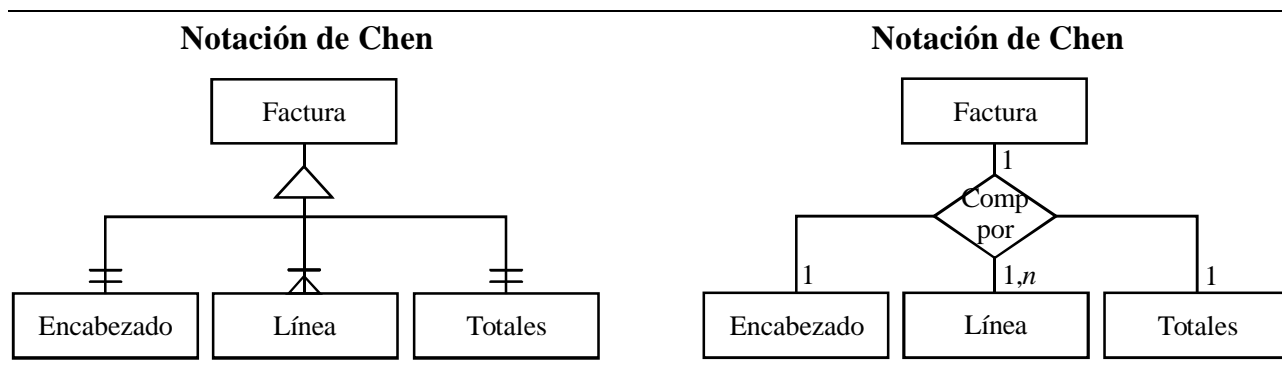


Fig. III-14

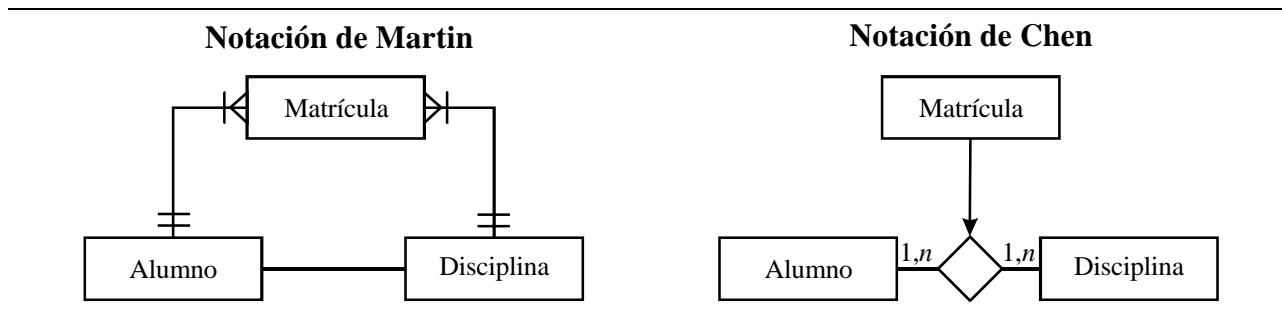


Fig. III-15

composición de una factura como se describe en la Fig. III-14.

### III.3.3.5. Entidades Relacionantes

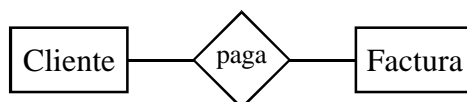
Existen situaciones en las cuales una relación se convierte en una entidad. Por ejemplo, si una relación tiene atributos asociados a ella, es una entidad sin perder su propiedad de vínculo entre entidades. La Fig. III-15 muestra un ejemplo.

Note que la notación de Martin no hace diferencia entre los dos tipos de entidades. Sin embargo, en la notación de Chen la relación convertida en entidad es notoriamente identificable.

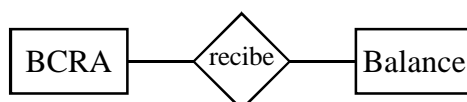
## III.3.4. Construcción de un Diagrama Entidad-Relación

Existe un conjunto de pasos los cuales guían el proceso de construcción de un modelo entidad relación, a partir de una lista de eventos, los cuales son descriptos a continuación:

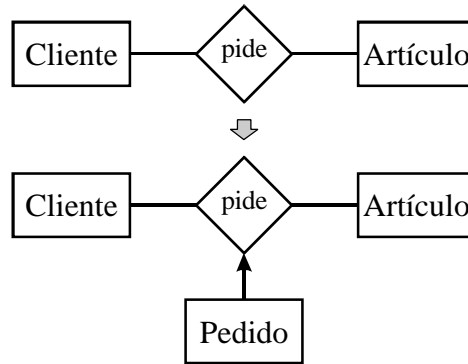
1. Para cada evento construir una relación
  - a. El sujeto del evento es una de las entidades de la relación.
  - b. El predicado del evento es la otra entidad de la relación.
  - c. El verbo del evento es el nombre de la relación.



2. Eliminar las entidades que no posean datos que identifiquen instancias diferentes.



3. Identificar relaciones que puedan servir como entidades asociativas



4. Construir el modelo resultante.
5. Identificar entidades demasiado generales o grupos de entidades demasiado particulares y construir relaciones de especialización.
6. Identificar relaciones de composición.
7. Identificar entidades poco significativas.
8. Completar el modelo de datos. Para cada entidad, cada relación y cada entidad asociativa, completar la correspondiente entrada en el diccionario de datos.

## III.4. Especificación de Procesos

Una especificación de procesos describe las actividades a ser desarrolladas para transformar los datos de entrada en datos de salida. Existen diversas herramientas para la especificación de procesos: tablas de decisión, árboles de decisión, lenguajes estructurados, pre/pos condiciones y diagramas de Nassi-Shneiderman, entre otras. En general, una herramienta puede ser utilizada para la especificación de procesos si cumple los siguientes dos requisitos:

- ✓ Una especificación de procesos debe ser expresada de forma tal que pueda ser verificada por el usuario y por el analista de sistemas. Por esta razón, no es recomendable utilizar el uso de lenguajes de programación, tampoco es recomendable el uso de lenguaje natural.
- ✓ Una especificación de procesos debe ser expresada de forma tal que pueda ser efectivamente comunicada a la audiencia involucrada. Habitualmente hay una audiencia diversa de usuarios, gerentes, auditores y controladores de calidad, los cuales leerán la especificación.

La especificación de procesos es realizada solo para los procesos de los niveles más bajos del diagrama de flujo de datos. Los procesos de un nivel intermedio son definidos por los diagramas de flujos de datos del nivel inferior inmediato. Sin embargo, en otros diagramas como por ejemplo el diagrama de estructura, todos los componentes deben ser especificados.

### III.4.1. Lenguaje Estructurado

Un lenguaje estructurado es un subconjunto del lenguaje natural con algunas restricciones en cuanto al tipo de comandos que pueden ser utilizados y la manera en que esos comandos pueden ser utilizados. Los lenguajes estructurados también son conocidos por otros nombres:

- ✓ Lenguaje de Diseño de Programas
- ✓ Lenguaje de Especificación de Problema
- ✓ Pseudo-código

El propósito es obtener un equilibrio razonable entre la precisión de un lenguaje de programación formal y, la informalidad y legibilidad del lenguaje que se utiliza normalmente. Los comandos en estos lenguajes pueden consistir en una ecuación algebraica o en una declaración imperativa sintética en un lenguaje natural. Las estructuras de control comúnmente usadas en este tipo de lenguajes son las descriptas a continuación:

Estructuras de Control			
<b>Secuencia</b>	S1 S2 . . . Sn	S1 S2 . . . Sn	Una secuencia de comandos simples es equivalente (estructuralmente) a un comando simple.
<b>Condicional</b>	Si C S1 Si no S2 Fin Si	IF C S1 ELSE S2 END IF	Una construcción Si-Sino (IF-ELSE) simple es considerada estructuralmente equivalente a una única sentencia simple. Esta construcción puede ser anidada. También se pueden utilizar múltiples sentencias condicionales.
<b>Repetición</b>	Mientras C S1 Fin	DOWHILE C S END DO	Una estructura Mientras (DO WHILE) simple es considerada estructuralmente equivalente a una única sentencia simple. Esta estructura puede ser anidada.

Una especificación de procesos no debe ser muy compleja. Existen tres consejos que pueden ayudar en su construcción:

- ✓ Restrinja la especificación de procesos en lenguaje estructurado a una única página de texto. Si una especificación necesita más de una página, el analista de sistemas debe utilizar otra manera de realizar la especificación. Si esto no es posible, probablemente el proceso (o módulo) es demasiado complejo y deba ser explotado en un DFD de nivel más bajo (o en más de un módulo en el diagrama de estructuras).
- ✓ No utilice más de tres niveles de anidamiento. Principalmente en el caso de estructuras **Si-Sino**, más de dos niveles de anidamiento representan un fuerte indicio que sería preferible utilizar la herramienta *tabla de decisión*.
- ✓ Evite confusiones sobre niveles de anidamiento utilizando indentación.

### III.4.2. Pre/Pos Condiciones

Esta herramienta presenta un modo práctico de describir una función que debe ser ejecutada por un proceso, sin que sea necesario extenderse demasiado sobre el algoritmo o procedimiento. Esta herramienta es particularmente útil cuando:

- ✓ El usuario tiene que expresar la política ejecutada por un proceso en términos de un algoritmo especial y particularizado que ha utilizado durante mucho tiempo.
- ✓ El analista de sistemas está razonablemente seguro que existen muchos algoritmos que pueden ser utilizados.
- ✓ El analista de sistemas quiere dejar que el programador explore algunos algoritmos pero no desea involucrarse personalmente.

En la Fig. III-16 se da un ejemplo de aplicación de Pre/Pos Condición.

Las *pre-condiciones* describen todo lo que debe ser verdadero (si hubiese algo) antes de la ejecución del proceso. Muchas veces es práctico imaginar pre-condiciones como un disparador para que un proceso pueda trabajar. También pueden ser consideradas como una garantía de ausencia de errores de datos de entrada cuando un proceso es activado. En general describen:

- ✓ *Qué entradas deben estar disponibles*. Estas entradas serán recibidas por medio de un flujo de datos. Puede haber casos en que haya varios flujos llegando a un proceso y uno de ellos sea una pre-condición necesaria para la activación del proceso.

### ESPECIFICACIÓN DE PROCESO 3.5: Calcular Impuesto Sobre Ventas

#### Pre-Condición 1:

**Datos-de-Venta** ocurre con **Tipo-de-Ítem** que coincide con **Categoría-de-Ítem** en **Categorías-de-Impuestos**

#### Pos-Condición 1:

**Tasa-Ventas** es ajustada en **Total-Ventas \* Valor-Tasa**

#### Pre-Condición 2:

**Datos-de-Venta** ocurre con **Tipo-de-Ítem** que no coincide con **Categoría-de-Ítem** en **Categorías-de-Impuestos**

#### Pos-Condición 2:

Mensaje-Error(**Ítem inválido**) es generado

Fig. III-16: Ejemplo de especificación de proceso utilizando pre/pos condiciones

- ✓ *Qué relación debe existir entre las entradas con su valor.* Con frecuencia una pre-condición especificará que deben llegar entradas con campos relacionados.
- ✓ *Qué relación debe existir entre entradas y depósitos de datos.* Una pre-condición puede estipular que exista un registro en un depósito que coincida con algún aspecto de un elemento de entrada.
- ✓ *Qué relación debe existir entre diferentes depósitos y el interior de un repositorio de datos.* Una pre-condición puede estipular que exista un registro en un repositorio que tenga alguna componente que coincida con alguna otra componente de un registro de otro repositorio de datos.

Las *pos-condiciones* describen qué debe ser verdadero cuando un proceso termina su tarea. Esto puede ser considerado como una garantía de validación de la función ejecutada por el proceso. Cuando el proceso es terminado, la pos-condiciones deben ser verdaderas. Las pos-condiciones habitualmente describen lo siguiente:

- ✓ *Las salidas que son producidas por el proceso.* Es la forma más común de pos-condición.
- ✓ *Las relaciones que existen entre los valores de salida y los valores originales de entrada.* Esto es común en situaciones en que una salida es una función matemática directa de un valor de entrada.
- ✓ *La relación que existe entre los valores de salida y los valores de uno o más repositorios de datos.* Esto es común cuando debe ser recuperada información de uno o más repositorios para completar la salida del proceso.
- ✓ *Las alteraciones que deben ser realizadas en los depósitos.* Incorporación de nuevos ítems, modificación o eliminación de ítems existentes.

Todos los datos (no locales) deben estar especificados en el diccionario de datos, por ejemplo considérese las siguientes definiciones en el diccionario de datos:

**Planif-Análisis** := \* Especificación de la fecha y procedimiento de análisis \*  
@Nro-paciente + @Tipo-Anal. + Fecha\_Anal. + Hora-Anal.

**Análisis** := (Planif-Análisis)

**Diario-Anal** := (Fecha-Anal. + Hora-Anal. + (Tipo-Anal. + Datos-Pac.))

La especificación de procesos correspondiente sería la siguiente:

#### Pre-condición

Existen **Planif-Análisis** cuya **Fecha** es el día siguiente (mañana)

#### Pos-condición

Se emite **Diario-Anal.** del **Planif-Análisis**

Conviene resaltar que suele ser conveniente utilizar términos locales dentro de la especificación de procesos para describir cálculos intermedios o relaciones entre entradas y datos guardados.

### III.4.3. Tabla de Decisión

Existen situaciones en las que el lenguaje estructurado o pre/pos condición no son adecuados para realizar una especificación de procesos. Esta situación ocurre cuando la lógica del proceso se basa en decisiones complejas. Si las decisiones se basan en muchas variables y estas variables pueden asumir varios valores diferentes, la especificación de procesos puede ser realizada utilizando tablas de decisión, como se muestra en el siguiente ejemplo:

	1	2	3	4	5	6	7	8
<b>Edad &gt; 21</b>	S	S	S	S	N	N	N	N
<b>Sexo</b>	S	S	N	N	S	S	N	N
<b>Peso &gt; 90</b>	S	N	S	N	S	N	S	N
<b>Medición 1</b>	✓				✓			✓
<b>Medición 2</b>		✓			✓			
<b>Medición 3</b>			✓			✓		✓
<b>Ninguna Medición</b>				✓			✓	

Como se describe en la tabla, una tabla de decisión es construida relacionando las variables interesadas con acciones a ser ejecutadas según alguna combinación particular de los valores de las variables (primera columna).

En este ejemplo, todas las variables son booleanas, es decir, sólo pueden tomar uno de dos valores: verdadero-falso, femenino-masculino, si-no, etc. Esta es una característica deseable pero no obligatoria. Cuando todas las variables son booleanas la tabla toma la forma de las denominadas usualmente como *tabla de condición / acción*.

Las otras columnas contienen todas las posibles combinaciones de valores de las variables. La parte inferior de la tabla describe la acción a tomar cuando esa combinación es obtenida. Si existen N variables con valores booleanos, entonces habrá  $2^N$  combinaciones diferentes.

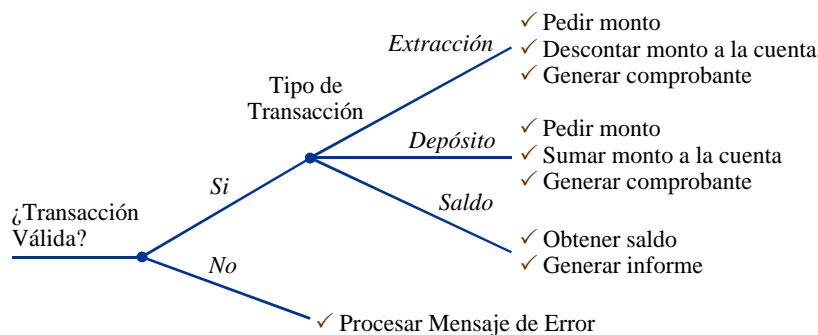
Para la creación de una tabla de decisión se deben seguir las siguientes etapas:

1. Identificar todas las condiciones o variables de la especificación. Identificar todos los valores que cada variable puede asumir.
2. Calcular el número de combinaciones de condiciones posibles. Si todas las condiciones fuesen booleanas (o binarias en general), habrá  $2^N$  combinaciones de N variables.
3. Identificar cada acción posible en la especificación.
4. Crear una tabla de decisiones *vacía*, relacionando todas las condiciones y acciones en el lado izquierdo, y enumerando las combinaciones de las condiciones en la parte superior de la tabla.
5. Relacionar todas las combinaciones de condiciones, una para cada columna de la tabla.
6. Examinar cada columna e identificar las acciones adecuadas a ser ejecutadas.
7. Identificar todas las omisiones, contradicciones y ambigüedades de la especificación
8. Discutir las omisiones, contradicciones y ambigüedades con el usuario.

### III.4.4. Árboles de Decisión

Un *árbol de decisión* sirve para modelar funciones discretas, en las que el objetivo es determinar el valor combinado de un conjunto de variables, y basándose en el valor de cada una de ellas, determinar la acción a ser tomada.





**Fig. III-17: Ejemplo de Árbol de Decisión**

Los árboles de decisión son normalmente contruidos a partir de la descripción de la narrativa de un problema. Ellos proveen una visión gráfica de la toma de decisión necesaria, especifican las variables que son evaluadas, qué acciones deben ser tomadas y el orden en la cual la toma de decisión será efectuada. Cada vez que se ejecute un árbol de decisión, solo un camino será seguido dependiendo del valor actual de la variable evaluada.

Considérese el siguiente ejemplo:

*En un cajero automático es posible realizar varias transacciones diferentes. Si el cliente desea realizar una transacción no válida un mensaje de error es generado. El cliente puede realizar las siguientes transacciones: consulta de saldo, extracción o depósito. Si selecciona consulta de saldo se busca el saldo del cliente y se genera un informe el cual es entregado al cliente. Si se desea realizar una extracción o un depósito se pide el monto, se actualiza la cuenta ya sea incrementando o descontando el monto al saldo y se genera un comprobante para el cliente.*

El árbol de decisión mostrado en la Fig. III-17 corresponde a la lógica del proceso descrito con la narrativa anterior.

### III.4.5. Diagramas de Nassi-Shneiderman

El diagrama de Nassi-Shneiderman surge a mediados de los años '70, cuando la programación estructurada se tornó popular. Ellos adicionan una representación visual de los lenguajes estructurados en la forma mostrada en la Fig. III-18.

A continuación se describen las estructuras básicas de este tipo de diagrama.

<b>Secuencia</b>	<div style="border: 1px solid black; padding: 5px; width: fit-content;"> <div style="border-bottom: 1px solid black; padding-bottom: 5px;">Sentencia 1</div> <div style="padding-bottom: 5px;">Sentencia 2</div> </div>	Una secuencia de sentencias simples es equivalente (estructuralmente) a una sentencia simple. Por lo tanto, una secuencia puede ser utilizada donde se pueda utilizar una sentencia simple.
<b>Alternativa</b>	<div style="border: 1px solid black; padding: 5px; width: fit-content;"> <div style="border-bottom: 1px solid black; padding-bottom: 5px; display: flex; justify-content: space-between;"> <span>V</span> <span>Cond</span> <span>F</span> </div> <div style="padding-bottom: 5px;"></div> </div>	Una construcción IF-THEN-ELSE simple es considerada estructuralmente equivalente a un única sentencia simple. Esto permite que puedan anidarse.
<b>Repetición</b>	<div style="border: 1px solid black; padding: 5px; width: fit-content;"> <div style="border-bottom: 1px solid black; padding-bottom: 5px; display: flex; justify-content: space-between;"> <span></span> <span>Condición</span> </div> <div style="padding-bottom: 5px;"></div> </div>	Una estructura DO WHILE simple es considerada estructuralmente equivalente a una única sentencia simple. Esto permite que puedan anidarse.

Sentencia 1		
Condición 1		
Condición 2		
Sentencia 2 v1	Sentencia 2 f1	
Sentencia 2 v2	Sentencia 2 f2	
Sentencia 2 v3	Condición 3	
Sentencia 2 v4	Sent. 3 v1	Sent. 3 f1
Sentencia 2 v5	Sent. 3 v2	Sent. 3 f2
Sentencia 2 v6	Sent. 3 v3	Sent. 3 f3

Fig. III-18: Ejemplo de diagrama Nassi-Shneiderman

## III.5. Diagramas de Transición de Estados (DTE)

Los diagramas de *transición de estados* sirven para el modelamiento de actividades donde un conjunto finito de estados diferentes puede ser reconocido. Ellos modelan los diferentes estados en los que puede estar una actividad, las condiciones de los eventos de transición entre los estados y las acciones que deben ser ejecutadas en el momento de cambiar de estado.

Los diagramas de transición de estados tienen una interpretación ejecutable basada en una teoría formal: las máquinas de estados finitos.

Una máquina de estados finitos es un mecanismo hipotético que puede estar en una colección discreta de estados, en un punto discreto del tiempo. Ciertos eventos pueden forzar el cambio de estado de la máquina a otro estado de la colección.

Los eventos de transición de estados ocurren en puntos discretos en el tiempo. No existen cambios lentos y continuos. La ocurrencia de un evento causa una transición *instantánea* en el estado de la máquina. Los eventos pueden ocurrir de modo asincrónico (en cualquier punto en el tiempo) o de modo sincrónico (en intervalos).

Comúnmente, los estados son diseñados como círculos, aunque algunos autores los diseñan con cajas rectangulares [Ward 85]. Las transiciones son diseñadas con flechas que unen dos estados apuntando al estado destinatario de la transición.

### III.5.1. Estados

Para el modelamiento usando diagramas de transición de estado, tal vez la actividad más difícil es la identificación de los diferentes estados. Si el problema es adecuado para ser modelado con DTE el conjunto de estados debe ser reconocido con relativa facilidad.

Un estado representa un modo de comportamiento externamente observable. El nombre del estado es el nombre del comportamiento. Existen dos tipos de estados:

- ✓ *Estado inicial*: Antes que ocurra cualquier transición
- ✓ *Estado final* (uno o más): Fin del comportamiento del sistema.

Un tipo de problema comúnmente adecuado para el modelo con DTE, es el diseño de interfaces hombre-máquina. La Fig. III-19 muestra un ejemplo. En estos problemas, los estados son, en general, situaciones fácilmente identificables donde el sistema espera por eventos de dispositivos de

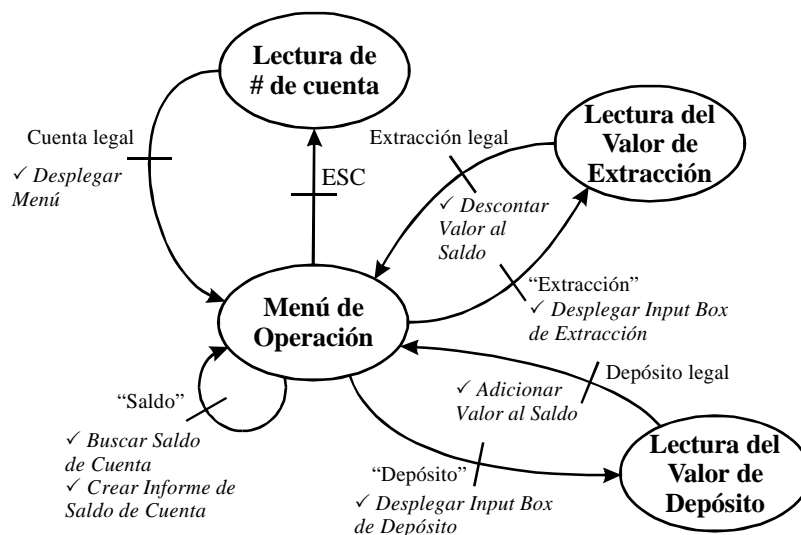


Fig. III-19: Modelo de Interfaz de Operaciones en Cuenta Corriente

entrada (teclado, lectores ópticos etc.) o dispositivos apuntadores como el mouse. Múltiples extensiones a los DTE existen para el modelamiento de este tipo de problema.

También son útiles para el modelamiento de problemas complejos de sincronización, por ejemplo en aplicaciones de bases de datos o procesamiento concurrente.

### III.5.2. Transiciones

Las transiciones entre estados en un DTE son producidas como resultado de la ocurrencia de un evento en el sistema. Los eventos son producidos cuando una *condición* es verdadera. En el ejemplo de la Fig. III-19, cuando el sistema está en el estado **Lectura del Valor de Extracción** y la condición **Extracción Legal** es verdadera, el estado del sistema cambia a **Menú de Operación**.

También es posible especificar las *acciones* que deben ser ejecutadas en el momento de la transición. Una acción es una actividad indivisible, no es importante el orden en el cual se realizan las actividades involucradas a menos que una secuencia explícita sea especificada. En el DTE de la Fig. III-19, cuando el sistema está en el estado **Menú de Operación** y la condición **“Saldo”** es verdadera (se selecciona la operación **“Saldo”** desde el menú de operaciones) inmediatamente se ejecutan las acciones **Buscar Saldo de la Cuenta** y **Crear Informe de Saldo de Cuenta**. Una vez ejecutadas las acciones, la transición puede ser completada.

### III.5.3. Representación en Tabla de Transición

Los diagramas de transición de estados pueden ser representados en una matriz de transición de estados, bidimensional, que contiene la misma información modelada en la versión gráfica. La tabla siguiente representa una *Tabla de Transición* del DTE de la Fig. III-19:

Condición	Estado			
	(E1) Lectura del Nro. De Cuenta	(E2) Menú de Operación	(E3) Lectura del Valor de Extracción	(E4) Lectura del Valor de Depósito
<b>Cuenta Legal</b>	⇒ E2 ✓ <i>Desplegar Menú</i>			
<b>ESC</b>		⇒ E1		
<b>Consulta Saldo</b>		⇒ E2 ✓ <i>Buscar Saldo</i> ✓ <i>Crear Informe</i>		
<b>Extracción</b>		⇒ E3 ✓ <i>Desplegar Input Box de Extracción</i>		
<b>Depósito</b>		⇒ E4 ✓ <i>Desplegar Input Box de Depósito</i>		
<b>Extracción Legal</b>			⇒ E2 ✓ <i>Descontar valor al saldo</i>	
<b>Depósito Legal</b>				⇒ E2 ✓ <i>Adicionar valor al saldo</i>

⇒ : Transición para el estado *E*

✓ *Acción* : Acción asociada a la transición

### III.5.4. Representación en Diagramas de Grade

Cuando el problema modelado representa un mecanismo complejo, puede haber muchas transiciones entre un número relativamente pequeño de estados y la representación gráfica de un DTE se torna confusa. En estos casos se utiliza un *diagrama de grade*. Un diagrama de grade es una representación de un DTE donde los estados son representados por líneas verticales y las transiciones por flechas horizontales. La Fig. III-20 muestra el ejemplo de la Fig. III-19 modelado con un diagrama de grade.

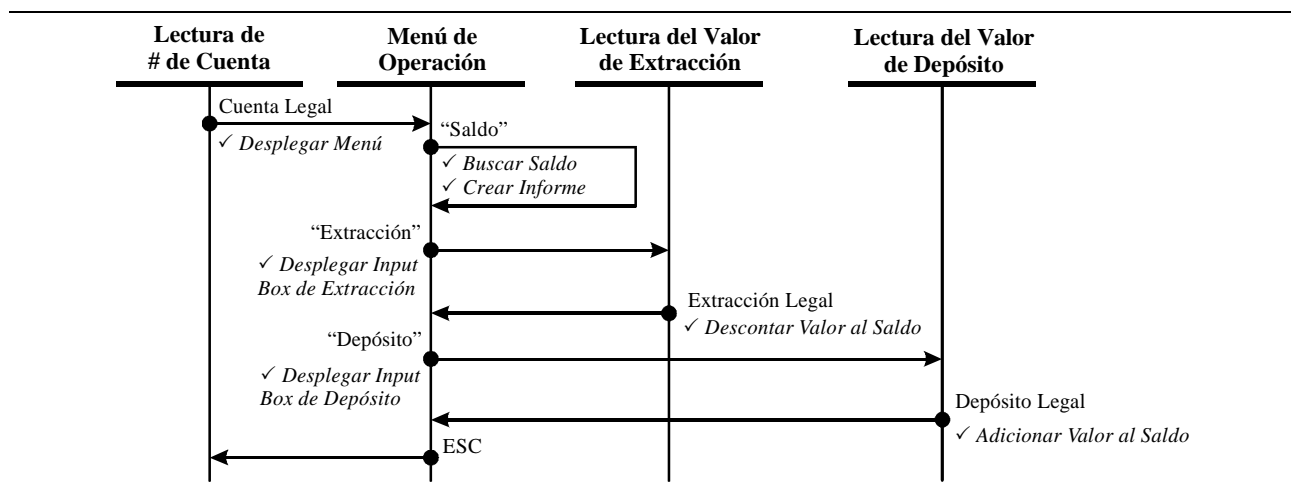


Fig. III-20: Diagrama de Grade para el Modelo de Interfaz de Operaciones en Cuenta Corriente



## Capítulo IV. ASML: Análisis Estructurado de Sistemas

### Contenidos

<b>Introducción .....</b>	<b>62</b>
<b>Modelo de Implementación del Usuario .....</b>	<b>63</b>
Determinación de los Límites de la Automatización .....	64
Determinación de la Interfaz del Usuario .....	65
Identificación de Actividades Manuales Complementarias.....	67
Especificación de Restricciones Operacionales .....	67
<b>Diagrama de Estructura.....</b>	<b>68</b>
Módulos .....	69
Relaciones entre Módulos (Invocaciones) .....	69
Comunicación entre Módulos (Cuplas) .....	70
Absorción .....	71
Estructuras de Control.....	71
<b>Derivación de los Diagramas de Estructura.....</b>	<b>72</b>
Análisis de Transformaciones .....	72
Análisis de la Especificación del Problema .....	72
Identificar el Centro de Transformación .....	73
Estrategia para Determinar el Centro de Transformación .....	74
Producir un Primer Diagrama de Estructura (First-Cut) .....	76
Mejorar el Diagrama de Estructura Obtenido .....	76
Garantizar la Funcionalidad del Diseño .....	77
Análisis de Transacción .....	77
<b>Criterios de Validación de Calidad .....</b>	<b>80</b>
Acoplamiento .....	81
Acoplamiento sin Cuplas .....	81
Acoplamiento de Datos .....	82
Acoplamiento Estampado .....	83
Acoplamiento de Control .....	83
Acoplamiento Híbrido.....	84
Acoplamiento Común .....	85
Acoplamiento por Contenido (o Patológico).....	85
Cohesión.....	85
Cohesión Funcional.....	86
Cohesión Secuencial .....	86
Cohesión Comunicacional.....	87
Cohesión Procedural .....	88
Cohesión Temporal .....	88
Cohesión Lógica.....	89
Cohesión Coincidental.....	89
Clusters de Información (Cohesión Informacional) .....	89
Determinación de la cohesión de un módulo .....	90
Descomposición (Factoring) .....	90
Reducir el Tamaño del Módulo.....	90

Hacer el Sistema más Claro.....	91
Minimizar la Duplicación de Código .....	91
Separar el Trabajo de la ‘Administración’ .....	91
Crear Módulos más Generales.....	91
Fan-Out.....	91
Fan-In .....	92
Criterios Adicionales de Diseño.....	92
Evitar División de Decisiones .....	92
Forma del Sistema .....	92
Sistemas Dirigidos por Entradas Físicas (Physically Input-Driven) .....	93
Sistemas Balanceados.....	94
Tratamiento de Errores .....	94
Relación entre Estructuras de Datos y Estructura de Programa .....	96
Memoria Estática.....	96
Módulos de Inicialización y Terminación .....	97
Edición.....	97

El diseño estructurado de sistemas se preocupa por la identificación, selección y organización de los módulos y sus relaciones. Se comienza con la especificación resultante del proceso de análisis, se realiza una descomposición del sistema en módulos estructurados en jerarquías, con características tales que permitan la implementación de un sistema que no requiera elevados costos de mantenimiento.

La idea original del diseño estructurado fue presentada en la década de los '70, por Larry Constantine [Constantine 74], y continuadas posteriormente por varios autores [Myers 78; Yourdon 78; Stevens 81].

## IV.1. Introducción

El diseño estructurado es un enfoque disciplinado de la transformación de *qué* es necesario para el desarrollo de un sistema, a *cómo* deberá ser hecha la implementación.

La definición anterior implica que: el análisis de requerimientos del usuario (determinación del *qué*) debe preceder al diseño y que, al finalizar el diseño se tendrá medios para la implementación de las necesidades del usuario (el *cómo*), pero no se tendrá implementada la solución al problema. Cinco aspectos básicos pueden ser reconocidos:

- ✓ Permitir que *la forma del problema guíe a la forma de la solución*. Un concepto básico del diseño de arquitecturas es: *las formas siempre siguen funciones*. Ese concepto debe ser también utilizado para el diseño de sistemas. Muchas veces, se han escogido formas preconcebidas para la solución de un problema y forzado a un problema a tomar la misma forma. Un enfoque adecuado debe ser: lograr que, la solución de un problema tenga los mismos componentes que los del problema original y respete las relaciones que existen en el problema. El objetivo debe ser obtener la mayor aproximación posible a esa idea.
- ✓ Intentar *resolver la complejidad de los grandes sistemas* a través de la segmentación de un sistema en *cajas negras*, y su organización en una jerarquía conveniente para la implementación. Además, no es necesario conocer cómo trabajan internamente esas cajas negras, pues su complejidad ya fue resuelta (separadamente de las otras). Para usarlo, solamente se precisa conocer las entradas que deben ser provistas, la funcionalidad que implementa y las salidas que retorna.

- ✓ Utilizar *herramientas*, especialmente *gráficas*, para realizar diseños de fácil comprensión. Un diseño estructurado usa diagramas de estructura (DE) en el diseño de la arquitectura de módulos del sistema y adiciona especificaciones de los módulos y cuplas (entradas y salidas de los módulos), en un Diccionario de Datos (DD).
- ✓ Ofrecer un conjunto de *estrategias para derivar el diseño* de la solución, basándose en los resultados del proceso de *análisis*. Principalmente, ofrece dos heurísticas (*Análisis de Transformaciones* y *Análisis de Transacciones*) para la derivación de los DE, partiendo de los diagramas de flujo de datos (DFD) construidos en el proceso de análisis.
- ✓ Ofrecer un conjunto de criterios para evaluar la calidad de un diseño con respecto al problema a ser resuelto, y las posibles alternativas de solución, en la búsqueda de la mejor de ellas. Permite clasificar varias características de los DE (y de los módulos y cuplas componentes) basado en los problemas potenciales que pueden ocasionar al sistema, una vez implementado. Una identificación de esos problemas, y una subsecuente corrección de errores en la etapa de diseño, disminuyen en gran medida los costos de mantenimiento (sea para corrección de errores o incorporación de nueva funcionalidad).

El diseño estructurado produce sistemas fáciles de entender y mantener, confiables, fácilmente desarrollados, eficientes y que funcionan.

## IV.2. Modelo de Implementación del Usuario

Con el Modelo Ambiental y el Modelo Comportamental completamos el desarrollo del Modelo Esencial. El modelo esencial contiene una especificación completa del sistema de información de las necesidades del usuario. Específicamente describe:

- ✓ La acción esencial de las funciones que deben ejecutarse.
- ✓ El volumen esencial de los datos usados y guardados por el sistema.
- ✓ El comportamiento esencial necesario para trabajar con las señales y las interrupciones del ambiente externo.

Una especificación de la funcionalidad, estructura el comportamiento esencial del sistema, debe ser suficiente para que el diseñador, pueda escoger el mejor hardware, el mejor sistema operativo, el mejor administrador de base de datos, el mejor lenguaje de programación, dentro de las restricciones generales de tiempo, dinero y recursos humanos destinados al proyecto de implementación. Descrito así, el trabajo del diseñador parece simple, pero no lo es: hay muchas características en las que ellos necesitan del usuario para que éstas sean definidas.

Estas características, incluso las de implementación (sin formar parte del Modelo Esencial) son suficientemente importantes ya que tienen gran impacto en el usuario del sistema, por lo tanto necesitan ser especificadas. El problema de implementación más evidente es el interés del usuario en la Frontera de Automatización, es decir, que parte del modelo esencial se llevara a cabo en la computadora, y que se deja para que sea ejecutada manualmente. El analista, ciertamente, tendrá una opinión en eso, y el diseñador también, pero éste es un problema en el que el usuario tiene la última palabra.

También, la interfaz hombre-máquina es de gran interés para el usuario, muchas veces él parece estar más interesado en las interfaces que en las funciones del sistema. Hay muchas teorías desarrolladas respecto a este problema, del factor humano y organización de la información, de la pantalla de la computadora, de la optimización de la comprensión del usuario, de criterios y técnicas acerca de la interacción y estructuración de las interfaces. Sin embargo, el usuario tiene la última palabra porque él es quién las *sufre* o las *aprecia*.

Más recientemente varias opciones y posibilidades, aumentaron la importancia de esos problemas de implementación del usuario:

- ✓ Los usuarios finales muchas veces tienen la oportunidad de usar las computadoras personales (PC) como parte de una red distribuida de computadoras. Esto genera una serie de cuestiones: ¿Qué partes del modelo esencial estarán disponibles en las computadoras terminales y qué partes en el servidor (mainframe o estaciones RISC)? Esto implica: ¿A qué datos se tendrá acceso por intermedio del PC, y qué datos serán almacenados en el PC? ¿Qué funcionalidad tendrá que ejecutar el PC? ¿Cuál será el formato de entradas y de salidas en el PC? ¿Qué actividades adicionales de apoyo deben ofrecerse para asegurar que el usuario no dañe desprevadamente los datos del PC (o del servidor)?
- ✓ Hay muchas ofertas de administradores de base de datos con lenguajes de cuarta generación que permiten, al usuario, escribir sus propios programas. ¿Con qué partes del sistema ellos pueden interactuar sin riesgos de producir inconsistencia en los datos?
- ✓ Hoy día, en muchas situaciones, el usuario y el analista pueden decidir prototipar el sistema y usar un lenguaje de cuarta generación o un paquete generador de aplicaciones. La prototipación puede hacerse porque el usuario no está seguro de la acción detallada que en el futuro tendrá que ser descrita en las especificaciones de procesos del modelo esencial; aun así, la mayoría de las veces, la prototipación se usa para la exploración y experimentación con formatos de entrada, diálogos en línea y formatos de salida para las pantallas e informes.
- ✓ Para muchas aplicaciones de una compañía, una opción es la de adquisición de un paquete de software. En estos casos se generan los mismos problemas de implementación: ¿Qué partes del modelo esencial serán llevados a cabo por el paquete adquirido y qué partes tendrán que ser llevadas a cabo en un sistema separado? ¿Cuáles serán las normas de integración y compatibilidad entre el paquete adquirido y el sistema a ser desarrollado?

Estos problemas deben ser tratados como parte del *Modelo de Implementación del Usuario*. La construcción de este nuevo modelo puede obligar a modificaciones en el Modelo Esencial, en ese caso, Yourdon [Yourdon 89] recomienda el mantenimiento de una versión intacta del modelo esencial original. Esto permitirá la experimentación de modelos alternativos del Modelo de Implementación del Usuario.

El modelo de Implementación del usuario, en términos generales, incluye los siguientes cuatro aspectos:

- ✓ La ubicación dentro del Modelo Esencial de las personas versus las máquinas.
- ✓ Detalles de la interacción hombre/máquina.
- ✓ Actividades manuales suplementarias que pueden ser necesarias.
- ✓ Restricciones operacionales que el usuario quiere imponer al sistema.

## IV.2.1. Determinación de los Límites de la Automatización

¿Qué funciones y que datos serán manipulados manualmente, y cuáles automatizados? Aunque pueda haber tenido una versión preliminar y experimental de los límites de la automatización durante el estudio de viabilidad, ellos no son fijos ni definitivos. De hecho, el límite de la automatización es casi irrelevante en el modelo esencial, porque aunque las necesidades del usuario sean que nosotros desarrollemos un sistema automatizado, él también necesita una declaración bien elaborada de requisitos de las funciones y de los datos que estarán fuera de las fronteras de la automatización. Existen tres casos extremos que tienen que ser considerados:

- ✓ *Al usuario no le debe interesar donde está la frontera de la automatización:* es poco probable que esto ocurra. En la práctica el usuario elige el equipo de implementación. ¿Qué partes del sistema deben ser manuales y cuáles automatizadas? Además del hecho que el usuario normalmente tiene gran sensibilidad en ese problema, éste espera que el analista produzca un



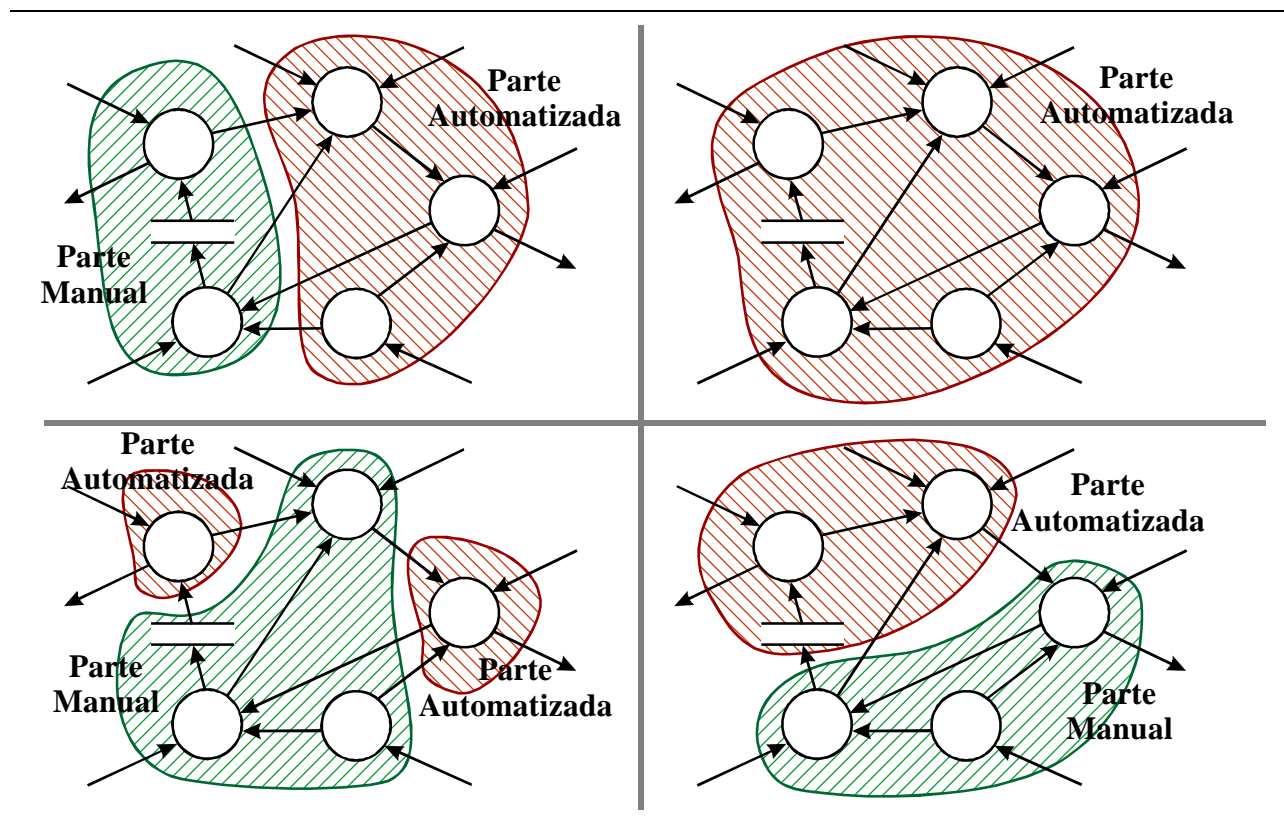


Fig. IV-1: Cuatro posibles definiciones de límites de automatización para el mismo sistema.

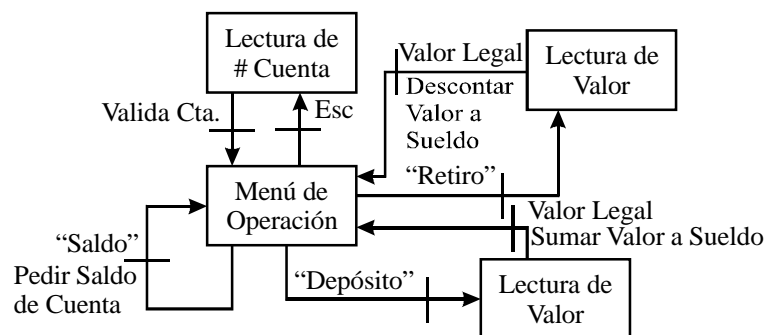
análisis detallado de costo-beneficio de todo el proyecto. Eso demanda alguna decisión preliminar acerca de que partes del modelo esencial serán automatizadas y cuales manuales.

- ✓ *El usuario puede optar por un sistema completamente automatizado:* Esta es una situación muy común, principalmente si el sistema esta desarrollándose para sustituir otro sistema existente. De esta manera, las actividades manuales ya pueden estar fuera del límite del sistema representado en el diagrama de contexto como agentes externos con los que el sistema se comunica.
- ✓ *El usuario puede optar por un sistema completamente manual:* Esta es una opción muy poco común, sobre todo en esta era de la automatización, dado que el analista habitualmente tiene interés en informatizar todo lo que sea posible. Sin embargo, esto puede pasar en momentos en que la intención del usuario no sea informatizar, pero si reorganizar la manera en que las actividades están ejecutándose.

La Fig. IV-1 presenta algunas posibles alternativas para establecer el límite de un sistema. Estas alternativas deben discutirse entre el usuario, el analista y el equipo de implementación, con base en un estudio del costo/beneficio.

## IV.2.2. Determinación de la Interfaz del Usuario

Hay varios estudios estadísticos de sistemas interactivos que indican que, entre el 40% y el 60% del código y datos de los programas están dedicados a la implementación de la interfaz hom-



**Fig. IV-2: Modelo de Interfaz de Operaciones en cuenta corriente.**

Interfaz para una transacción de consulta de Saldo de Cuenta, Incrementar Saldo y Descontar Valor a Saldo.

bre-máquina<sup>1</sup>. Este hecho pone en evidencia la importancia en costos de mantenimiento de un adecuado diseño de la interfaz hombre-máquina.

La especificación de la interfaz de usuario es, probablemente, la actividad que más tiempo consume y en la que el usuario más interesado está. Eso involucra cuatro problemas relacionados:

- ✓ La elección de dispositivos de entrada y salida (por ejemplo, las terminales alfanuméricas o gráficas, dispositivos de lectura óptica, impresoras de gran velocidad, etc.). Por ejemplo, en algunos casos, un requisito importante es la posibilidad de hacer análisis estadístico del desempeño de las funciones de algunas de las áreas de organización (como volúmenes de venta de ciertos productos relacionados al periodo anual, o los volúmenes de ventas generales relacionados al periodo mensual, etc.), o proyecciones financieras respecto a mercaderías en stock. En esos casos, la elección de una terminal gráfica para la gerencia (de ventas, de stock o financiera) puede ser un requisito fundamental. También, si se identifican volúmenes muy elevados de facturación, una impresora de gran velocidad puede ser muy importante.
- ✓ El formato de las entradas que fluyen de los agentes externos hacia el sistema. Es preciso especificar con gran detalle las características de los datos ingresados y como ellos serán aceptados.
- ✓ El formato de las salidas que fluyen del sistema hacia los agentes externos. ¿Se representan ellos en la pantalla de la computadora o en informes impresos? Comúnmente el formato de los informes impresos ya es predeterminado y, es común que el usuario quiera mantener estos formatos. Si el agente externo es gubernamental, habitualmente los informes que se desean tienen un formato muy estricto.
- ✓ La secuencia y el escalonamiento de entradas y salidas en el componente en línea. Estos escalonamientos deben diseñarse con mucho cuidado. Ellos pueden especificarse con diagramas de transición de estados.

El modelo de la interfaz hombre-máquina involucra dos problemas. El primero es el componente estructural, la secuencia y el escalonamiento de entradas y salidas. Ese problema puede plantearse con diagramas de transición de estados, las transacciones del sistema se ejecutan en las transiciones entre los estados. El segundo problema es el formato y la técnica de interacción para los flujos de datos de entrada, y el formato de presentación de los flujos de datos de salida de las transacciones. En la Fig. IV-2 se puede ver un ejemplo.

<sup>1</sup> Bobrow, en su artículo "Expert Systems: Perils and Promise", CACM Vol 23, N° 9 Sep 86, presenta un análisis de aplicaciones en inteligencia artificial de donde se desprenden los resultados citados. Existen también otros estudios que confirman esto, por ejemplo un estudio de Sutton en 1978 (Pedro Szekely, "Separating the User Interfaz from the Functionality of Application Programs", CMU-CS-88-101, Enero de 1988) sobre 16 aplicaciones comerciales el 59% del código estuvo dedicada a la interfaz.

### IV.2.3. Identificación de Actividades Manuales Complementarias

En el modelo esencial, presumimos la existencia de una tecnología perfecta. Entre otras cosas consideramos que nuestra tecnología de implementación nunca falla ni comete errores. Así mismo con el funcionamiento de la tecnología perfecta, los usuarios cometen errores de operación que, si no fueron considerados apropiadamente, producen daños en el sistema. El usuario puede estar trabajando con datos financieros, caso en el que pueden existir exigencias legales (impuestas por interventores) para asegurar la integridad de las entradas y salidas de los archivos del sistema. En la mayoría de los casos estas actividades complementarias de apoyo serán representadas por nuevos procesos en DFDs del modelo comportamental. La Fig. IV-3 presenta un ejemplo.

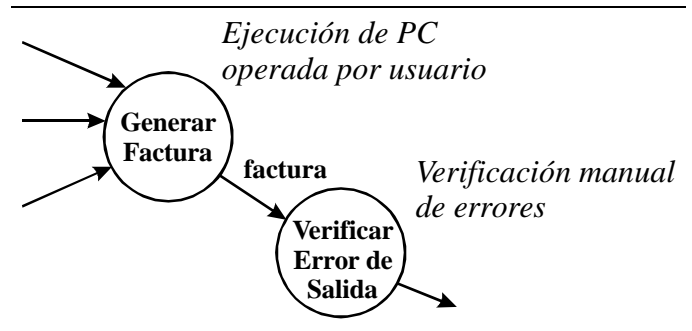


Fig. IV-3: Una actividad de apoyo manual

De una manera general, tendremos que contemplar la posibilidad de fallas tecnológicas en:

- ✓ *Las entradas y salidas del sistema:* Si los datos de entrada han sido introducidos por dispositivos de vídeo unidos a las computadoras principales por líneas de telecomunicación, la posibilidad de errores en la transmisión es muy grande. También puede tener errores de los medios magnéticos como diskettes (o discos blandos).
- ✓ *La ejecución de cálculos:* una vez introducidos los datos en el sistema, los errores del hardware pueden existir (memoria, procesador, etc.) o de programa.
- ✓ *El almacenamiento de datos por periodos largos:* la mayoría de los sistemas retienen datos en discos por periodos largos de tiempo. Es posible que, esos datos, sean destruidos debido a los errores del hardware (o software).

Lo que debe hacerse con respecto a estas posibles áreas de tecnología defectuosa dependerá mucho de:

- ✓ El nivel estimado de fiabilidad del hardware y del software.
- ✓ La naturaleza y la fiabilidad del usuario.
- ✓ Los costos o multas asociadas a entradas y/o salidas defectuosas.

Por la naturaleza de la incertidumbre en la predicción de estos tipos de fallas, la solución más común es la introducción de redundancias: entradas y/o salidas redundantes, procesadores redundantes, la redundancia interna en los depósitos de datos, campos redundantes (de control) en las estructuras de datos.

### IV.2.4. Especificación de Restricciones Operacionales

El equipo de implementación tendrá que decidir que combinación de hardware, sistema operativo, recursos de comunicación, lenguajes de programación, administradores de base de datos y estrategias de proyecto llevarán a cabo mejor los requisitos. Pero, será difícil hacerlo sin el establecimiento de algunas restricciones de operaciones (que el modelo esencial evita deliberadamente). Los problemas principales, normalmente, son:

- ✓ *Volúmenes de datos:* Necesitamos que el usuario especifique los volúmenes de transacciones de entrada y el tamaño necesario de los depósitos de datos, si el volumen es estable o hay grandes variaciones.
- ✓ *Tiempo de respuesta para las entradas:* Las restricciones de tiempos de respuesta pueden existir para algunas entradas del sistema. Por ejemplo, todas las transacciones de depósitos de

clientes deben procesarse por la noche para que de mañana los clientes puedan saber su saldo, o las novedades para los empleados se admiten hasta antes de las 10 horas de la mañana, o la liquidación de haberes que deba estar hecha a las 8 horas del día siguiente.

- ✓ *Restricciones políticas en opciones de implementación:* El usuario puede, por motivos racionales o irracionales, especificar la marca del hardware a ser usado (o a ser evitado), el vendedor del equipo, y así sucesivamente. O, cuando ya fue comprado el hardware y el problema es que éste no es el que mejor se adapta al problema. O al revés, el problema puede ser respecto al hardware ya existente
- ✓ *Restricciones de seguridad y fiabilidad:* El usuario puede especificar el tiempo medio entre fallas y el tiempo medio entre paradas para el mantenimiento del sistema.
- ✓ *Restricciones de seguridad:* El usuario puede especificar la intención de minimizar el uso no autorizado del sistema. Eso puede incluir a usuarios administradores del sistema con claves de acceso. Los usuarios pueden tener el acceso a diferentes niveles (para los diferentes niveles de confidencialidad de información o de riesgo del funcionamiento).

## IV.3. Diagrama de Estructura

Los *diagramas de estructura* (DE) sirven para el modelamiento *top-down* de la estructura de control de un programa descrito a través de un árbol de *invocación de módulos*. Fueron presentados en la década de los 70 como la principal herramienta utilizada en diseños estructurados, por autores como Constantine, Myers, Stevens e Yourdon [Constant 74; Yourdon 78; Myers 78; Stevens 81]. La Fig. IV-4 muestra un ejemplo.

Un diagrama de estructura permite modelar un programa como una jerarquía de módulos. Cada nivel de la jerarquía representa una descomposición más detallada del módulo del nivel superior. La notación usada se compone básicamente de tres símbolos:

- ✓ Módulos
- ✓ Invocaciones
- ✓ Cuplas

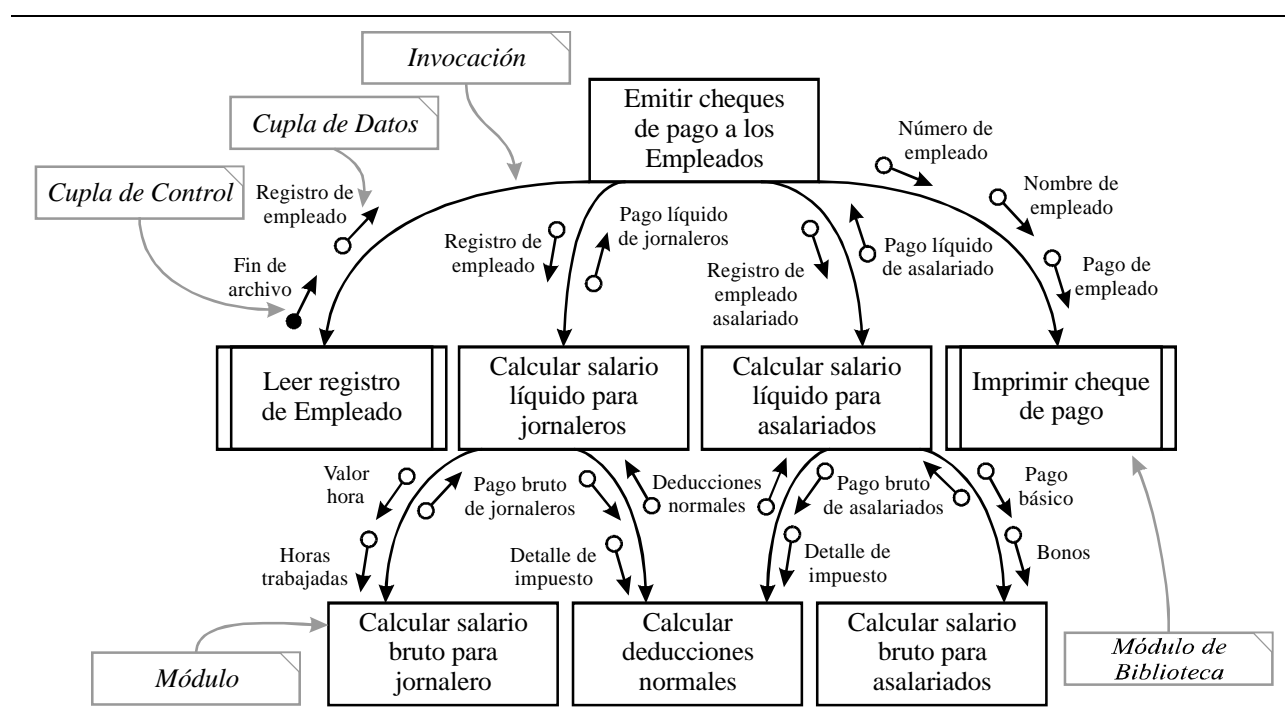


Fig. IV-4: Ejemplo de Diagrama de Estructura

### IV.3.1. Módulos

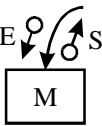
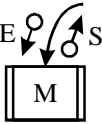
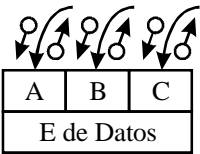
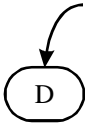
Un módulo es un conjunto de instrucciones que ejecutan alguna actividad, un procedimiento o función en PASCAL, una función en C o un párrafo en COBOL. Tal vez, la definición más precisa es que un módulo es una *caja negra*, pero como será mostrado a continuación son *cajas “casi” negras o grises*.

Desde un punto de vista práctico, un módulo es una colección de instrucciones de un programa con cuatro características básicas:

- ✓ *Entradas y Salidas*: lo que un módulo recibe en una invocación o retorna como resultado.
- ✓ *Función*: las actividades que un módulo hace con la entrada para producir la salida.
- ✓ *Lógica Interna*: por la cual se ejecuta la función.
- ✓ *Estado Interno*: su área de datos privada, datos para los cuales sólo el módulo hace referencia.

Las entradas y salidas son, respectivamente, datos que un módulo necesita y produce. Una función es la actividad que hace un módulo para producir la salida. Entradas, salidas y funciones proveen una visión externa del módulo. La lógica interna son los algoritmos que ejecutan una función, esto es, junto con su estado interno representan la visión interna del módulo.

Un módulo es diseñado como una caja, su función es representada por un nombre en el interior y las entradas y salidas de un módulo son representadas por pequeñas flechas que entran y salen del módulo. Existen distinciones entre módulos, la tabla siguiente describe los diferentes tipos de módulos:

<b>Módulo</b>		Un módulo es una <i>caja negra</i> conteniendo una colección de instrucciones y áreas de datos locales. Los módulos tienen una <i>función</i> , definida por el nombre contenido en el interior ( <b>M</b> ), <i>datos de entrada</i> y <i>datos de salida</i> generados por la aplicación de la función a los datos de entrada.
<b>Módulo de Biblioteca</b>		Un módulo predefinido o <i>módulo de biblioteca</i> representa un módulo que no tiene que ser especificado porque ya existe en el sistema o en bibliotecas. Frecuentemente, también, se utiliza la misma notación para representar llamadas al sistema operacional como <b>read</b> o <b>write</b> de archivos. Un módulo de biblioteca es verdaderamente una caja negra ya que no son modeladas las actividades que realiza.
<b>Cluster de Información</b>		Un <i>cluster de información</i> es una colección de módulos que trabajan sobre la misma estructura de datos. Son utilizados para modelar tipos abstractos de datos, esto es, encapsulan una estructura de datos y los módulos que la crean y referencian. En la parte inferior del módulo se anota el nombre de la estructura de datos.
<b>Área de Datos Global</b>		Es fuertemente recomendable la especificación de <i>áreas de datos</i> globales cuando son utilizadas por los módulos en un diseño. El uso de áreas globales no es una práctica confiable, en muchos diseños no pueden ser evitadas o precisan ser incluidas para mejorar el desempeño del sistema. Una buena especificación de estas áreas permite un mejor análisis de la calidad del diseño y disminuye costos de mantenimiento.

### IV.3.2. Relaciones entre Módulos (Invocaciones)

En la realidad, los módulos no son realmente cajas negras. Los diagramas de estructura muestran las invocaciones que un módulo hace a otros módulos. Estas invocaciones son diseñadas como una flecha que sale del módulo llamador y apunta al módulo llamado. La Fig. IV-5 muestra un ejemplo.

En el ejemplo de la Fig. IV-5 el módulo **A** invoca (o llama) a los módulos **B**, **C** y **D**. La interpretación de las invocaciones provee información de la estructura interna del módulo llamador, que no concuerda con la idea de caja negra. Una caja negra no permite que se observe su interior y, las invocaciones que un módulo hace son componentes de su estructura interna. De todas formas, se dice que un módulo es una *caja casi negra* o *caja gris* porque permite que se observen sólo las invocaciones.

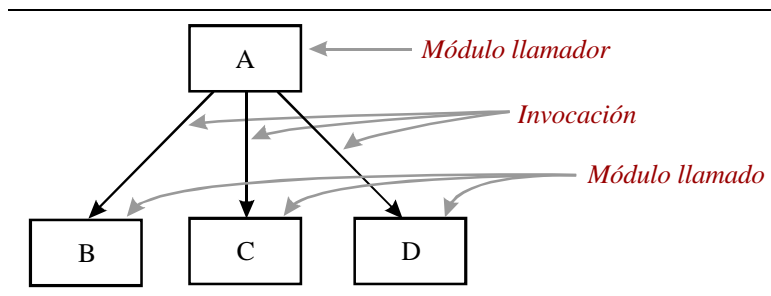


Fig. IV-5: Ejemplo de Invocación

Los diagramas de estructura no tienen especificado el orden de invocación de los módulos invocados. El orden de dibujo de los módulos **B**, **C**, y **D** (de izquierda a derecha) no debe ser interpretado como el orden de invocaciones ejecutado por el módulo **A**. Ese orden puede ser cambiado, al dibujar, para evitar que se crucen flechas o se dupliquen módulos, como ocurre con el módulo *Calcular Deducciones Normales* en la figura 1. A pesar que el orden de invocación de los módulos del mismo nivel en un diagrama de estructura, no es especificado por el formalismo, se recomienda que siempre que fuese posible, se siga un orden de izquierda a derecha (si esto no produce que se crucen flechas) que se corresponde con el orden de invocación, y permitiendo un orden de lectura que es patrón en la mayoría de los idiomas.

Una invocación representa la idea de invocación a funciones o procedimientos en los lenguajes de programación convencionales. En un diagrama de estructura se pueden modelar varios tipos de invocaciones:

Tipos de Invocación		
<b>Invocación Estándar</b>	<pre>graph TD; A[A] --&gt; B[B]</pre>	El módulo A invoca al módulo B con la semántica de invocación de procedimientos o funciones en los lenguajes de programación convencionales (C, Pascal, etc.).
<b>Invocación Concurrente</b>	<pre>graph TD; A[A] -.-&gt; B[B]; A -.-&gt; C[C]</pre>	El módulo A comienza dos tareas concurrentes, B y C.
<b>Invocación Co-rutina</b>	<pre>graph LR; A[A] --&gt; B[B]</pre>	El módulo A invoca al módulo B con una semántica de transferencia de control al punto de entrada de una co-rutina.
<b>Transferencia de Control Patológica</b>	<pre>graph TD; A[A] --&gt; B[B]</pre>	El módulo A hace una transferencia de control (ej.: con un GOTO o JUMP en ASSEMBLER) al interior del módulo B. Claramente es una práctica patológica y no es recomendable en ningún caso. Pero, si una patología existe es preferible que sea mostrada.
<b>Referencia Patológica</b>	<pre>graph TD; A[A] --&gt; B[B]</pre>	El módulo A hace referencia a un área de datos local del módulo B. También es una práctica patológica y debe ser evitada.

### IV.3.3. Comunicación entre Módulos (Cuplas)

Cuando una función o un procedimiento, en un lenguaje convencional, es invocado, comúnmente un conjunto de argumentos es comunicado y, en el caso de las funciones, también se espera que retorne un resultado. Estos datos comunicados en una invocación son modelados por medio de flechas, sobre el símbolo de invocación, llamadas *cuplas*.

Como se ve en la Fig. IV-6, existen varios tipos de cuplas, basado en lo que ellas pueden producir en el módulo receptor, las cuales se describen a continuación.

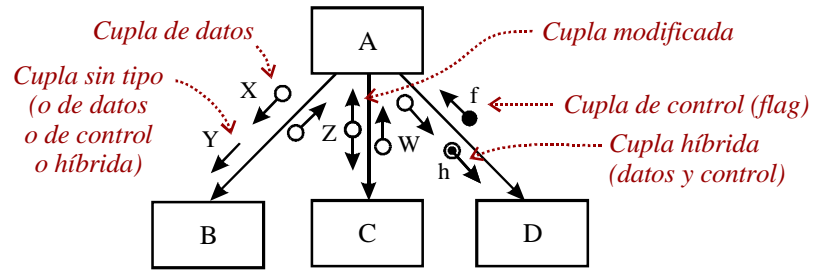


Fig. IV-6: Ejemplo de invocación con cuplas

Tipos de Cuplas		
<b>Cupla de Datos</b>	♀	Una <i>cupla de datos</i> transporta datos “puros” a un módulo. No es necesario conocer la lógica interna del módulo receptor, para determinar los valores válidos de la cupla (ej.: número de cuenta, saldo, tabla de movimientos).
<b>Cupla Modificada</b>	↕	Con una flecha doble (apuntando al modulo llamador y al módulo llamado) se especifica un argumento enviado a un módulo que deberá modificar su valor, fijando un nuevo valor disponible para el módulo llamador (en la implementación, se precisará que el lenguaje posea un mecanismo de pasaje de parámetros por referencia) (ej.: el buffer enviado a un módulo de lectura de un archivo).
<b>Cupla de Control</b>	●	Una <i>cupla de control</i> transporta un dato indispensable para la toma de una decisión, en la lógica interna del módulo invocado (ej. código de operación).
<b>Cupla Híbrida</b>	♀	Una cupla que para algunos módulos transporta datos “puros” y para otros transporta un <i>flag</i> de control es denominada <i>cupla híbrida</i> . Esta cupla representa un híbrido entre una cupla de datos y una cupla de control. La identificación de ellas depende de la lógica definida por el algoritmo de implementación del módulo invocado.
<b>Cupla sin Tipo</b>	↓	Se utiliza para representar que hasta el momento no esta definida o se tienen dudas con respecto a la interpretación que el módulo invocado hace de ella.

### IV.3.4. Absorción

En muchos casos en el diseño top-down de un programa, la descomposición realizada con el objetivo de mejorar la interpretación y modelamiento, da como resultado pequeños módulos (apenas una o dos instrucciones o una ecuación matemática). Esta descomposición puede ayudar en la comprensión del programa diseñado pero se tiene la certeza que en la implementación el módulo correspondiente no será separado. Este hecho puede ser especificado en el diagrama de estructura con el símbolo mostrado en la Fig. IV-7.

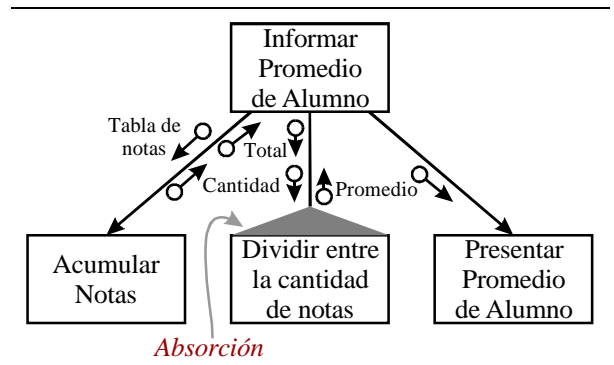

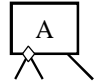


Fig. IV-7

### IV.3.5. Estructuras de Control

Los diagramas de estructura también definen símbolos que posibilitan el modelamiento de estructuras de control como repetición o alternativa.

<b>Repetición</b>		Una flecha circular, abarcando un número de invocaciones, especifica que las invocaciones que abarca son ejecutadas repetidamente.
<b>Alternativa</b>		Un rombo incluyendo una o más invocaciones especifica la ejecución condicional de ellas.

## IV.4. Derivación de los Diagramas de Estructura

Para un DFD dado siempre puede ser construido un diagrama de estructura que describe la arquitectura de módulos que debe ser desarrollada para implementar la funcionalidad descrita. La creación de los DE puede resultar una actividad muy difícil, principalmente para DFDs con muchos procesos.

Sin embargo, el diseño estructurado ofrece dos estrategias para construir rápidamente una primera versión del diseño. Las estrategias para la validación y el refinamiento (que serán presentados más adelante) pueden llevarnos en dirección a obtener un buen diseño. Sin tales estrategias, se tendría, frecuentemente, que efectuar sucesivas modificaciones a los diagramas de estructura y esperar que la inspiración celestial nos guíe hacia algo bueno.

Las dos estrategias de derivación de diagramas de estructura son:

- ✓ *Análisis de Transformaciones*: provee un vínculo entre el análisis estructurado y un diseño estructurado.
- ✓ *Análisis de Transacciones*: La estructura general de los sistemas tiende a caer en modelos reconocibles. Uno de los modelos comunes en sistemas comerciales es el llamado sistema de transacciones. El análisis de transacciones es una estrategia específica para diseñar estos tipos de sistemas.

### IV.4.1. Análisis de Transformaciones

El principal enfoque del análisis de transformaciones es convertir un DFD, resultado del análisis estructurado, en un diagrama de estructura. La principal ventaja de este enfoque es que, el diagrama de estructura resultante tiene una forma tal que permite un fácil desarrollo y mantenimiento más barato que la mayoría de los diagramas de estructura que podamos construir ad-hoc. Como será visto mas adelante, cuando los criterios de calidad son aplicados en los diagramas de estructura, el resultado es un DE donde, la mayoría de los módulos son independientes de los dispositivos de entrada y salida, esto es: describe el diseño de un sistema *balanceado*. La Fig. IV-8 describe los pasos realizados durante el análisis de transformaciones.

#### IV.4.1.1. Análisis de la Especificación del Problema

Si un diseño estructurado está siendo desarrollado, luego del análisis estructurado, entonces habrá un conjunto de DFDs que describen el problema, para los cuales se debe diseñar una solución. Si el análisis estructurado no precede al diseño, entonces se pueden reconocer dos situaciones muy diferentes:

- ✓ *Un problema muy simple*: Si el problema puede ser representado en la *memoria de una persona* [DeMarco 86], es muy simple y no precisa mayor esfuerzo que la especificación. En ese caso las herramientas del Análisis no son necesarias y el DE puede ser desarrollado ad-hoc. Sin embargo, el análisis estructurado ofrece una colección de técnicas y criterios que permiten analizar y especificar un problema cuando es muy complejo para ser comprendido y especificado con una simple declaración narrativa. Según DeMarco [DeMarco 86], un modelo es una *maqueta* de una realidad donde algunas características son estudiadas y, se construyen dife-



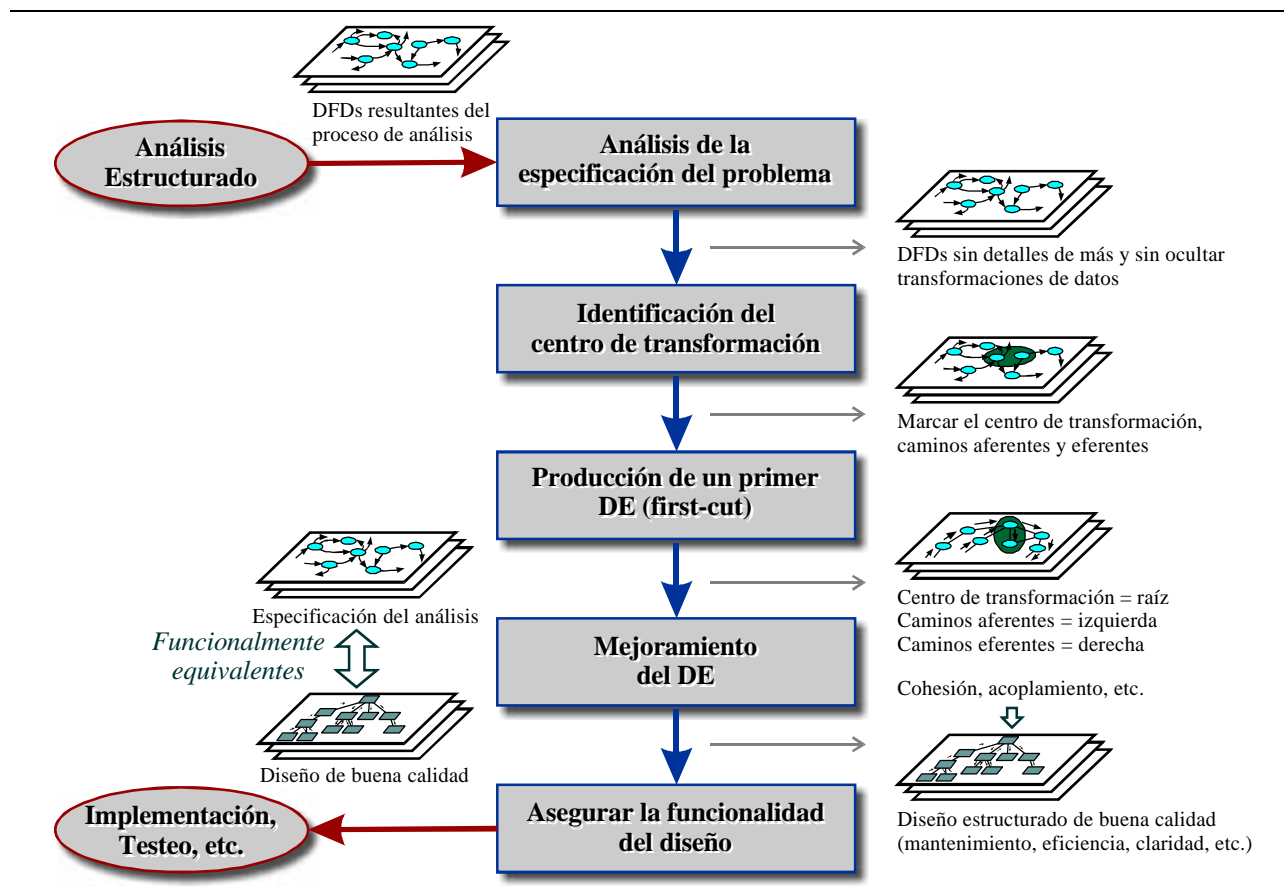


Fig. IV-8: Pasos del análisis de transformaciones

rentes modelos de la misma realidad (cada uno de ellos representando diferentes características) para estudiar diferentes partes del problema por separado.

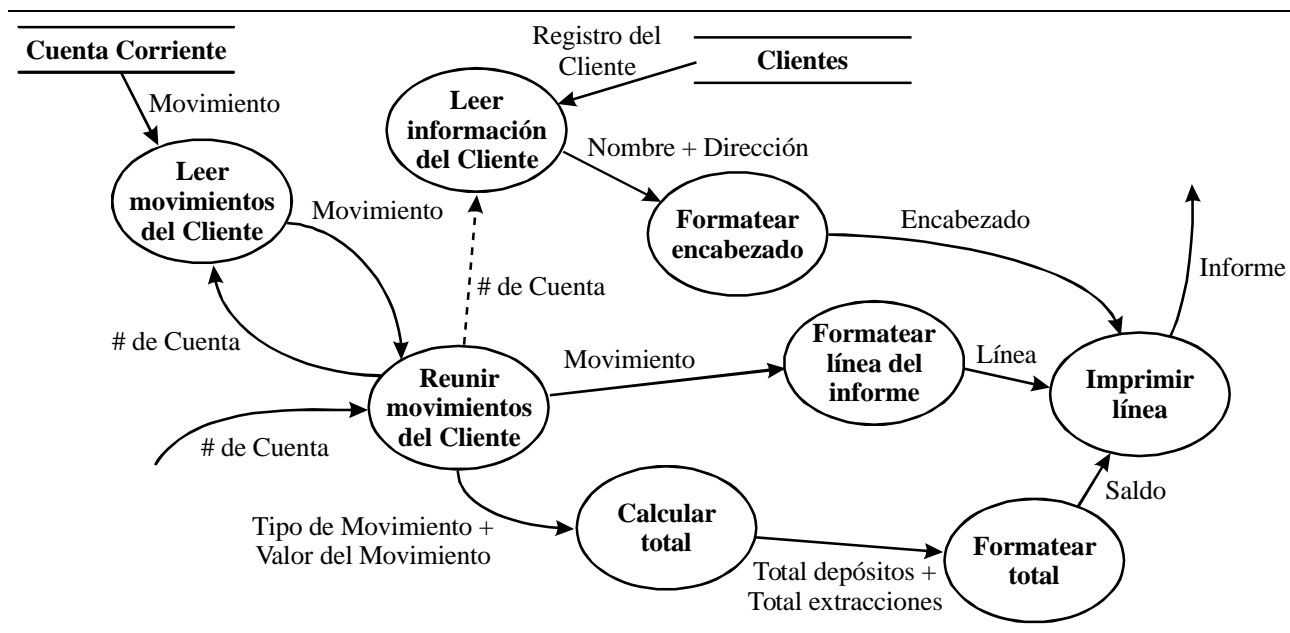
- ✓ *Un problema complejo:* La mayoría de las veces, el problema es mayor que la capacidad de nuestra memoria principal y diversos modelos deberán ser desarrollados, en el proceso de análisis estructurado, para conseguir una buena comprensión y especificación. En ese caso, será necesario construir algunos DFDs a partir de la narrativa del problema (declaraciones surgidas de las entrevistas con los diversos usuarios involucrados).

Para obtener un buen resultado con el análisis de transformaciones, los DFDs no deben llegar a un nivel de detalle en el que se tenga demasiada cantidad de procesos. Si un DFD es muy detallado puede ser necesario que se necesite hacer abstracción de algunos procesos (para reducir la cantidad). El DFD tampoco puede ser demasiado abstracto y ocultar algunas características que el análisis de transformaciones debe estudiar. Además, puede ser necesario descender un nivel (describiendo los flujos de datos y los procesos interiores de algunos procesos mostrados en el DFD) para que, el DFD presente todas las transformaciones de datos producidas para llevar los flujos de entrada en dirección de generar los flujos de salida.

#### IV.4.1.2. Identificar el Centro de Transformación

El centro de transformación es la parte del DFD que contiene la funcionalidad principal del sistema y es independiente de la implementación particular de las entradas y salidas. Por ejemplo, considere el DFD de la Fig. IV-9.

El diseñador podría considerar al proceso **Reunir Movimientos del Cliente** como la transformación principal del DFD, si un proceso tiene mas flujos de entrada que de salida, la pregunta que deberá ser respondida es: ¿Qué proceso hace el trabajo principal de la funcionalidad que representa el DFD?.



**Fig. IV-9: Generación de informe de movimientos de una cuenta corriente**

Claramente, el principal trabajo no es hecho por los procesos: Leer Movimiento del Cliente, Leer Información del Cliente, Calcular Total o Imprimir Línea. Tampoco es hecho por alguno de los procesos: Formatear Encabezado, Formatear Línea del Reporte o Formatear Total por separado. La función que el DFD modela, con certeza, no es Reunir Movimientos del Cliente, podría corresponderse con un proceso de abstracción mayor, tal vez llamado Generar Reporte de Movimientos, que incluya los procesos: Formatear Encabezado, Formatear Línea de Reporte y Formatear Total.

La elección del centro de transformaciones no es una actividad simple, generalmente requiere una interpretación detallada de la funcionalidad descrita por el DFD, y, en muchos casos, podría incluir mas de un proceso. Para eso existe una estrategia basada en la estructura del DFD, independiente de la interpretación particular de los procesos, que permite obtener una buena aproximación de la porción del DFD que debe incluir el centro de transformaciones.

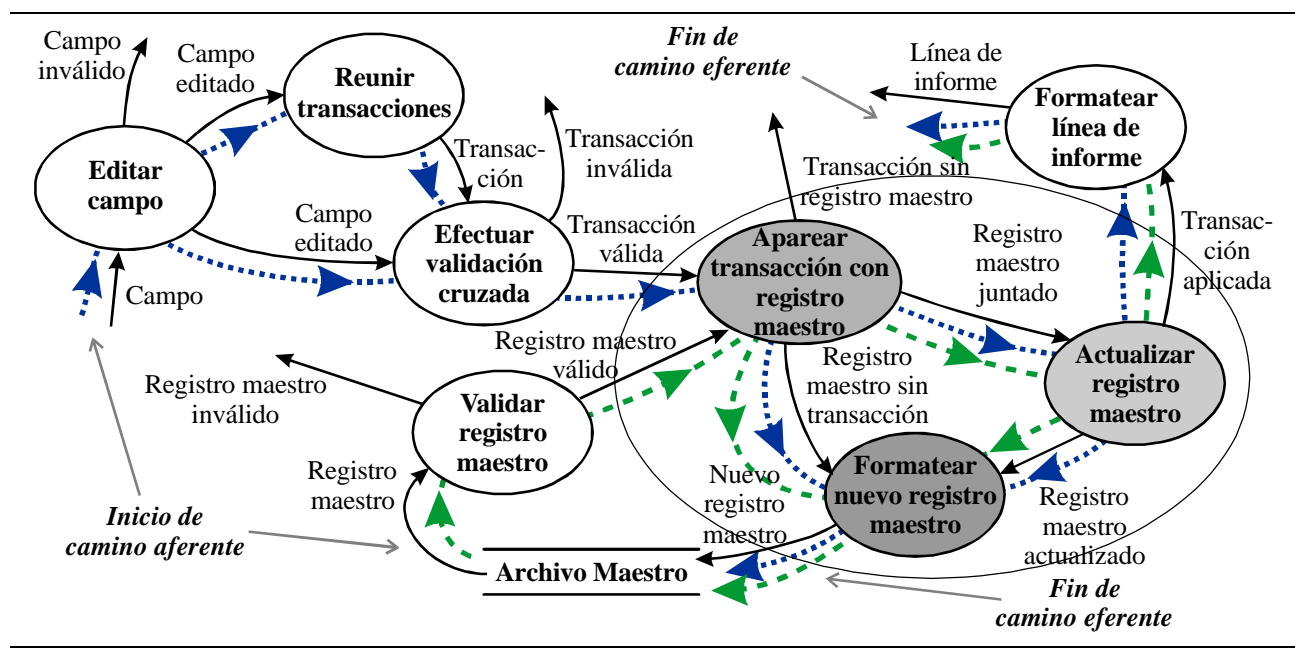
No es un algoritmo, ya que no establece una secuencia de etapas y tampoco garantiza la obtención acertada del centro de transformaciones. Una vez identificada la porción del DFD que incluye el centro de transformaciones, se debe interpretar detalladamente la función de los procesos incluidos para determinar si alguno de ellos representa la *transformación principal* del DFD o si una actividad de abstracción mayor deberá ser escogida.

#### IV.4.1.2.1. Estrategia para Determinar el Centro de Transformación

La estrategia intenta identificar la transformación central siguiendo los caminos de datos afe-rentes y eferentes. Este proceso puede ser desarrollado a través de los tres pasos siguientes:

1. Marcar cada camino aferente partiendo del lado externo del DFD (los flujos provenientes de depósitos de datos, agentes externos o porciones del DFD no incluidas en el Análisis<sup>2</sup>), y terminar en cada flujo eferente alcanzado (los flujos dirigidos para depósitos de datos, agentes externos o porciones de DFD no incluidas en el Análisis).
2. Diseñar una curva que una los puntos de intersección de caminos diferentes. Los procesos incluidos en el interior de la curva son candidatos iniciales para el centro de transformación.

<sup>2</sup> El DFD analizado es solamente una porción correspondiente a una transformación identificable. Como separar un DFD proveniente del Análisis en porciones correspondientes a transformaciones es una actividad muy intuitiva, quedará mas claro cuando sea presentado el Análisis de Transacciones.



**Fig. IV-10: Ejemplo de análisis de transformaciones**

Las líneas punteadas marcan los diferentes caminos de datos a través del DFD. Hay dos Caminos Aferentes que comienzan en los flujos *Campo* y *Registro Maestro* y dos Caminos Eferentes que finalizan en los flujos *Nuevo Registro Maestro* y *Línea de Informe*. Los procesos en el interior de la curva son candidatos a integrar el centro de transformaciones, ellos son **Aparear Transacción con Registro Maestro** y **Actualizar Registro Maestro**.

3. Analizar los límites del centro. Generalmente, en el límite se puede reconocer la finalización del refinamiento de las entradas (para los caminos de entrada o aferentes) y el comienzo del refinamiento de las salidas (para los caminos de salida o eferentes). Si no es así, modifique el contorno, yendo hacia el interior o exterior de forma tal que el interior, incluya solamente datos en estado lógico (independientes de las fuentes y destinos y totalmente refinados).

En el ejemplo de la Fig. IV-10 se ilustra la actividad descrita arriba. Primero se marcan todos los caminos de datos a través del DFD comenzando por los flujos de comienzo de los caminos aferentes y finalizando en los flujos de finalización de los caminos eferentes. En el ejemplo, los flujos de datos *Campo* y *Registro Maestro* son flujos de comienzo de caminos aferentes y, *Nuevo Registro Maestro* y *Línea de Informe* son flujos de finalización de caminos eferentes.

En el segundo paso se hace una curva uniando los puntos de intersección de caminos. La curva reúne los procesos candidatos para centros de transformación, en el ejemplo, reúne los procesos: **Aparear Transacción con Registro Maestro**, **Actualizar Registro Maestro** y **Formatear Nuevo Registro Maestro**.

La curva también indica la finalización de los caminos aferentes y el comienzo de los caminos eferentes, verificar eso es el objetivo del tercer paso.

La primera tarea es verificar que, en el interior de la curva, no haya transformaciones de refinamiento de los flujos de entrada o salida. En el ejemplo, el flujo de datos *Transacción Válida* es la versión mas refinada de los datos que comenzaron con el flujo *Campo* y, el proceso **Aparear Transacción con Registro Maestro** procesan datos de los dos caminos aferentes para crear una salida muy diferente (el *Registro Maestro Apareado*).

Con los caminos eferentes no ocurre la misma cosa. El proceso **Formatear Nuevo Registro Maestro**, aunque sea integrante del selecto grupo de procesos candidatos para centro de transformación, ejecuta una tarea de refinamiento de datos de salida. La tarea de transformación real de datos fue realizada por los procesos **Aparear Transacción con Registro Maestro** y **Actualizar Registro Maestro**. El módulo **Formatear Nuevo Registro Maestro** simplemente refina el *Registro Maestro Actualizado* o el *Registro Maestro sin Transacción* para generar el *Nuevo Registro Maes-*

tro. Así el proceso **Formatear Nuevo Registro Maestro** no compone el centro de transformación e incrementa el camino eferente.

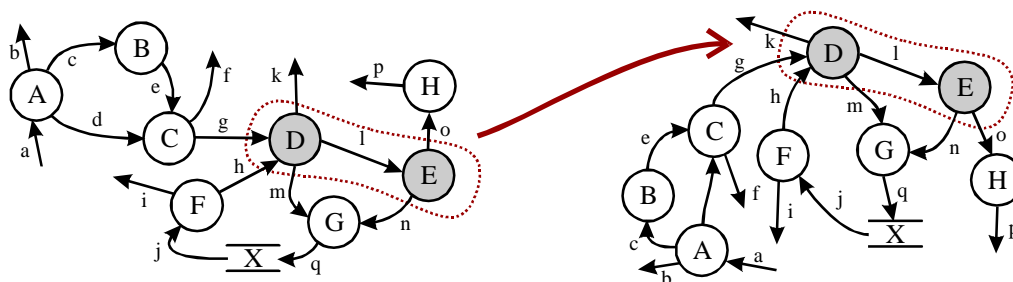
Como podemos ver, existe subjetividad en la elección de la transformación central, raramente surgen grandes acuerdos relativos a esa elección. El diseñador se podrá preguntar sobre un proceso aquí o allá, sin embargo, eso parece hacer poca diferencia en el diseño final. Por eso, si hubiera dudas sobre un proceso, ése no debe pertenecer al centro de transformación.

En sistemas de información el centro de transformación está compuesto por una pequeña porción del DFD y no incluye procesos de edición, formateo o verificación y corrección de errores.

#### IV.4.1.3. Producir un Primer Diagrama de Estructura (First-Cut)

Una vez reconocido el centro de transformación y los caminos aferentes y eferentes, una primera versión del DE puede ser desarrollada aplicando los cuatro pasos siguientes:

1. Convertir el DFD en una jerarquía de módulos: Tirar el DFD desde los procesos marcados como participantes del centro de transformaciones y dejar caer los otros procesos, por acción de la gravedad.



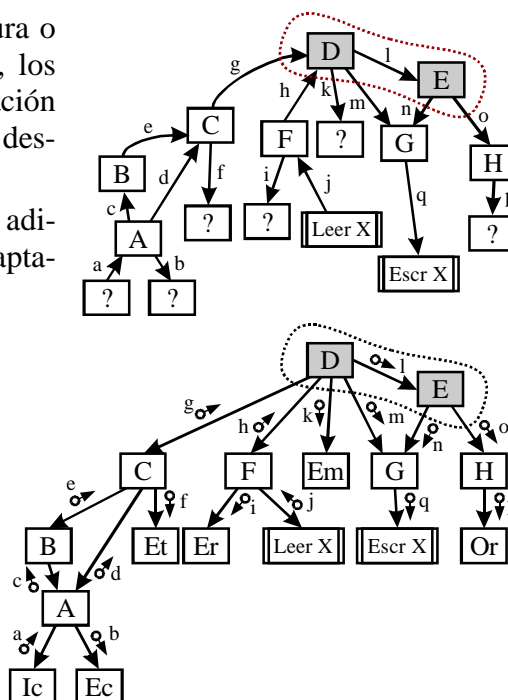
2. Sustituir los depósitos de datos por módulos de lectura o grabación (dependiendo de la orientación del flujo), los agentes externos por módulos de captación o presentación de datos y adicionar módulos debajo de los flujos sin destinatario u origen.

Se deben asociar nombres adecuados a los módulos adicionales, dependiendo de la actividad de lectura (captación) o escritura (presentación) que deben ejecutar.

3. Convertir los flujos de datos en invocaciones (apuntando al módulo invocado) y los datos transportados por los flujos en cuplas.

Cada uno de los módulos deberá ser analizado para determinar y adicionar los datos de entrada necesarios. Por ejemplo, el módulo **Leer X** debe recibir como entrada la clave de acceso para la lectura del registro.

4. Indicar un único módulo como raíz del DE, sea por selección de uno de los módulos participantes del centro de transformaciones o, por la incorporación de un módulo nuevo.



#### IV.4.1.4. Mejorar el Diagrama de Estructura Obtenido

El diagrama de estructura resultante del paso anterior, con certeza, puede ser mejorado. Eso simplemente es una primera aproximación y se debe beneficiar con la aplicación de los criterios de calidad (presentados mas adelante), especialmente Descomposición, Cohesión y Acoplamiento.

Por lo menos, la siguiente revisión debería ser hecha en el diagrama de estructura:

- ✓ Reorganizar, y descomponer si fuese necesario, los módulos aferentes y eferentes.
- ✓ Descomponer la transformación central, si fuese necesario, usando el DFD como base. Los niveles del DFD son útiles en este caso.
- ✓ Adicionar los módulos de manipulación de errores.
- ✓ Adicionar detalles de inicialización y terminación (si son requeridos).
- ✓ Tener certeza que todos los módulos tengan nombres correspondientes a su representación en la jerarquía.
- ✓ Adicionar todas las cuplas necesarias (de datos y de control).
- ✓ Chequear todos los criterios de calidad y mejorar el diseño de acuerdo con esos criterios.

De esta manera, aplicando los cuatro pasos en el DFD de la Fig. IV-10, el DE resultante es el que muestra la Fig. IV-11.

#### IV.4.1.5. Garantizar la Funcionalidad del Diseño

El paso final es el paso más importante: tener certeza que el DE es funcionalmente equivalente al DFD de origen. El propósito del análisis de transformaciones, es obtener rápidamente un DE que implemente correctamente la especificación del problema y cumpla los criterios de calidad.

#### IV.4.2. Análisis de Transacción

El análisis de transformaciones es la principal estrategia para convertir un DFD (de transformación de datos) en un DE. Sin embargo, una pregunta está sin responder: ¿que criterio puede ser aplicable para particionar un DFD mayor en un conjunto de DFDs de transformación?

Una técnica suplementaria, llamada análisis de transacción es extremadamente valiosa para

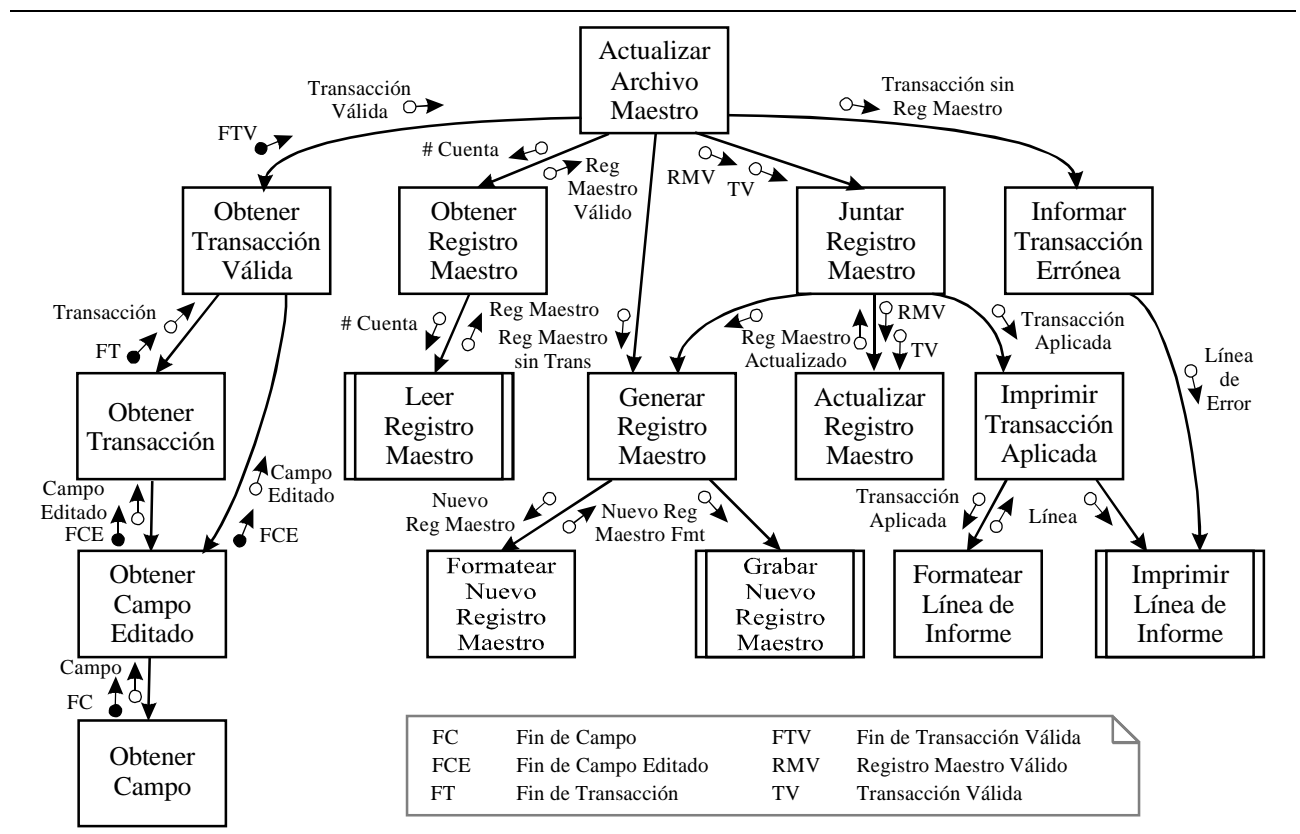


Fig. IV-11: Resultado de aplicar análisis de transformaciones en el DFD de la Fig. IV-10

dividir un DFD de alto grado de complejidad en DFDs de menor complejidad. Esta técnica divide en distintos DFDs, uno para cada transformación que el sistema procesa. Esos DFDs menores serán suficientemente simples como para permitir su conversión por medio del análisis de transformaciones en Diagramas de Estructura (DE). El análisis de transacción también puede ser usado para combinar los diagramas de estructura individuales (de transacciones separadas) en un diagrama de estructura mayor y más flexible.

Una transacción, en general, es un estímulo para un sistema que posee un conjunto de actividades a ser realizadas internamente. Ejemplos de transacciones son: incluir un nuevo cliente, generar una factura por venta de mercaderías, actualizar el stock de un producto, disminuir la temperatura de un reactor nuclear, actualizar archivo maestro o generar el reporte de movimientos de cuenta corriente.

Los DE que resultan del análisis de transacción tienen la forma descrita por la Fig. IV-12. De manera similar al análisis de transformaciones, en el análisis de transacción la actividad principal para derivar un DE a partir del DFD es identificar el centro de transacción. Frecuentemente, es muy fácil reconocer transacciones, centros de transacciones y procesos de transacción a través del formato del diagrama. Siempre que un flujo de datos entra en un proceso que determina su tipo y lo envía a un proceso relacionado con el tipo, se puede tener certeza de haber localizado un centro de transacciones.

El DFD para un centro de transacción de operaciones en cuenta corriente está representado en la Fig. IV-13.

El proceso **Iniciar Operación Deseada** contiene el centro de transacción el cual activa el

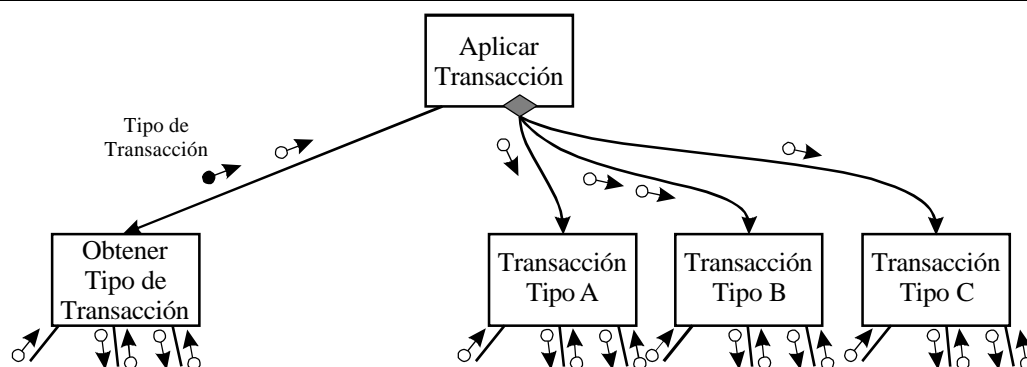


Fig. IV-12: La estructura típica de los DE generados por análisis de transacción

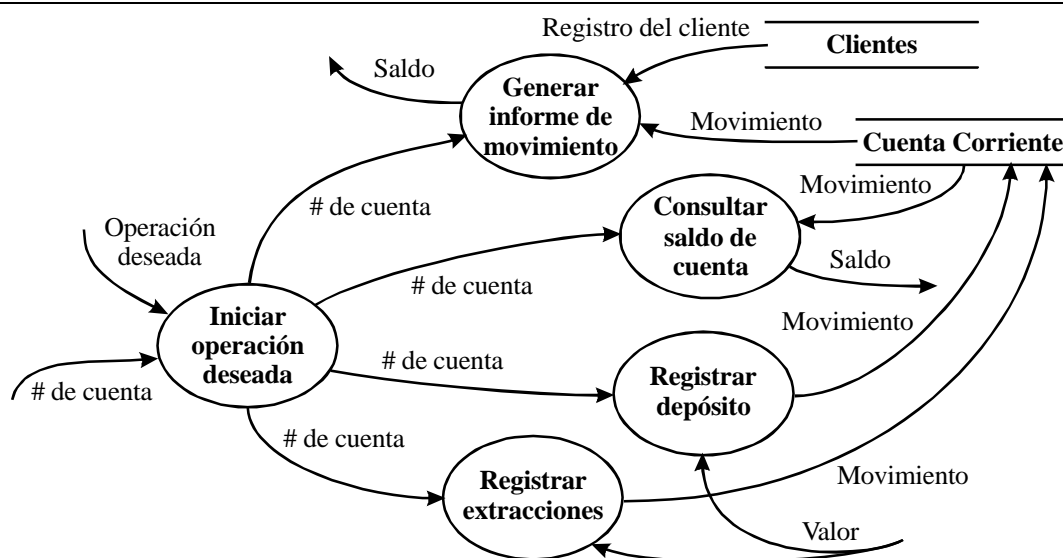


Fig. IV-13: Ejemplo de DFD de transacciones

proceso apropiado dependiendo de la *Operación Deseada*. Sin embargo, la manifestación del centro de transacción en un DFD es frecuentemente más sutil.

En el DFD de la Fig. IV-14, las diferentes transacciones son identificadas claramente pero, ¿dónde está el centro de transacción?. Una posibilidad es adicionar un proceso que reciba todos los flujos de entrada y determine la transacción adecuada pero, esa situación artificial complicaría innecesariamente el diseño y tornaría el sistema inflexible (ya que un único proceso debería preocuparse de todos los tipos de transacciones del sistema).

La solución más adecuada es incorporar un proceso de control que solamente reciba la información de control necesaria para determinar la transacción que tiene que ser ejecutada. En la realidad, un centro de transacción tiene la mayoría de las veces la funcionalidad de un proceso de control. Así, el DFD de la Fig. IV-14, con el centro de transacción incorporado, es mostrado en la Fig. IV-15.

El ejemplo de las transacciones bancarias de la Fig. IV-13 es un poco diferente. El centro de transacción **Iniciar Operación Deseada** no fue incluido artificialmente. Eso se muestra en el DFD, tal vez, por algún motivo de modelado y puede traer alguna otra funcionalidad diferente a la de control. Ese es un proceso normal que tiene el rol de control y además tiene la función de control; ese hecho, puede ser modelado de la forma mostrada en la Fig. IV-16.

Una vez identificado el centro de transacción, el DFD original resulta subdividido en un nú-

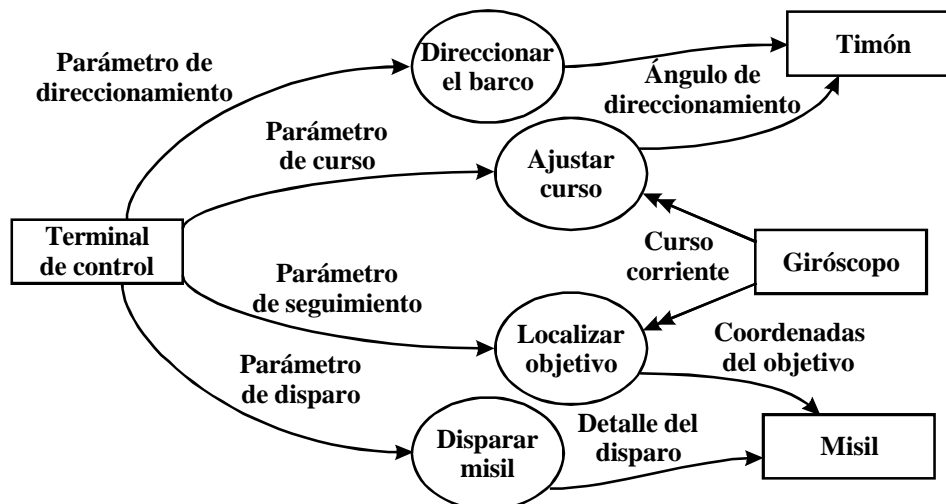


Fig. IV-14: Ejemplo de DFD de transacciones sin mostrar el centro

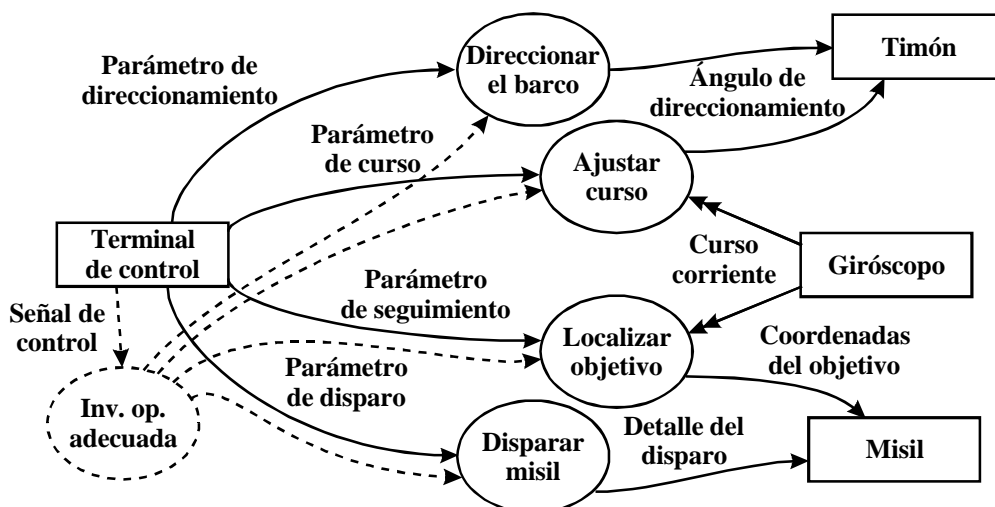


Fig. IV-15: Ejemplo de DFD de transacciones incorporando el centro

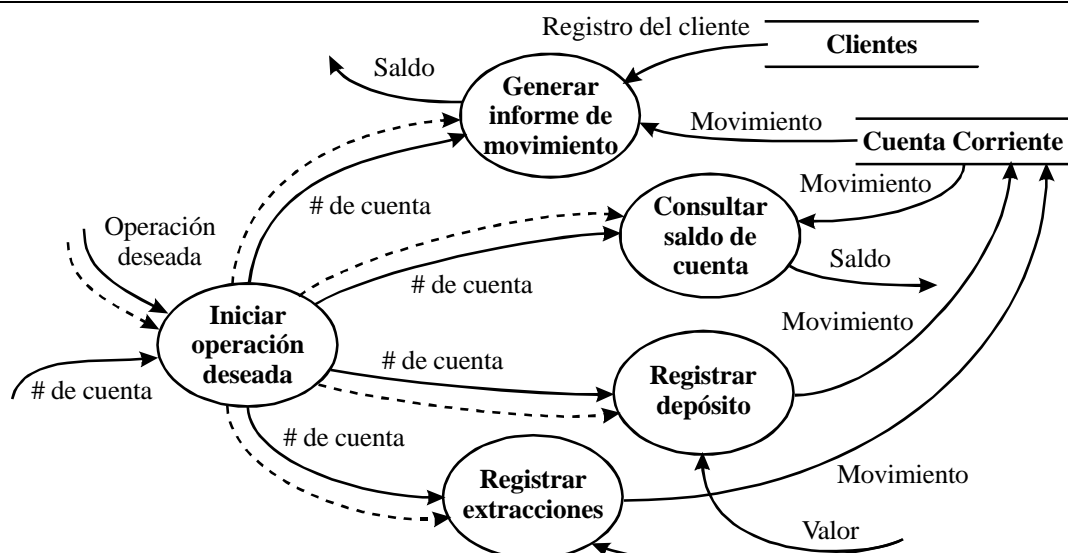


Fig. IV-16: Ejemplo de DFD de transacciones

mero de DFDs menores, uno por cada transacción, que pueden ser derivados por análisis de transformaciones o, nuevamente, por análisis de transacción. La Fig. IV-17 muestra el DE resultante para los ejemplos de la Fig. IV-15 y la Fig. IV-16.

El análisis de transacciones genera un *esqueleto* de diagrama de estructura que deberá ser unido (substituyendo las hojas) con los diagramas de estructura de cada una de las transformaciones identificadas.

## IV.5. Criterios de Validación de Calidad

Las estrategias de análisis de transformación y análisis de transacción nos permiten derivar

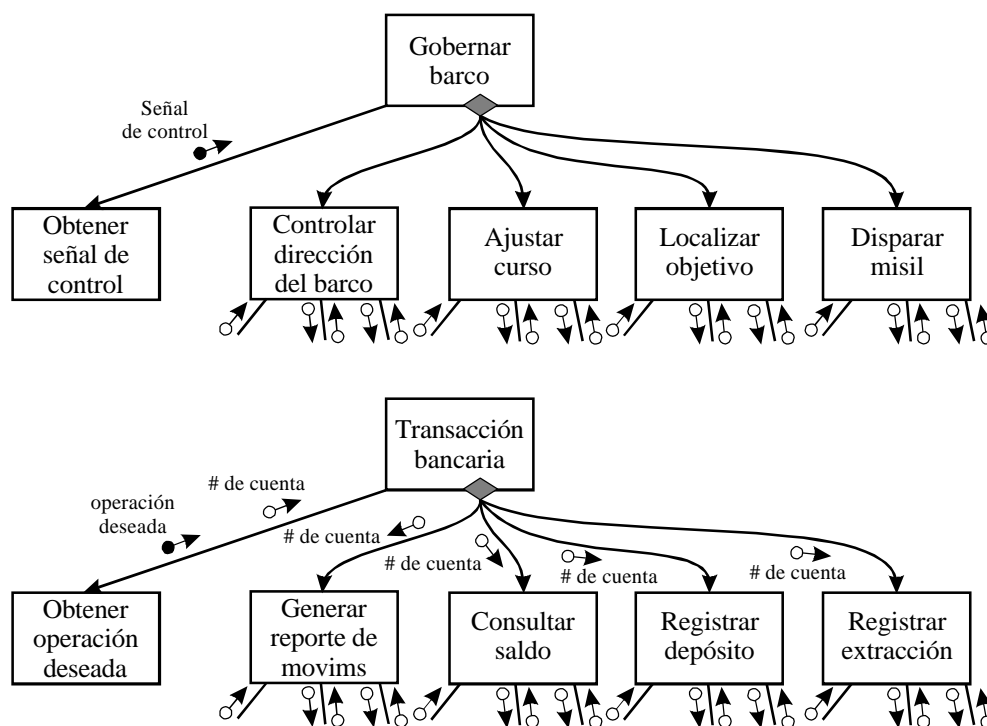


Fig. IV-17: DE para los ejemplos de la Fig. IV-15 y la Fig. IV-16



diagramas de estructura a partir de los DFDs del análisis, pero estos diagramas de estructura resultantes necesitan mejoras. Ni los diagramas de estructura ni las especificaciones de módulos por si solos nos dicen mucho con respecto a la calidad de un diseño.

Los diagramas de estructura son simplemente una herramienta para modelar los módulos de un sistema y sus relaciones y, junto con las especificaciones de funcionalidad de los módulos y las estructuras de datos de las cuplas, componen un diseño inicial que deberá ser analizado y mejorado.

Uno de los principios fundamentales del diseño estructurado es que un sistema grande debería particionarse en módulos mas simples. Sin embargo, es vital que esa partición sea hecha de tal manera que los módulos sean tan independientes como sea posible y que cada módulo ejecute una única función. Para que los diseños tengan esas cualidades, son necesarios algunos criterios de medición que permitan clasificar y mejorar los diagramas de estructura. A continuación se describen los criterios utilizados para mejorar un diseño.

## IV.5.1. Acoplamiento

El acoplamiento entre módulos clasifica el grado de independencia entre pares de módulos de un DE. El objetivo es minimizar el acoplamiento, es decir, maximizar la independencia entre módulos. A pesar que el acoplamiento, es un criterio que clasifica características de *una invocación* (una relación existente entre dos módulos), será usado para clasificar un DE completo. Un DE se caracteriza por el *peor* acoplamiento existente entre pares de sus módulos, ya que ese es el problema que debe ser resuelto para mejorar la calidad del DE completo.

Un bajo acoplamiento indica un sistema bien particionado y puede obtenerse de tres maneras:

- ✓ *Eliminando relaciones innecesarias*: Por ejemplo, un módulo puede recibir algunos datos, innecesarios para él, porque debe enviarlos para un módulo subordinado.
- ✓ *Reduciendo el número de relaciones necesarias*: Cuanto menos conexiones existan entre módulos, menor será la chance del efecto en cadena (un error en un módulo aparece como síntoma en otro)
- ✓ *Debilitando la dependencia de las relaciones necesarias*: Ningún módulo se tiene que preocupar por los detalles internos de implementación de cualquier otro. Lo único que tiene que conocer un módulo debe ser su función y las cuplas de entrada y salida (cajas negras).

A cada invocación de un DE le será atribuido uno de los cinco tipos posibles de acoplamiento: *de datos, estampado, de control, común o por contenido* (Fig. IV-18).

### IV.5.1.1. Acoplamiento sin Cuplas

Hay una categoría de acoplamiento no considerada en la tabla anterior por ser un caso poco frecuente, el *Acoplamiento sin Cuplas*. Un acoplamiento sin cuplas representa una invocación donde ningún argumento se comunica y el módulo llamado no retorna ningún valor.

El acoplamiento sin cuplas puede ser considerado el *punto cero* en una tabla de acoplamientos

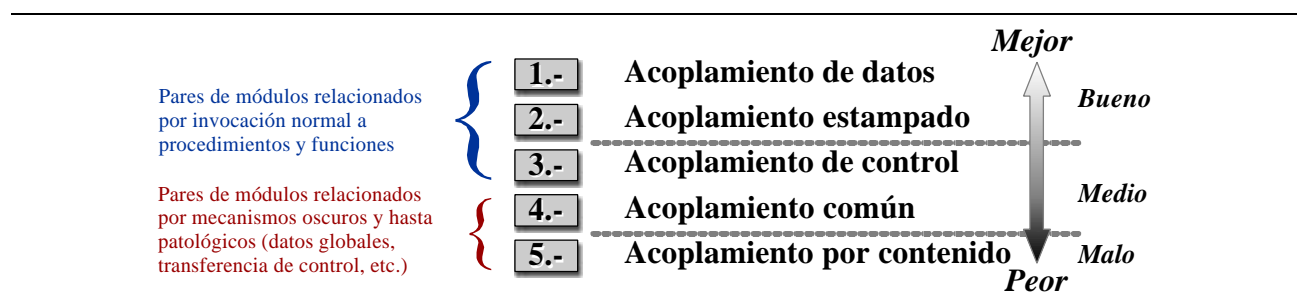


Fig. IV-18: Tipos de acoplamiento

siempre que, en la realidad, ningún dato sea comunicado. Si el módulo llamado hace referencia para áreas de memoria globales, podría corresponder a otro tipo de acoplamiento (típicamente *acoplamiento común* o *acoplamiento por contenido*).

#### IV.5.1.2. Acoplamiento de Datos

Dos módulos están acoplados por datos si todas las cuplas comunicadas en una invocación son *parámetros simples* (tipos de datos básicos en un lenguaje de programación: *integer*, *real*, *string*, etc.) o *tablas homogéneas*. Una tabla homogénea es una tabla en la cual cada entrada contiene el mismo tipo de dato y tiene la misma interpretación. La Fig. IV-19 muestra ejemplos de ambos tipos.

El acoplamiento de datos es una comunicación de datos necesaria entre módulos. Una vez que los módulos tienen que comunicarse, la vinculación de datos es inevitable e inofensiva (si es mínima). A pesar que es el tipo de acoplamiento recomendado, existen dos importantes riesgos:

- ✓ **Interfaz Compleja:** Cuando un módulo recibe un gran cantidad de cuplas (ej. 10 o 15), a pesar que todas sean simples o tablas homogéneas, la interfaz del módulo queda muy oscura. Así, las grandes ventajas de mantenimiento que el acoplamiento de datos ofrece, son de poca importancia para el gran esfuerzo que representa la comprensión. Una gran cantidad de cuplas, en general, es un síntoma de cohesión pobre. La solución es intentar una mejor descomposición de los módulos.
- ✓ **Datos Vagabundos (tramp data):** Cuando una cupla recorre una cantidad de módulos innecesariamente, se dice que es una cupla vagabunda. Los riesgos de mantenimiento, en presencia de cuplas vagabundas es casi semejante a los riesgos del acoplamiento común, ya que cualquiera de los módulos visitados puede producir modificaciones, por error, y los síntomas apuntarán para el módulo que la usa efectivamente. Las cuplas vagabundas son muy comunes, principalmente en DE derivados por análisis de transformaciones. Para corregir el defecto, se recomienda la reorganización del DE.

El acoplamiento de datos es el acoplamiento recomendado ya que: la interfaz es la mínima posible, la interfaz es clara y fácil de comprender, los síntomas de los errores aparecen en el mismo módulo donde se produce el error, la modificación de un módulo no produce *efecto en cascada* en

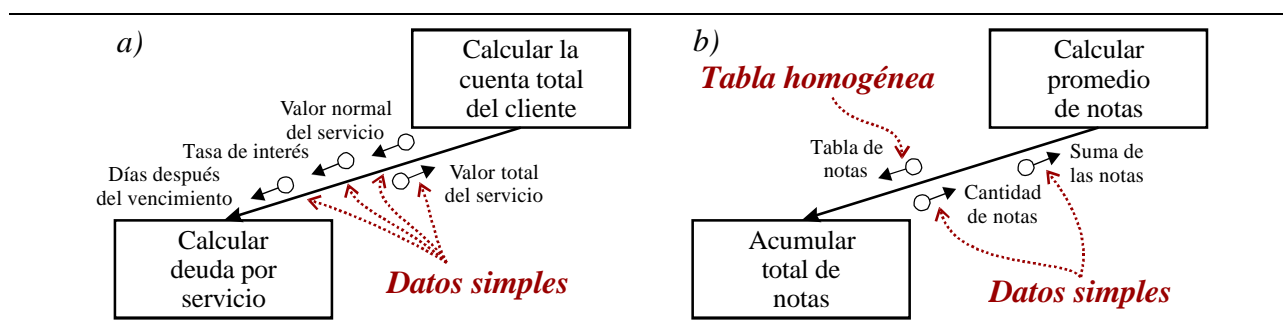


Fig. IV-19: Ejemplos de acoplamiento de datos

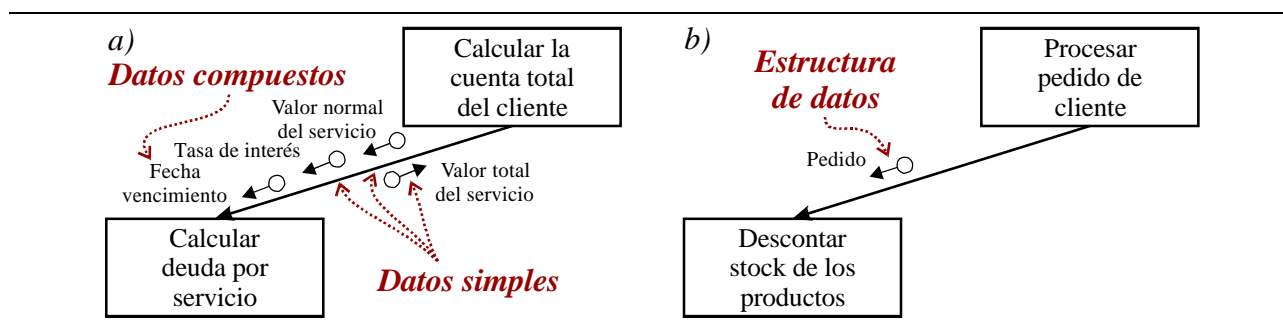


Fig. IV-20: Ejemplos de acoplamiento estampado

otros módulos, un módulo puede ser cambiado por otro fácilmente siempre que, el nuevo módulo tenga la misma funcionalidad y la misma interfaz (a pesar que sean implementados de manera muy diferente).

### IV.5.1.3. Acoplamiento Estampado

Dos módulos están vinculados por un acoplamiento estampado si, por lo menos una cupla contiene una estructura de datos (un grupo de ítems de datos simples o compuestos), o un dato simple interpretado como una estructura de datos, como en los ejemplos de la Fig. IV-20.

Por ejemplo, si en un programa PASCAL, un argumento es de tipo RECORD, o un argumento de tipo INTEGER que representa un dato empaquetado (4 bits para el día:  $2^4 = 16$ ; 5 bits para el mes:  $2^5 = 32$ ; y los 7 bits restantes para el año  $2^7 = 128$ , del año 1900 al 2028).

A pesar que el acoplamiento estampado no es el mejor, es un acoplamiento bueno si es bien usado. Por ejemplo considere la porción de DE de la Fig. IV-21.

Un tablero es una estructura de datos natural para el problema: no sería muy inteligente considerar cada cuadrado del tablero separadamente, porque eso haría que necesitaríamos de 64 cuplas para el tablero y una mas para el movimiento. Pero existen dos riesgos importantes en este tipo de acoplamiento:

- ✓ *Información innecesaria:* Cuando una estructura de datos es pasada a un módulo y, frecuentemente, no precisa de todos los ítems de datos de la estructura. Cuando el módulo usa apenas uno o dos ítems de un conjunto mayor, la interfaz queda oscura, compleja y poco flexible. Un ejemplo es el módulo **Descontar Stock de los Productos** mostrado anteriormente. Recibe la estructura de datos *Pedido* y solamente precisa de los identificadores de producto y las cantidades para descontar del stock.
- ✓ *Empaquetamiento artificial (bundling):* Considere el módulo de la Fig. IV-22. Para reducir la cantidad de cuplas de datos que recibe el módulo, se empaquetan en una estructura de datos llamada *Información para cálculo de costo*. Esta idea de agrupar datos, no relacionados razonablemente, en un estructura de datos artificial genera una interfaz oscura innecesariamente.

Los módulos acoplados por estructuras de datos tienen un acoplamiento mayor que los módulos relacionados solamente por datos simples. Si una estructura de datos debe ser modificada, todos los módulos que la reciben deberán ser modificados (o, por lo menos re-compilados). Así mismo, si una estructura de datos no es artificial (se basa en ideas de modelado) y el módulo precisa de todos los componentes, es un buen acoplamiento.

### IV.5.1.4. Acoplamiento de Control

Dos módulos están acoplados por control si uno pasa para el otro un dato (o grupo de datos) destinado a controlar la lógica interna del otro.

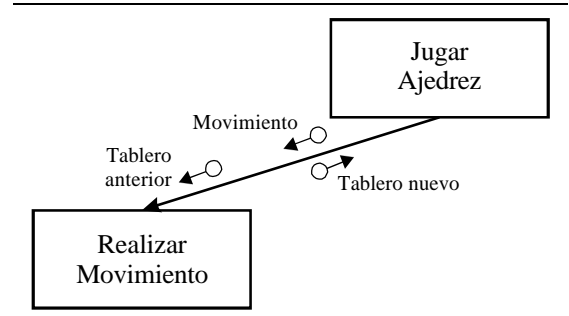


Fig. IV-21

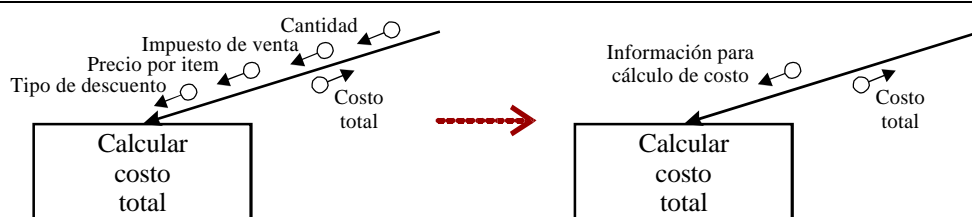


Fig. IV-22

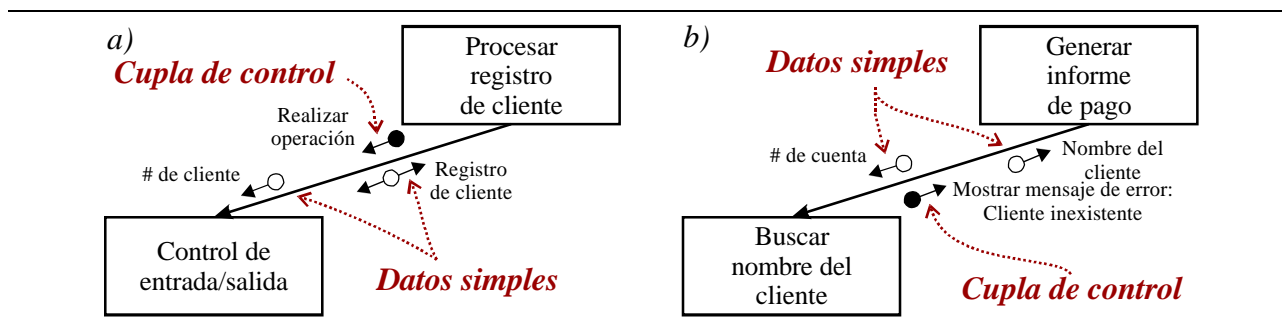


Fig. IV-23: Ejemplos de acoplamiento de control

En la Fig. IV-23-a, la cupla **Realizar Operación**, tiene como valor:

1. Leer siguiente registro de cliente.
2. Actualizar registro de cliente.
3. Borrar registro de cliente.

Así, el módulo **Procesar Registro de Cliente** explícitamente decide cual parte de la lógica interna del **Control de Entrada/Salida** se debe ejecutar. Para que el módulo llamador tome tal decisión, necesita saber como está organizada e implementada la lógica interna del módulo llamado.

En módulos acoplados por control, aquel que recibe la cupla de control no es una *caja negra*.

Peor aún es cuando una cupla de control se dirige hacia arriba, de un módulo subordinado a su jefe (como el acoplamiento entre **Generar Informe de Pagos** y **Buscar Nombre del Cliente**, en la Fig. IV-23-b). Esta situación es también síntoma de partición errónea y es llamada *Inversión de Autoridad*, que significa que un módulo subordinado da órdenes a su jefe. Esa partición errónea podría ser mejorada de dos maneras:

- ✓ El tratamiento de errores debería ser una función del módulo que lo reconoce. Así, el módulo **Buscar Nombre del Cliente** debería emitir el mensaje de error.
- ✓ El módulo **Buscar Nombre del Cliente** puede no saber realmente que ocurre el error. El módulo jefe debería ser capaz de corregir un error que no puede ser identificado por el módulo subordinado. En este caso, el módulo subordinado debería solamente reportar que el nombre no fue encontrado (Fig. IV-24).

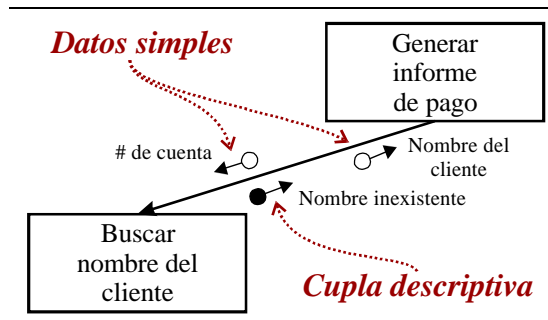


Fig. IV-24: Solución del acoplamiento de control de la Fig. IV-23-b

En la segunda alternativa, los dos módulos cumplen con las restricciones impuestas como *cajas negras* y, así, el acoplamiento deja de ser de control para convertirse en un acoplamiento de datos y, la cupla diseñada como una cupla de control es llamada *Cupla Descriptiva*.

#### IV.5.1.5. Acoplamiento Híbrido

Una cupla de control frecuentemente contiene valores discretos y fácilmente identificables, pero en algunos casos una cupla de datos cumple el rol de cupla de control. Por ejemplo, un código de cliente que tiene diferentes significados dependiendo del valor (de 1 a 50.000 es un cliente normal, de 50.001 a 60.000 es un cliente con bonificación de 10%, de 60.001 a 90.000 es un proveedor, etc.). El elemento de datos descripto, a pesar de no ser un flag, gobierna la lógica interna del módulo que el recibe. Si, en un futuro, tuviéramos más de 50.000 clientes sin bonificación (o mas de 30.000 proveedores), el sistema entraría en colapso.

Ese tipo de acoplamiento es llamado *Acoplamiento Híbrido* porque, la cupla es un híbrido entre cupla de datos y cupla de control.

#### IV.5.1.6. Acoplamiento Común

Dos módulos poseen acoplamiento común si hacen referencia a la misma área de datos global. En este caso, hablamos de una clase de acoplamiento existente entre dos módulos que no precisan estar relacionados por una invocación, como se puede ver en la Fig. IV-25. Tanto el acoplamiento común como el acoplamiento por contenido pueden aparecer entre dos módulos no relacionados en el DE. Así, son categorías de acoplamiento que no se aconsejan por las dificultades de comprensión y mantenimiento que generan. El acoplamiento común no es aconsejable por:

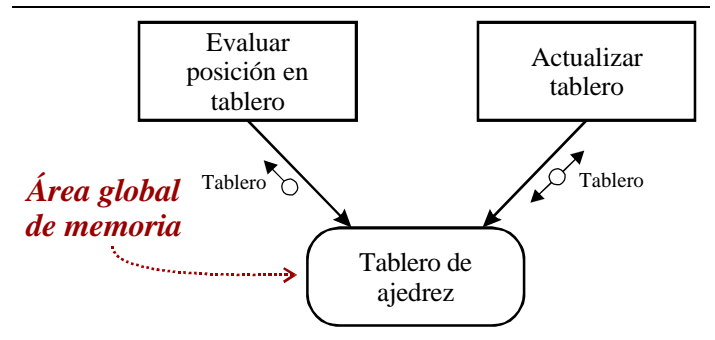


Fig. IV-25: Ejemplo de acoplamiento común

- ✓ *No localización de errores:* Un error en cualquier módulo que use un área global de datos puede aparecer en cualquier otro módulo que use aquella área.
- ✓ *Baja Reusabilidad:* Módulos que referencian datos globales generalmente lo hacen a través de nombres explícitos. Por ejemplo, un módulo que verifica la corrección de un dato almacenado en un área global, hace referencia a una fecha por un nombre explícito y queda sujeto al nombre, no puede verificar una fecha si no está almacenado en una área global con ese nombre. Tiene perdido una gran posibilidad de reutilización de ese módulo simplemente por la complejidad innecesaria de las convenciones de uso.
- ✓ *Conversión Implícita de Tipos:* La misma área global puede ser usada por módulos diferentes para almacenar datos de tipos diferentes.
- ✓ *Difícil Comprensión:* Programas con muchos datos globales son extremadamente difíciles de entender.
- ✓ *Poca Flexibilidad:* Cuando un módulo tiene que ser modificado es difícil descubrir qué datos precisan ser cambiados y, peor aún, si alguna área de datos global es modificada es muy difícil descubrir qué módulos deben ser modificados también.

#### IV.5.1.7. Acoplamiento por Contenido (o Patológico)

Dos módulos presentan acoplamiento por contenido (o patológico) si uno hace referencia al interior del otro: por ejemplo, si un módulo desvía la secuencia de control para adentro de otro o si un módulo altera un comando o un área de datos local del otro. El acoplamiento por contenido torna el concepto de módulo (como una caja negra) sin sentido, ya que obliga a un módulo a conocer el contenido e implementación de otro. El acoplamiento por contenido hace que los módulos sean tan dependientes entre ellos que es no posible modificar uno sin tener que modificar el otro.

### IV.5.2. Cohesión

Otro medio para evaluar la partición en módulos (además del acoplamiento) es observar como las actividades de un módulo están relacionadas unas con otras, este es el criterio de cohesión. Generalmente el tipo de cohesión de un módulo determina el nivel de acoplamiento que tendrá con otros módulos del sistema.

Cohesión es la medida de intensidad de asociación funcional de los elementos de un módulo. Por elemento debemos entender una instrucción, o un grupo de instrucciones o una llamada a otro módulo o, un conjunto de procedimientos o funciones empaquetados en el mismo módulo.

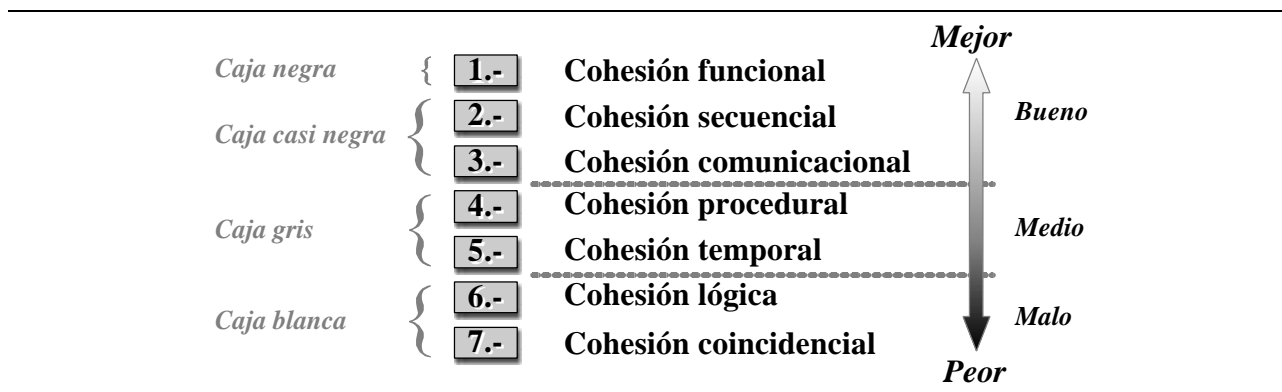


Fig. IV-26: Tipos de cohesión

El objetivo del diseño estructurado es obtener módulos altamente cohesivos, cuyos elementos estén fuerte y genuinamente relacionados unos con otros. Por otro lado, los elementos de un módulo no deberían estar fuertemente relacionados con elementos de otros módulos, porque eso llevaría a un fuerte acoplamiento entre ellos.

Clasificamos cada uno de los módulos de un DE con uno de los tipos de cohesión mostrados en la Fig. IV-26.

#### IV.5.2.1. Cohesión Funcional

Un módulo con cohesión funcional contiene elementos que contribuyen con la ejecución de una y solo una función. Ejemplos de nombres de módulos con cohesión funcional son:

- ✓ Calcular el Coseno de un ángulo
- ✓ Leer el Registro de Transacción
- ✓ Determinar la Deuda del Cliente
- ✓ Calcular el Punto de Impacto del Misil
- ✓ Calcular el Salario Líquido del empleado

Note que cada uno de estos módulos tiene un propósito fuerte y único. Cuando el módulo es invocado tiene solamente una tarea para concluir sin quedar involucrado en otra actividad. No importa cuán complejo sea el módulo y cuantas sub-funciones deben ser ejecutadas para completar su trabajo, si pueden ser resumidas en una única función, entonces el módulo tiene cohesión funcional.

El nombre del módulo con cohesión funcional siempre incluye un verbo. Si el nombre contiene dos verbos, entonces seguramente ejecuta dos tareas.

#### IV.5.2.2. Cohesión Secuencial

Un módulo con cohesión secuencial es aquel cuyos elementos están involucrados en actividades tales que los datos de salida de una actividad sirven como datos de entrada para la próxima. Por ejemplo, considere un módulo que incluye los pasos necesarios para pintar un auto de un color diferente al actual:

1. Limpiar el Auto
2. Tapar los Orificios de Chapa
3. Lijar el Auto
4. Aplicar la Primer Capa de Pintura

¿Qué nombre tiene el módulo? Es difícil asociar un nombre que contenga apenas un verbo, un nombre posible es: **Limpiar, Tapar Orificios y Lijar la Chapa y Aplicar la Primera Capa de Pintura** o se puede imaginar un nombre mas resumido como: **Reparar la Chapa y Aplicar la Primer Capa de Pintura**, pero no es posible resumirlo un nombre que contenga apenas un verbo. Si adicionamos los pasos siguientes:

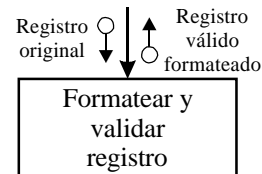
---

```

Módulo   Formatear y Validar Registro
usa      Registro original
           Formatear Registro original
           Validación cruzada de campos del registro
retorna   Registro válido formateado
Fin del módulo

```

---



**Fig. IV-27: Ejemplo de módulo con cohesión secuencial**

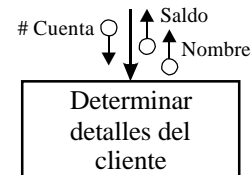
---

```

Módulo   Determinar detalles del cliente
usa      # Cuenta
           Informar nombre del cliente
           Informar saldo del cliente
retorna   Saldo y Nombre
Fin del módulo

```

---



**Fig. IV-28: Ejemplo de módulo con cohesión comunicacional**

---

5. Aplicar la Segunda Capa de Pintura
6. Encerar el Auto
7. Pulir el Auto

Ahora sí se puede imaginar un nombre con apenas un verbo (que indique una única función): **Pintar el Auto.**

La Fig. IV-27 muestra otro ejemplo, la salida de la primera actividad (formato) es la entrada para la siguiente (validación). Note que el módulo reúne dos funciones bien definidas.

Un módulo con cohesión secuencial generalmente tiene buen acoplamiento y es de fácil mantenimiento. La gran diferencia con los módulos con cohesión funcional es que un módulo con cohesión secuencial no es tan reutilizable, por contener actividades que en general no son útiles juntas.

### IV.5.2.3. Cohesión Comunicacional

Un módulo con cohesión comunicacional es aquel cuyas actividades usan los mismos datos de entrada. Tomemos un módulo que contiene funciones que retornan datos sobre un libro:

1. Buscar el Título del Libro
2. Buscar el Precio del Libro
3. Buscar el Código del Libro
4. Buscar el Autor del Libro

Las cuatro actividades anteriores están relacionadas porque todas trabajan sobre los mismos datos de entrada: el libro.

Otro ejemplo de un módulo con cohesión comunicacional se muestra en la Fig. IV-28, las dos actividades del módulo usan el mismo dato de entrada *# Cuenta*.

El acoplamiento entre un módulo con cohesión comunicacional y su llamador es usualmente aceptable. Usualmente son de fácil mantenimiento, a pesar que puedan existir problemas; por ejemplo, es posible que algún otro módulo en el sistema necesite obtener el nombre del cliente pero no estuviese interesado en su saldo. Este módulo debería descartar los datos del saldo o precisaría de otro módulo que implemente apenas la funcionalidad de consultar el nombre (duplicación de código).

Los módulos con cohesión comunicacional y secuencial se parecen mucho, pues ambos contienen actividades organizadas en torno a los datos. Ellos también tienen acoplamientos bastante claros porque pocos de sus elementos están relacionados con elementos en otros módulos. La principal diferencia entre ellos es que un módulo con cohesión secuencial trabaja en *secuencia* sobre los datos, una secuencia de transformación de datos semejante a una línea de producción (hay una orden escrita entre los componentes). Pero en un módulo con cohesión comunicacional, el orden de



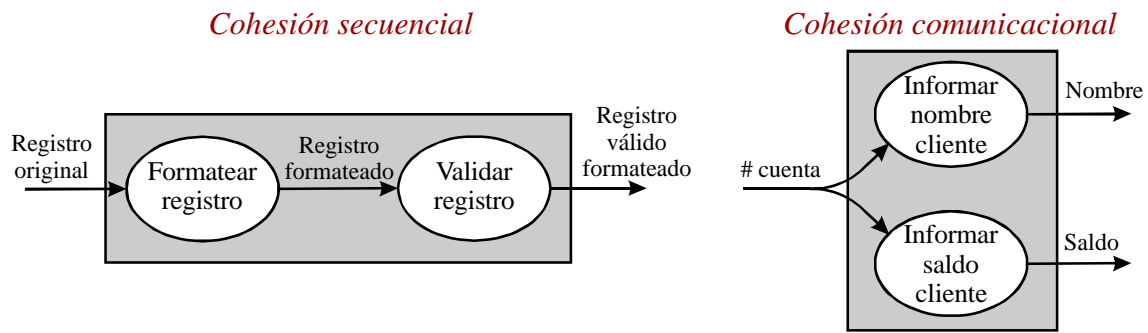


Fig. IV-29: Diferencia entre Cohesión Secuencial y Comunicacional

ejecución no es importante. Esta diferencia es ilustrada en la Fig. IV-29, usando una notación combinada de DE y DFDs.

#### IV.5.2.4. Cohesión Procedural

Un módulo con cohesión procedural es aquel cuyos elementos están involucrados en actividades diferentes, en las que el control es la principal relación. De manera semejante a los módulos con cohesión secuencial, el orden de los elementos es importante pero, en este caso, la secuencia es guiada por el control y no por la transformación de datos.

Los módulos con cohesión procedural tienden a estar compuestos por algunas funciones poco relacionadas entre sí (excepto por ser ejecutadas en un cierto orden). Sin embargo, esas funciones probablemente tienen mucho que ver con funciones en otros módulos. En la Fig. IV-30, **Grabar, Leer y Editar Registro** termina la ejecución de la última transacción (grabar el registro) y comienza la transacción siguiente (leer y editar el registro nuevo).

Es típico de un módulo con cohesión procedural que los datos de entrada tengan poca relación con los datos de salida. También es típico que sus salidas sean resultados parciales, flags, claves, etc.

#### IV.5.2.5. Cohesión Temporal

Un módulo con cohesión temporal es aquel cuyos elementos están involucrados en actividades que están relacionadas en el tiempo. Considere el siguiente ejemplo:

1. Inicializar Tabla
2. Leer Configuración del Sistema
3. Inicializar Variables Auxiliares
4. Completar Tabla con Valores Fijos

Estas actividades no están relacionadas entre sí, excepto porque son ejecutadas en un momento dado: son todas actividades de un módulo de inicialización al comienzo de ejecución de un programa. Los ejemplos típicos de cohesión temporal son módulos de inicialización y finalización. Los módulos con cohesión temporal, como los módulos con cohesión procedural, tienden a estar compuestos de funciones parciales cuya única relación es que todas suceden en un cierto momento. Las actividades están generalmente más relacionadas a actividades en otros módulos que entre sí, una situación que lleva a un elevado acoplamiento.

Los módulos con cohesión temporal, como los de cohesión procedural, no representan cajas negras. Son cajas grises ya que no es muy fácil definir simplemente la función de tal módulo sin mencionar sus detalles internos. También son parecidos, en el sentido que los módulos procedurales y temporales varían de mediocres a pobres. La diferencia entre ellos es igual a la diferencia entre

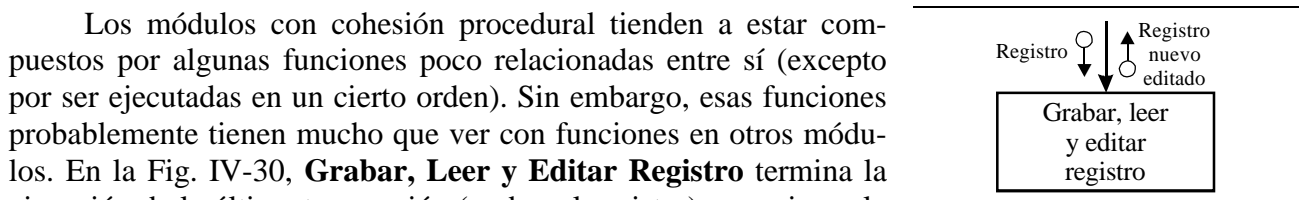


Fig. IV-30: Ejemplo de módulo con cohesión procedural



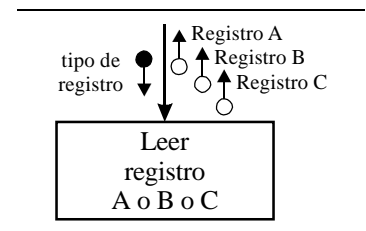
cohesión secuencial y comunicacional: el orden de ejecución de actividades es importante sólo en módulos con cohesión procedural. Además, los módulos procedurales tienden a compartir ciclos y decisiones entre funciones, mientras que los módulos temporales tienden a contener una codificación mas lineal.

#### IV.5.2.6. Cohesión Lógica

Un módulo con cohesión lógica es aquel cuyos elementos contribuyen en actividades de una misma categoría general, donde la actividad o las actividades a ser ejecutadas son seleccionadas fuera del módulo.

1. Leer Registro de Cliente
2. Leer Registro de Vendedor
3. Leer Registro de Producto
4. Leer Registro de Proveedor

La única relación entre las actividades del módulo del ejemplo anterior es que todas son operaciones de lectura. Un punto crucial y que identifica esta categoría de módulos es que en ejecución, se debe escoger una de estas actividades. Los módulos con cohesión lógica representan *cajas blancas* ya que son totalmente transparentes. Son de difícil comprensión y mantenimiento, no solamente por la mezcla de sus actividades, sino también, por la interfaz. Considere el ejemplo de la Fig. IV-31.



**Fig. IV-31: Ejemplo de módulo con cohesión lógica**

Comúnmente, un módulo con cohesión lógica necesita recibir una cupla de control para determinar la operación que tiene que ser ejecutada y, varios argumentos adicionales de los que sólo uno es usado. Esa es una interfaz muy compleja y de difícil mantenimiento.

#### IV.5.2.7. Cohesión Coincidental

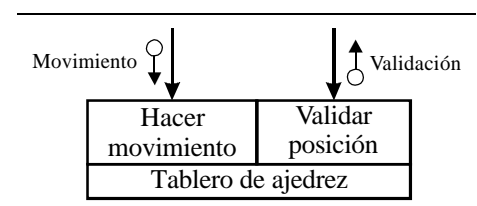
Un módulo con cohesión coincidental es aquel cuyos elementos contribuyen en actividades sin relación significativa entre ellos. Felizmente es rara. Entre sus causas están las tentativas inútiles para economizar tiempo o memoria, la incrustación de codificación monolítica en módulos arbitrarios y las alteraciones de mantenimiento intentando mejorar la mala cohesión de algunos otros módulos del sistema (dejando en un mismo módulo las porciones de código descartados de los otros).

Los módulos lógicos y coincidentales son similares en sus problemas. Como ellos no tienen ninguna función bien definida, el módulo jefe necesita enviar un flag completamente artificial para decirle qué hacer. Esto viola el principio de caja negra, ya que el módulo superior necesita conocer los detalles internos de los subordinados. El acoplamiento para ambos tipos de módulos es terriblemente oscuro. Hay una diferencia en favor de cohesión lógica, por lo menos las componentes de un módulo con cohesión lógica están relacionadas.

#### IV.5.2.8. Clusters de Información (Cohesión Informacional)

Los clusters de información sirven para modelar tipos abstractos de datos en un diagrama de estructura. Muestran una estructura de datos junto con los módulos de acceso y actualización.

Una pregunta a veces difícil de responder es: ¿Qué cohesión tiene un cluster de información? En realidad, cada uno de los módulos incluidos en un cluster de información es un módulo independiente y debe ser estudiado por separado. Así, en la Fig. IV-32, debemos identificar la cohesión y el acopla-



**Fig. IV-32: Ejemplo de cluster de información**

miento del módulo **Hacer Movimiento** y **Evaluar Posición** por separado.

A pesar que todos los módulos hacen referencia a un área de datos que, para ellos es global, ese área de datos está aislado en el cluster y no hereda los problemas de un acoplamiento común.

Cuando todos los módulos de un cluster de información tienen cohesión funcional, el cluster completo podría ser clasificado en otra categoría de cohesión identificada por Glenford Myers [Myers 78] como *Cohesión Informacional*:

Cuando un conjunto de módulos de cohesión funcional trabajan sobre la misma estructura de datos (en realidad son dependientes de esa estructura de datos), podrían ser reunidos en un sólo módulo y, el módulo resultante, tendría **Cohesión Informacional**.

Reunir todos los módulos que trabajan sobre la misma estructura de datos en un módulo sólo con cohesión informacional tiene la ventaja de aislar la estructura de datos del resto de los módulos. De esta manera, si la estructura de datos debe ser modificada, los módulos involucrados son fácilmente reconocidos. Un cluster de información permite reunir los módulos sin pérdida de identidad.

#### IV.5.2.9. Determinación de la cohesión de un módulo

Es posible determinar el tipo de cohesión que presenta un módulo contestando una sucesión de preguntas acerca de él, como lo muestra la Fig. IV-33.

### IV.5.3. Descomposición (Factoring)

La descomposición es la separación de una función contenida en un módulo, para un nuevo módulo. Puede ser hecha por cualquiera de las siguientes razones.

#### IV.5.3.1. Reducir el Tamaño del Módulo

La descomposición es una manera eficiente de trabajar con módulos grandes. Un buen tamaño para un módulo es alrededor de media página (30 líneas). Ciertamente, toda codificación de un módulo debería ser visible en una página (60 líneas) o en dos páginas consecutivas.

La cantidad de líneas no es un patrón rígido, otros criterios para determinar cuando es conveniente terminar de realizar la descomposición, son los siguientes:

- ✓ *Funcionalidad*: Terminar de realizar la descomposición cuando no se pueda encontrar una función bien definida. No empaquetar líneas de código dispersas, de otros módulos, porque probablemente juntas podrán formar módulos con cohesión temporal, coincidental o procedural.
- ✓ *Complejidad de Interfaz*: Terminar de realizar la descomposición cuando la interfaz de un módulo es tan compleja como el propio módulo. Un módulo de mil líneas es muy confuso, mas mil módulos de una línea son aún más confusos.

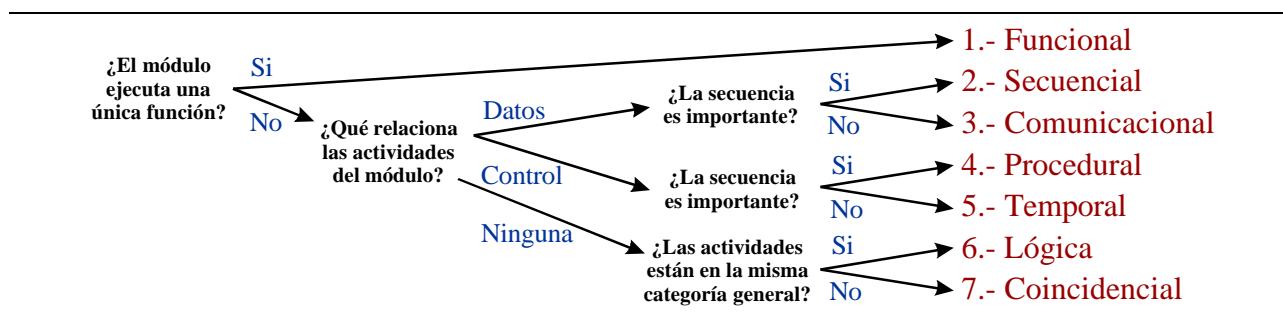


Fig. IV-33: Determinación de la cohesión de un módulo

### IV.5.3.2. Hacer el Sistema más Claro

La descomposición no debería ser hecha de una manera arbitraria, los módulos resultantes de la descomposición de un módulo deben representar sub-funciones del módulo de mas alto nivel en el DE.

En una descomposición no se debe preocupar por conceptos de programación. Si una sub-función, presentada como un módulo separado permite una mejor comprensión del diseño, puede ser descripta como se presenta en la Fig. IV-34, aún cuando, en una implementación, el código del módulo sea programado dentro del módulo jefe.

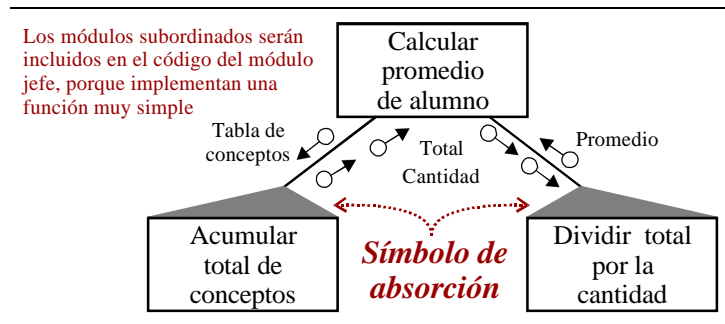


Fig. IV-34

### IV.5.3.3. Minimizar la Duplicación de Código

Cuando se reconoce una función que puede ser reutilizada en otras partes del DE, lo mas conveniente es convertirla en un módulo separado. Así, se puede localizar mas fácilmente las funciones ya identificadas y evitar la duplicación del mismo código en el interior de otro módulo. De esta manera, los problemas de inconsistencia en el mantenimiento (si esa función debe ser modificada) pueden ser evitados, y se reduce el costo de implementación.

### IV.5.3.4. Separar el Trabajo de la 'Administración'

Un administrador o gerente de una compañía bien organizada debería coordinar el trabajo de los subordinados en lugar de *hacer* el trabajo. Si un gerente hace el trabajo de los subordinados no tendrá tiempo suficiente para coordinar y organizar el trabajo de los subordinados y, por otro lado, si hace el trabajo los subordinados no serían necesarios. Lo mismo se puede aplicar al diseño del DE, relacionado a los módulos de *Trabajo* (edición, cálculo, etc.) y a los módulos de *Gerencia* (decisiones y llamadas para otros módulos).

El resultado de este tipo de organización es un sistema en cual los módulos en los niveles medio y alto de un DE son fáciles de implementar, porque ellos obtienen el trabajo hecho por la manipulación de los módulos de los niveles inferiores.

La separación del trabajo de la *administración* mejora la mantenibilidad del diseño. Una alteración en un sistema es: un cambio de control o un cambio de trabajo, pero raramente ambos.

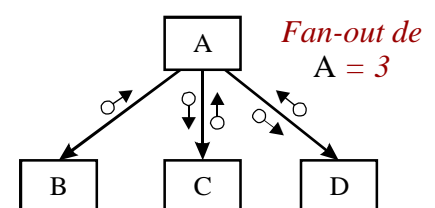
### IV.5.3.5. Crear Módulos más Generales

Otra ventaja de la descomposición es que, frecuentemente, se pueden reconocer módulos más generales y, así, más útiles y reutilizables en el mismo sistema y, además, pueden ser generadas bibliotecas de módulos reutilizables en otros sistemas.

## IV.5.4. Fan-Out

El *fan-out* de un módulo es usado como una medida de *complejidad*. Es el número de subordinados inmediatos de un módulo (cantidad de módulos invocados).

Si un módulo tiene un fan-out muy grande, entonces compone el trabajo de muchos módulos subordinados y, casi con certeza, tiene toda la funcionalidad no trivial representada por ese



subárbol en el DE.

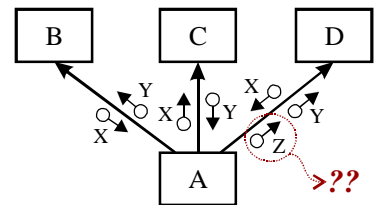
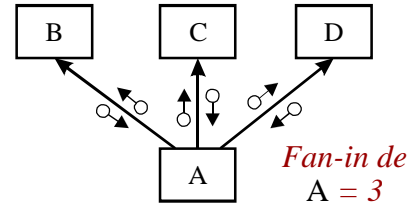
Para tener acotada la complejidad de los módulos se debe limitar el *fan-out* a no más de siete más o menos dos ( $7 \pm 2$ ). Un módulo con muchos subordinados puede fácilmente ser mejorado por descomposición.

### IV.5.5. Fan-In

El *fan-in* de un módulo es usado como una medida de *reusabilidad*, es el número de superiores inmediatos de un módulo (la cantidad de módulos que lo invocan).

Un alto fan-in es el resultado de una descomposición inteligente. Durante la programación, tener una función llamada por muchos superiores evita la necesidad de codificar la misma función varias veces. Existen dos características fundamentales que deben ser garantizadas en módulos con un alto fan-in:

- ✓ **Buena Cohesión:** Los módulos con mucho fan-in deben tener cohesión funcional, secuencial o comunicacional (es muy probable que tenga acoplamiento de datos).
- ✓ **Interfaz Consistente:** Cada invocación para el mismo módulo debe tener el mismo número y tipo de parámetros. En el ejemplo de la derecha, hay un error de este tipo.



### IV.5.6. Criterios Adicionales de Diseño

Existen otros criterios para evaluar un diseño que, usados en combinación con cohesión, acoplamiento, descomposición, fan-out y fan-in permiten proyectar sistemas más fáciles de mantener. Algunos de estos criterios son presentados a continuación.

#### IV.5.6.1. Evitar División de Decisiones

Una decisión tiene dos partes: el reconocimiento de una condición particular y la ejecución de acciones asociadas. Esas partes son muy diferentes, considere el siguiente ejemplo:

```
Si el # de cuenta del cliente es desconocido <--- Reconocimiento
Entonces rechazar el registro del cliente <--- Ejecución
Fin
```

Los datos usados por la parte de reconocimiento (*# de cuenta del cliente*) y los usados por la parte de ejecución (*registro del cliente*) no son los mismos, y tal vez no estén disponibles en mismo módulo. Si los datos no están disponibles en el mismo módulo, ocurre una división de decisión, esto es, la separación de las dos partes de una decisión en módulos diferentes.

La parte de *ejecución* de decisión debe ser mantenida tan cerca como sea posible de la parte de *reconocimiento* (si es posible, en el mismo módulo), con el objetivo que la información reconocida no tenga que recorrer un largo camino para ser procesada. La división de decisiones es un síntoma de pobre organización de módulos y, frecuentemente, genera datos vagabundos, como se muestra en la Fig. IV-35.

#### IV.5.6.2. Forma del Sistema

Hasta ahora, todos los criterios descriptos, estudian las características de un módulo o de un par de módulos. Sin embargo, algunas características generales del diagrama de estructura, considerado en conjunto, proporcionan también indicios de calidad del sistema.

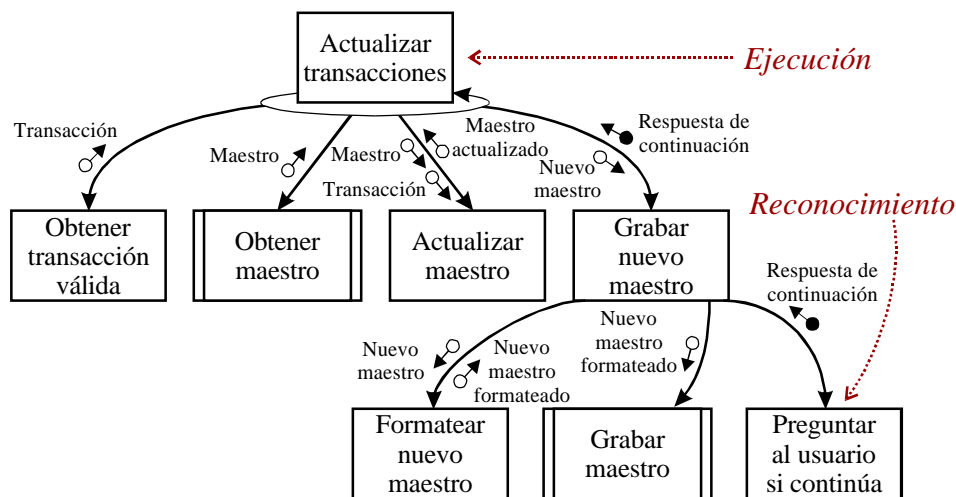


Fig. IV-35: DE con división de decisiones

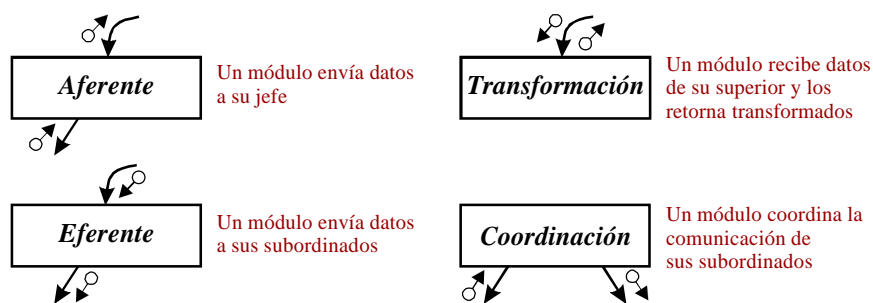


Fig. IV-36: Cuatro tipos de módulos

La forma del sistema se relaciona con la combinación de cuatro tipos básicos de módulos, determinados por la dirección de sus flujos de datos: módulos aferentes, módulos eferentes, módulos de transformación (o de trabajo), y módulos de coordinación. Esos cuatro tipos de módulos son mostrados en la Fig. IV-36.

Tradicionalmente los diferentes tipos de módulos están agrupados en el diagrama de estructura, como indica la Fig. IV-37.

#### IV.5.6.2.1. Sistemas Dirigidos por Entradas Físicas (Physically Input-Driven)

Un sistema dirigido por entradas físicas es aquel que realiza poco procesamiento en su lado aferente, de modo que los módulos superiores tienen que trabajar con datos vírgenes, muy dependientes de los dispositivos físicos (no editados y no-refinados).

Considere el ejemplo de la Fig. IV-38. Este es un sistema dirigido por la entrada física, porque el módulo superior **Actualizar Archivo**, debe trabajar con datos físicos y no refinados. Algunos defectos de estos tipos de sistemas son:

- ✓ El acoplamiento es generalmente alto, debido a los problemas de división de decisiones. Note que el módulo **Formatear Campo** envía una cupla de control para arriba *Precisa de Otra Línea* (acoplamiento de control con inversión de autoridad).
- ✓ Muchos módulos en la parte superior del DE están relacionados con el formato físico de entrada. Una gran parte del sistema sería afectado si se produce un cambio de especificación secundaria (el formato físico de entrada de las transacciones).
- ✓ Los módulos que producen entradas editadas pueden no ser reutilizables.

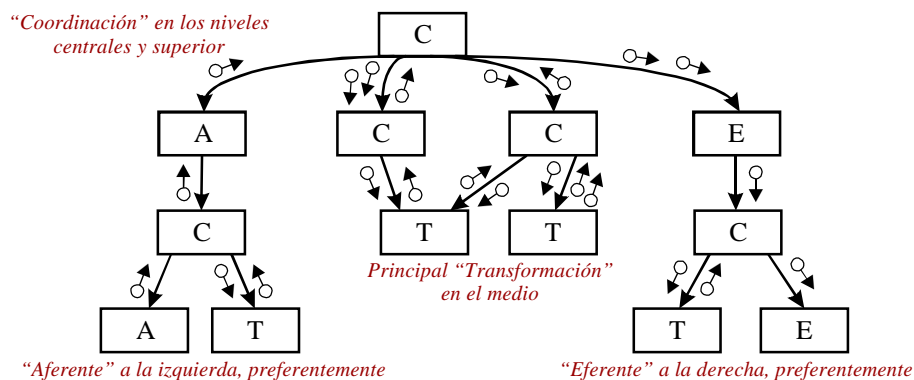


Fig. IV-37: Forma del patrón de agrupamiento de los tipos de módulos

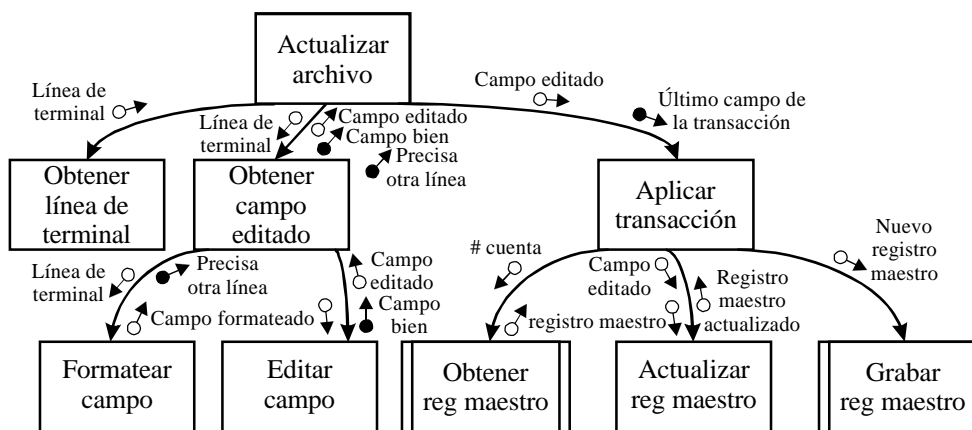


Fig. IV-38: Ejemplo de sistema dirigido por la entrada física

Menos comunes son los sistemas dirigidos por la salida física. En este caso, el módulo superior queda sujeto al formato físico exacto de la salida. Esos sistemas tienen casi todos los problemas idénticos a los sistemas dirigidos por la entrada física.

#### IV.5.6.2.2. Sistemas Balanceados

En un sistema *balanceado*, los detalles dependientes de los dispositivos físicos de entrada y salida quedan aislados en los niveles más bajos del diagrama de estructura. Una mínima cantidad de módulos deben trabajar con dispositivos físicos y, los módulos del nivel medio y superior, trabajar solamente con datos refinados. En otras palabras, la idea es independencia de dispositivos.

Un sistema balanceado de esta forma puede ser definido como que no está siendo dirigido por la entrada física, ni por la salida física. Una de las principales ventajas del análisis de transformaciones es que el resultado siempre son diagramas de estructura balanceados.

La Fig. IV-39 muestra un ejemplo.

#### IV.5.6.3. Tratamiento de Errores

Los errores deben ser informados por el módulo que los detecta y que conoce su naturaleza. Por ejemplo, supóngase que se necesita generar un registro de entrada leyendo campo a campo, y chequear la validez de cada campo.

En la Fig. IV-40 se presentan dos ejemplos de tratamiento de error. En el caso *a*), el error se presenta en el momento en que es reconocido, el módulo **Presentar Mensaje de Error** puede presentar dos tipos de mensajes:

- ✓ "el nombre de la ciudad es inválido": En este caso, un módulo claramente reutilizable como **Validar Campo Alfabético** solamente puede ser usado para la validación de nombres de ciudades. El módulo pierde su flexibilidad de uso innecesariamente.
- ✓ "el campo alfabético es inválido": este mensaje sólo serviría para confundir al usuario.

La mejor solución sería postergar el tratamiento del error para que sea tratado por un módulo que conozca más acerca del campo alfabético que esta siendo leído. Así, el módulo **Validar Campo Alfabético** simplemente informa que un error fue identificado y, el error será tratado por algún otro módulo.

En el caso *b)*, la anomalía anterior es corregida pero, el error está siendo tratado a cierta distancia de donde es detectado. Eso acarrea un acoplamiento extra (la cupla *Nombre de Ciudad Inválido*).

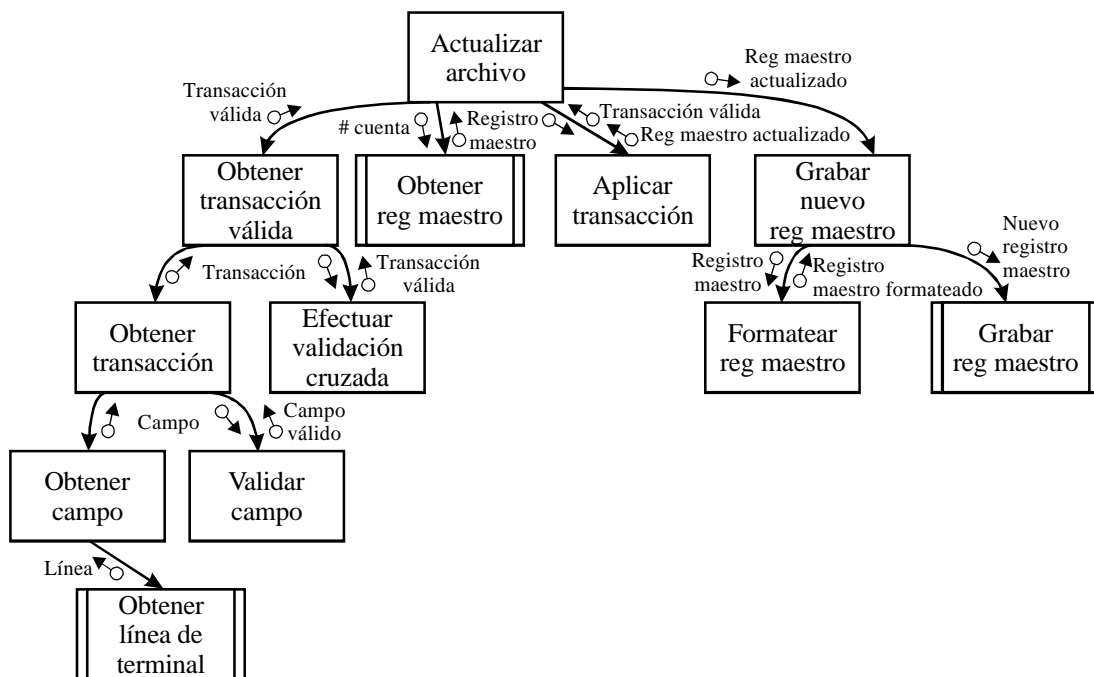


Fig. IV-39: Ejemplo de sistema balanceado

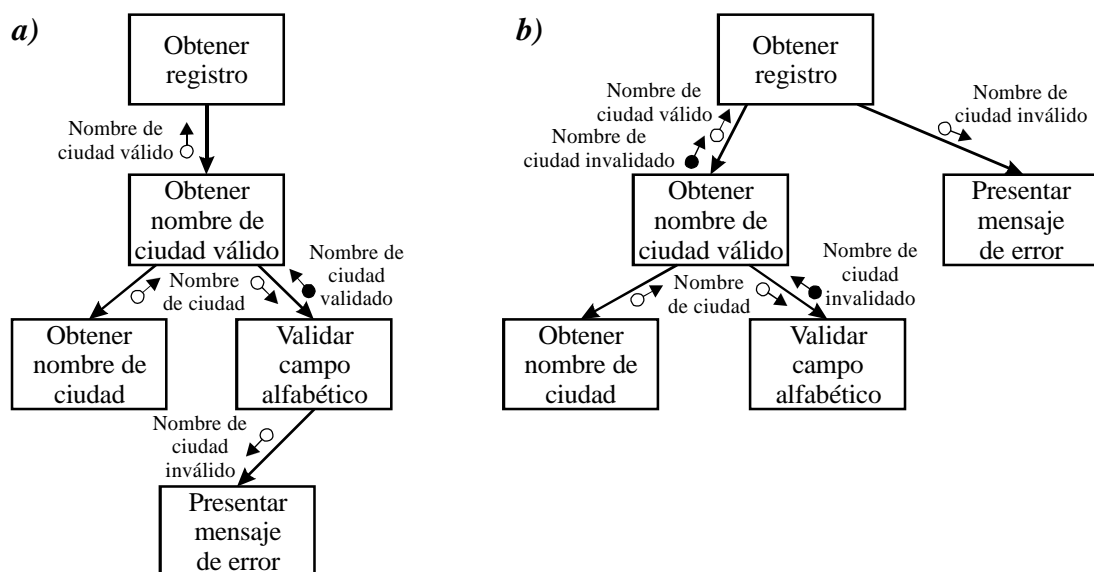
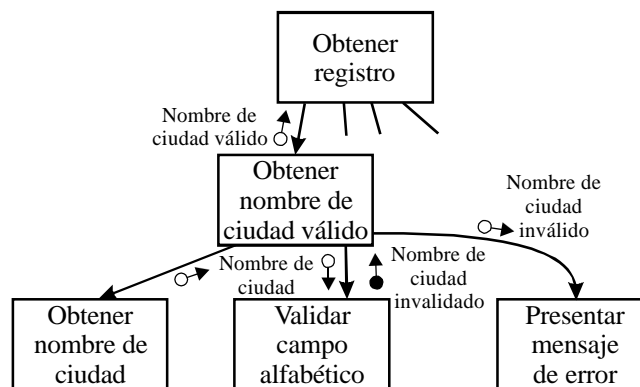


Fig. IV-40: Dos ejemplos de tratamiento de error



**Fig. IV-41: Otro ejemplo de tratamiento de error**

*lido*, a pesar que sea una cupla descriptiva, es un dato vagabundo). La Fig. IV-41 muestra una forma más adecuada donde el error es tratado en un módulo que tiene conocimiento del error.

Una cuestión aún no respondida es dónde tratar los mensajes de error. Los mensajes pueden estar esparcidos por todo el sistema, de modo que cada mensaje pertenezca al módulo que detecta el error, o pueden estar todos almacenados en un sólo módulo. La segunda opción (almacenar todas juntas) consiste en incorporar un módulo que probablemente tenga cohesión lógica y que, con certeza, tendrá acoplamiento de control. Sin embargo esta solución posee varias ventajas, algunas de las cuales son:

- ✓ Es más fácil conservar consistente el texto y el formato de las mensajes. Los usuarios quedan muy confundidos cuando reciben dos mensajes diferentes o con diferente formato.
- ✓ Es posible almacenar el texto de los mensajes en un archivo y no tener siempre probablemente una gran cantidad de mensajes diferentes, cargados en memoria.
- ✓ Es más fácil evitar mensajes duplicados.
- ✓ Es más fácil actualizar mensajes y hasta traducirlos a otro lenguaje.

#### IV.5.6.4. Relación entre Estructuras de Datos y Estructura de Programa

La programación de muchos módulos se facilita si la estructura del sistema sigue la estructura de los datos. Por ejemplo, suponga que es preciso leer una línea de 120 caracteres, sin embargo, la entrada tiene un dispositivo que proporciona caracteres simples. En este caso, en el diagrama de estructura se debería tener una estructura semejante a la estructura de datos. Es decir, un módulo para procesamiento de líneas y un módulo subordinado a él para el procesamiento de cada carácter (que será invocado 120 veces por el módulo jefe).

#### IV.5.6.5. Memoria Estática

Un módulo común muere y retorna a sus superiores una vez cumplida su función. Cuando el mismo módulo es llamado nuevamente, este reacciona como si fuera la primera vez que lo llaman. El módulo no tiene memoria de ninguna cosa que haya ocurrido en las llamadas anteriores, menos aún, no tiene conocimiento de si existió algún llamado previo.

Hay una clase de módulos que tiene conocimiento de su pasado a través de la memoria estática. Es un dato interno que sobrevive a las invocaciones sucesivas.

Un módulo con memoria estática generalmente es impredecible, porque aunque lo llamemos con entradas idénticas, el módulo puede actuar diferente o producir diferentes salidas cada vez que se lo invoca. Es más difícil mantener módulos con memoria estática que módulos comunes, ya que los errores asociados a módulos con memoria estática pueden depender del estado actual de la memoria. Esta es una buena razón para evitar los módulos con memoria estática.



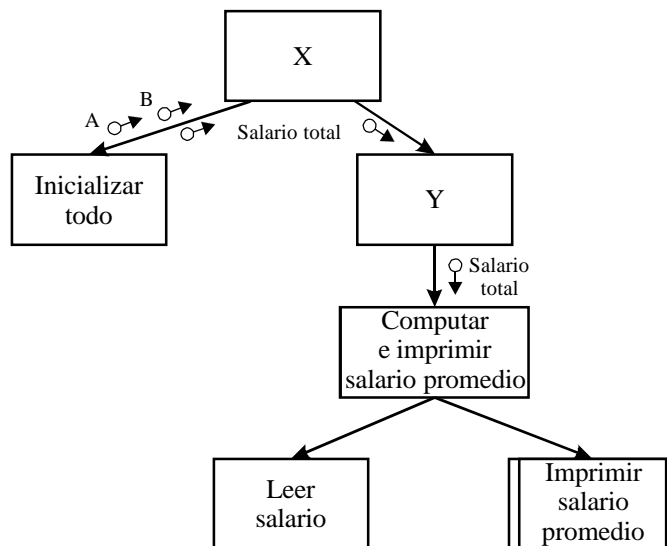


Fig. IV-42: Ejemplo de módulo de inicialización

#### IV.5.6.6. Módulos de Inicialización y Terminación

Las inicializaciones y terminaciones en algún momento deben hacerse, pero no deberían crear problemas de cohesión temporal.

Veamos un ejemplo. La Fig. IV-42 muestra que se inicializaron todos los datos en un módulo **Inicializar Todo**, incluyendo *Salario Total*, que es usado por el módulo **Computar e Imprimir Salario Promedio**. Para eso *Salario Total* pasa como una cupla vagabunda a otros módulos.

Hay otros problemas:

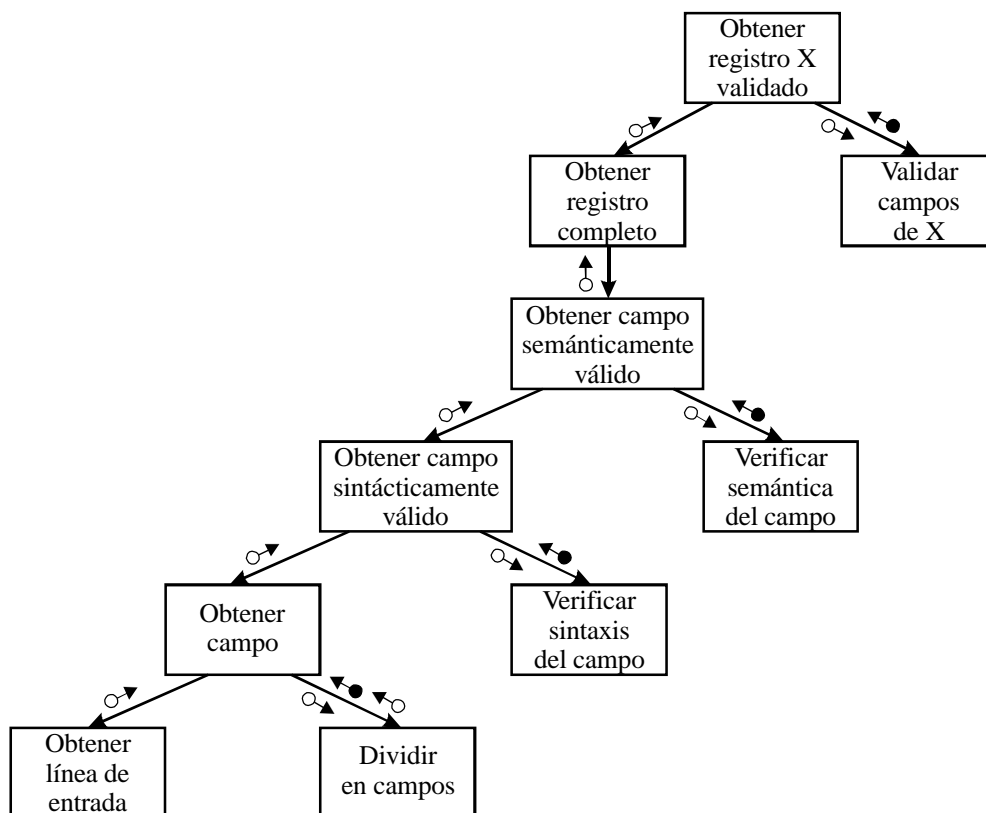
- ✓ La inicialización está muy lejos del uso, es muy probable que la inicialización se olvide.
- ✓ El módulo **Computar e Imprimir Salario Promedio** no es muy útil, no puede ser reutilizado ya que no inicializa nada.
- ✓ El módulo **Inicializar Todo** tampoco es muy útil ya que cada vez que se lo llama inicializa todos los datos.

Podemos mejorar **Inicializar Todo** si lo factorizamos en una cierta cantidad de módulos de inicialización, uno por cada función inicializadora. Además cada uno de estos módulos se debería ubicar en el lugar donde se lo usa.

#### IV.5.6.7. Edición

¿Cómo debería hacerse la edición de una pieza de entrada? Para responder esta pregunta suponga que se debe ingresar el nombre de una provincia argentina. Para ello tenga en cuenta lo siguiente:

- ✓ Siempre se debe chequear el formato y luego el sentido. Por ejemplo, MEN;DOZA es sintácticamente incorrecto, mientras que MENDOSA es semánticamente incorrecto.
- ✓ Primero se validan los datos individualmente y luego en conjunto.
- ✓ Antes de procesar la entrada esté seguro que todo está verificado. Por ejemplo, en una operación de extracción de dinero, la suma de 200 dólares es sintácticamente correcta, pero debe rechazarse si se tiene un saldo de 150 dólares.



**Fig. IV-43: Fragmento de DE que realiza la edición de una entrada**

Un diagrama de estructura general para editores podría ser similar al que aparece en la Fig. IV-43.

# Bibliografía

- [Adler88] Mike Adler, "An Algebra for Data Flow Diagram Process Decomposition", IEEE Trans on Soft Eng, Vol 14 N° 2, Feb 1988.
- [Batini92] Carlo Batini, Stefano Ceri & Shamkant B. Navathe, "Conceptual Database Design an Entity-Relationship Approach", Addison-Wesley, Benjamin/Cummings Pub, Inc., 1992.
- [Boria87] Jorge Boria, "Ingenieria de Software", (I EBAI), Kapelusz S.A., Bs.As. Arg, 1987.
- [Chen76] Peter Pin-Shan Chen, "The Entity-Relationship Model - Toward a Unified View of Data", ACM Trans. on Database Systems, Vol 1, N° 1, Mar 1976.
- [Codd71] Codd, E.F., "Normalized Database Structures: A brief tutorial", Proc ACM SIGFIDET 1971, Workshop, San Diego, Calif., Nov 1971.
- [Codd72] Codd, E.F., "Further Normalization of the Data Base Relational Model", Courant Computer Science Symposia, 6, Data Base Systems, edited by R. Rustin, Prentice-Hall Inc., Englewood Cliffs, N.J., 1972.
- [Constant74] Wayne Stevens, Glenford j. Myers & Larry L. Constantine, "Structured Design", IBM System Journal, Vol 13, N° 2, May 1974.
- [DeMarco79] Tom DeMarco, "Structured Analysis and Systems Specification", Prentice-Hall, Inc., Englewood Cliffs, N.J., 1979.
- [Flavin81] M. Flavin, "Fundamental Concepts of Information Modeling", Yourdon Press, N.Y., 1981.
- [Gane79] Chris Gane and Trish Sarson, "Structured Systems Analysis: Tools and Techniques", Prentice-Hall, Inc., Englewood Cliffs, N.J., 1979.
- [Martin81] James Martin & Clive Finkelstein, "Information Engineering", Savant Research Studies, Vol 1, 2, e 3, 1981.
- [Martin87] James Martin, "Recommended Diagramming Standards for Analysts and Programmers: A Basis for Automation", Prentice-Hall, Englewood Cliffs, 1987.
- [Martin91] James Martin & Carma McClure, "Técnicas Estruturadas e CASE", McGraw-Hill Ltda., S.P., Brasil, 1991.
- [McMenam84] Sthephen M. McMenamim and Jhon F. Palmer, "Essential Systems Analysis", Prentice-Hall, Inc., 1984.
- [McMenam91] Sthephen M. McMenamim and Jhon F. Palmer, "Análise Essencial de Sistemas", Tradução Lars Gustav Erik Unonius, McGraw-Hill, Ltda e Makron Books Editora Ltda, 1991.
- [Myers78] Glenford J. Myers, "Composite/Structured Design", Van Nostrand Reinhold, Litton Educational Pub., Inc., 1978.
- [Page88] Meilir Page-Jones, "The Practical Guide to Structured System Design", Prentice-Hall, Inc. 1988.
- [Page88b] Meilir Page-Jones, "Diseño Estructurado de Sistemas", McGraw-Hill, Ltda., S.P. 1988.
- [Parnas72] D.L. Parnas, "On the Criteria to Be Used in Decomposing Systems into Modules", CACM, Dec 1972.
- [Parnas79] D.L. Parnas, "Designing Software for Ease of Extension and Contraction", IEEE Trans. on Soft. Eng., Mar 1979.
- [Peters81] Lawrence J. Peters, "Software Design: Methods & Techniques", Yourdon Press Inc, N. Y., 1981.
- [Rochfeld92] Arnold Rochfeld & Pascal Negros, "Relationship of Relationships and other inter-relationship links in E-R model", Data & Knowledge Engineering, N° 9, 1992.
- [Ross76] D. T. Ross & J. W. Brackett, "An approach to structured analysis", Comp. Decisions, Vol 8, N° 9, Sep 1976.
- [Stevens81] Wayne P. Stevens, "Using Structured Design", John Wiley & Sons, Inc., 1981.
- [Stevens85] Wayne P. Stevens, "Projeto Estruturado de Sistemas", Editora Campus Ltda. R.J., 1985.
- [Ullman82] Jeffrey D. Ullman, "Principles of Database Systems", Computer Science Press, Inc. 1982.

- [Ward83] Paul T. Ward, "Systems Development Without Pain: a user's guide to modeling organizational patterns", Yourdon Press. 1983.
- [Ward85] Paul T. Ward & Stephen J. Mellor, "Structured Development for Real-Time Systems", Vol 1: Introduction & Tools, Prentice-Hall Inc., Englewood Cliffs, N.J., 1985.
- [Ward85a] Paul T. Ward and Stephen J. Mellor, "Structured Development for Real-Time Systems" Vol 1: Introduction & Tools, Prentice-Hall, Inc., 1985.
- [Ward85b] Paul T. Ward and Stephen J. Mellor, "Structured Development for Real-Time Systems" Vol 2: Essential Modeling Techniques, Prentice-Hall, Inc., 1985.
- [Ward86] Paul T. Ward and Stephen J. Mellor, "Structured Development for Real-Time Systems" Vol 3: Implementation Modeling Techniques, Prentice-Hall, Inc., 1986.
- [Ward86] Paul T. Ward, "The Transformation Schema: An Extension of the Data Flow Diagram to Represent Control and Timing", IEEE Trans. on Soft. Eng., Vol 12, N° 2, Feb 1986.
- [Yourdon78] Edward Yourdon & Larry L. Constantine, "Structured Design: Fundamentals of la Discipline of Computer Program and Systems Design", Yourdon Press, N.Y., 1978.
- [Yourdon89] Edward Yourdon, "Modern Structured Analysis", Prentice-Hall, Inc., 1989.
- [Yourdon90] Edward Yourdon, "Análise Estruturada Moderna", Traducción Dalton Conde de Alencar, Editora Campus Ltda., 1990.

# Contenido

<b>Introducción .....</b>	<b>1</b>
<b>CAPÍTULO I. ASML: A SYSTEM MODELING LANGUAGE .....</b>	<b>5</b>
<b>Contenido.....</b>	<b>5</b>
<b>I.1. Estructura de la Metodología.....</b>	<b>5</b>
I.1.1. El Modelo Esencial.....	5
I.1.2. El Modelo de Implementación.....	6
<b>I.2. Secuencia de Creación de los Modelos .....</b>	<b>8</b>
<b>CAPÍTULO II. ANÁLISIS ESTRUCTURADO MODERNO .....</b>	<b>9</b>
<b>Contenido.....</b>	<b>9</b>
<b>II.1. Introducción .....</b>	<b>9</b>
II.1.1. El Modelo Esencial .....	10
II.1.2. Dificultades en la Construcción del Modelo Esencial.....	10
II.1.3. Componentes del Modelo Esencial .....	11
II.1.4. ¿Cuándo es Necesario Desarrollar un Modelo del Sistema Actual? .....	11
<b>II.2. El Modelo Ambiental.....</b>	<b>11</b>
II.2.1. La Declaración de Objetivos .....	13
II.2.2. El Diagrama de Contexto .....	13
II.2.2.1. Construcción del Diagrama de Contexto.....	14
II.2.3. La Lista de Eventos .....	16
II.2.3.1. Construcción de la Lista de Eventos.....	18
II.2.4. ¿Qué va primero: el Diagrama de Contexto o la Lista de Eventos?.....	18
II.2.5. El Diccionario de Datos Inicial .....	19
<b>II.3. El Modelo Comportamental .....</b>	<b>19</b>
II.3.1. Creación del DFD.....	19
II.3.2. Creación del Modelo de Datos .....	22
II.3.3. Subdivisión en Niveles.....	23
II.3.4. El Diccionario de Datos .....	24
<b>CAPÍTULO III. HERRAMIENTAS DE ANÁLISIS ESTRUCTURADO .....</b>	<b>26</b>
<b>Contenidos .....</b>	<b>26</b>
<b>III.1. Diagramas de Flujo de Datos .....</b>	<b>27</b>
III.1.1. Flujos de Datos.....	28
III.1.2. Procesos .....	29
III.1.3. Depósitos de Datos.....	29
III.1.4. Agentes Externos .....	30
III.1.5. Refinamiento de Procesos en un DFD .....	30
III.1.6. Reglas de Validación .....	31
III.1.7. Álgebra de Descomposición de Procesos .....	32
III.1.8. Construcción Sistemática del DFD .....	36
III.1.8.1. Caso de Estudio – Administración Hotelera .....	36
III.1.8.1.1. Objetivos Funcionales .....	36
III.1.8.1.2. Construcción de la Lista de Eventos.....	37
III.1.8.1.3. Lista de Eventos construida para el Caso de Estudio .....	37

III.1.8.1.4. Construcción del Diagrama de Contexto .....	38
III.1.8.1.5. Diagrama de Contexto y Tabla de Estímulo-Respuesta para el Caso de Estudio.....	38
III.1.8.1.6. Derivación del DFD preliminar por eventos.....	40
<b>III.2. Diccionario de Datos (DD) .....</b>	<b>44</b>
III.2.1. Notación .....	44
<b>III.3. Modelo Entidad Relación (ERD) .....</b>	<b>45</b>
III.3.1. Entidades y Atributos .....	46
III.3.2. Relaciones.....	47
III.3.2.1. Relación Uno-a-Uno.....	48
III.3.2.1.1. Opcionalidad.....	48
III.3.2.2. Relación Uno-a-Muchos.....	49
III.3.2.2.1. Opcionalidad.....	49
III.3.2.3. Relación Muchos-a-Muchos.....	50
III.3.2.3.1. Opcionalidad.....	50
III.3.2.4. Relaciones Indefinidas.....	50
III.3.3. Mecanismos de Abstracción.....	51
III.3.3.1. Clasificación .....	51
III.3.3.2. Agregación de Atributos ( <i>atributos compuestos</i> ) .....	51
III.3.3.3. Especialización ( <i>es-un</i> o <i>es-subtipo-de</i> ) .....	51
III.3.3.4. Agregación de Entidades ( <i>compuesto-por</i> ) .....	51
III.3.3.5. Entidades Relacionantes .....	52
III.3.4. Construcción de un Diagrama Entidad-Relación .....	52
<b>III.4. Especificación de Procesos.....</b>	<b>53</b>
III.4.1. Lenguaje Estructurado.....	53
III.4.2. Pre/Pos Condiciones .....	54
III.4.3. Tabla de Decisión .....	56
III.4.4. Árboles de Decisión .....	56
III.4.5. Diagramas de Nassi-Shneiderman.....	57
<b>III.5. Diagramas de Transición de Estados (DTE).....</b>	<b>58</b>
III.5.1. Estados.....	58
III.5.2. Transiciones.....	59
III.5.3. Representación en Tabla de Transición.....	59
III.5.4. Representación en Diagramas de Grade.....	60
<b>CAPÍTULO IV. ASML: ANÁLISIS ESTRUCTURADO DE SISTEMAS .....</b>	<b>61</b>
Contenidos.....	61
<b>IV.1. Introducción .....</b>	<b>62</b>
<b>IV.2. Modelo de Implementación del Usuario .....</b>	<b>63</b>
IV.2.1. Determinación de los Límites de la Automatización .....	64
IV.2.2. Determinación de la Interfaz del Usuario .....	65
IV.2.3. Identificación de Actividades Manuales Complementarias .....	67
IV.2.4. Especificación de Restricciones Operacionales .....	67
<b>IV.3. Diagrama de Estructura.....</b>	<b>68</b>
IV.3.1. Módulos.....	69
IV.3.2. Relaciones entre Módulos (Invocaciones) .....	69
IV.3.3. Comunicación entre Módulos (Cuplas).....	70
IV.3.4. Absorción .....	71
IV.3.5. Estructuras de Control.....	71
<b>IV.4. Derivación de los Diagramas de Estructura.....</b>	<b>72</b>
IV.4.1. Análisis de Transformaciones .....	72
IV.4.1.1. Análisis de la Especificación del Problema.....	72

IV.4.1.2. Identificar el Centro de Transformación .....	73
IV.4.1.2.1. Estrategia para Determinar el Centro de Transformación .....	74
IV.4.1.3. Producir un Primer Diagrama de Estructura (First-Cut) .....	76
IV.4.1.4. Mejorar el Diagrama de Estructura Obtenido .....	76
IV.4.1.5. Garantizar la Funcionalidad del Diseño .....	77
IV.4.2. Análisis de Transacción .....	77
<b>IV.5. Criterios de Validación de Calidad.....</b>	<b>80</b>
IV.5.1. Acoplamiento .....	81
IV.5.1.1. Acoplamiento sin Cuplas .....	81
IV.5.1.2. Acoplamiento de Datos .....	82
IV.5.1.3. Acoplamiento Estampado .....	83
IV.5.1.4. Acoplamiento de Control .....	83
IV.5.1.5. Acoplamiento Híbrido.....	84
IV.5.1.6. Acoplamiento Común .....	85
IV.5.1.7. Acoplamiento por Contenido (o Patológico) .....	85
IV.5.2. Cohesión .....	85
IV.5.2.1. Cohesión Funcional.....	86
IV.5.2.2. Cohesión Secuencial .....	86
IV.5.2.3. Cohesión Comunicacional .....	87
IV.5.2.4. Cohesión Procedural .....	88
IV.5.2.5. Cohesión Temporal .....	88
IV.5.2.6. Cohesión Lógica .....	89
IV.5.2.7. Cohesión Coincidental .....	89
IV.5.2.8. Clusters de Información (Cohesión Informacional).....	89
IV.5.2.9. Determinación de la cohesión de un módulo .....	90
IV.5.3. Descomposición (Factoring).....	90
IV.5.3.1. Reducir el Tamaño del Módulo.....	90
IV.5.3.2. Hacer el Sistema más Claro .....	91
IV.5.3.3. Minimizar la Duplicación de Código .....	91
IV.5.3.4. Separar el Trabajo de la ‘Administración’ .....	91
IV.5.3.5. Crear Módulos más Generales .....	91
IV.5.4. Fan-Out .....	91
IV.5.5. Fan-In.....	92
IV.5.6. Criterios Adicionales de Diseño .....	92
IV.5.6.1. Evitar División de Decisiones.....	92
IV.5.6.2. Forma del Sistema.....	92
IV.5.6.2.1. Sistemas Dirigidos por Entradas Físicas (Physically Input-Driven).....	93
IV.5.6.2.2. Sistemas Balanceados .....	94
IV.5.6.3. Tratamiento de Errores.....	94
IV.5.6.4. Relación entre Estructuras de Datos y Estructura de Programa.....	96
IV.5.6.5. Memoria Estática .....	96
IV.5.6.6. Módulos de Inicialización y Terminación.....	97
IV.5.6.7. Edición .....	97
<b>BIBLIOGRAFÍA.....</b>	<b>99</b>
<b>CONTENIDO.....</b>	<b>101</b>