CAPÍTULO

14 DISEÑO ARQUITECTÓNICO

EL diseño se ha descrito como un proceso multifase en el que se sintetizan representaciones de la estructura de los datos, la estructura del programa, las características de la interfaz y los detalles procedimentales desde los requisitos de la información. Esta descripción es ampliada por Freeman [FRE80]:

> El diseño es una actividad en la que se toman decisiones importantes, frecuentemente de naturaleza estructural. Comparte con la programación un interés por la abstracción de la representación de la información y de las secuencias de procesamiento, pero el nivel de detalle es muy diferente en ambos casos. El diseño construye representaciones coherentes y bien planificadas de los programas, concentrándose en las interrelaciones de los componentes de mayor nivel y en las operaciones lógicas implicadas en los niveles inferiores...

Como dijimos en el capítulo anterior, el diseño está dirigido por la información. Los métodos de diseño del software se obtienen del estudio de cada uno de los tres dominios del modelo de análisis. El dominio de los datos, el funcional y el de comportamiento sirven de directriz para la creación del diseño.

En este capítulo se introducen los métodos requeridos para la creación de «representaciones coherentes y bien planeadas» de los datos y las capas arquitectónicas del modelo de diseño. El objetivo es proporcionar un enfoque sistemático para la obtención del diseño arquitectónico —el anteproyecto preliminar desde el que se construye el software—.

VISTAZO RÁPIDO

¿Qué es? El diseño arquitectónico representa la estructura de los datos y los componentes del programa que se requieren para construir un sistema basado en computadora. Constituye el estilo arquitectónico que tendrá el sistema, la estructura y las propiedades de los componentes que ese sistema comprende, y las interrelaciones que tienen lugar entre todos los componentes arquitectónicos del sistema.

¿Quién lo hace? Aunque los ingenieros del software pueden diseñar tanto los datos como la arquitectura, cuando se trata de construir sistemas grandes y complejos, el trabajo es, a menudo, asignado a especialistas. El diseñador de una base de datos o un almacén de datos crea la arquitectura de datos para el sistema. El «arquitecto del sistema» selecciona un estilo arquitectónico apropiado a los requisitos derivados durante el análisis de la ingeniería del sistema y de los requisitos del software.

¿Por qué es importante? En la sección *Vistazo rápido* del capítulo anterior preguntamos: «No serías capaz de intentar construir una casa sin un plano, ¿verdad?» Tú tampoco comenzarías el esbozo del plano por los bosquejos del diseño de tuberías de la casa. Necesitarías ver la foto completa —la casa en sí misma— antes de preocuparte por los detalles. Esto es lo que el diseño arquitectónico hace —proporciona la foto completa y asegura que lo estás haciendo bien—.

¿Cuáles son los pasos? El diseño arquitectónico comienza con el diseño de datos y después procede a la derivación de una o más representaciones de la estructura arquitectónica del sistema. Los estilos arquitectónicos alternativos o patrones son analizados con el fin de obtener la estructura que mejor se ajusta a los requisitos del cliente y a las normas de calidad. Una vez que es seleccionada una de las alternativas, la arquitectura se elabora utilizando el método de diseño arquitectónico.

¿Cuál es el producto obtenido? Durante el diseño arquitectónico se crea un modelo de arquitectura que abarca la arquitectura de datos y la estructura del programa. Además, se describen las propiedades de los componentes y sus relaciones (interacciones).

¿Cómo puedo estar seguro de que lo he hecho correctamente? En cada etapa, los productos resultantes del diseño del software son revisados para clarificar, corregir, completar y dar consistencia acorde a los requisitos establecidos, y entre unos y otros.

14.1 ARQUITECTURA DEL SOFTWARE

Shaw y Garlan [SHA96], en su libro de referencia sobre la materia, tratan la arquitectura del software de la siguiente forma:

> Incluso desde que el primer programa fue dividido en módulos, los sistemas de software han tenido arquitecturas, y los programadores han sido responsables de sus interacciones a través de módulos y de las propiedades globales de ensamblaje. Históricamente, las arquitecturas han estado implícitas —bien como accidentes en la implementación, bien como sistemas legados del pasado—. Los buenos desarrolladores de software han adoptado, a menudo, uno o varios patrones arquitectónicos como estrategias de organización del sistema, pero utilizaban estos patrones de modo informal y no tenían ningún interés en hacerlos explícitos en el sistema resultante.

Hoy, la arquitectura de software operativa y su representación y diseño explícitos se han convertido en temas dominantes de la ingeniería de software.

14.1.1. ¿Qué es arquitectura?

Cuando hablamos de la «arquitectura» de un edificio, nos vienen a la cabeza diferentes atributos. A nivel más básico, pensamos en la forma global de la estructura física. Pero, en realidad, la arquitectura es mucho más. Es la forma en la que los diferentes componentes del edificio se integran para formar un todo unido. Es la forma en la que el edificio encaja en su entorno y con los otros edificios de su vecindad. Es el grado en el que el edificio consigue su propósito fijado y satisface las necesidades de sus propietarios. Es el sentido estético de la estructura —el impacto visual del edificio— y el modo en el que las texturas, los colores y los materiales son combinados para crear la fachada externa y el «entorno vivo» interno. Son los pequeños detalles - e l diseño de las instalaciones eléctricas, del tipo de suelo, de la colocación de tapices y una lista casi interminable—. Y, finalmente, es arte.

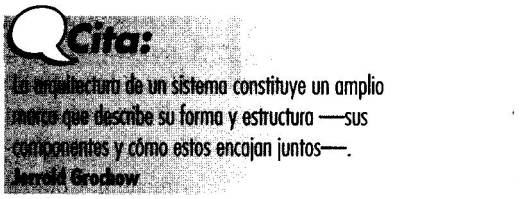
Referencia Web

Se puede encontrar una lista útil de recursos de arquitectura de software en **www2.umassd.edu/SECenter/SAResources.html**

Pero, ¿qué pasa con la arquitectura de software? Bass y sus colegas [BAS98] definen este esquivo término de la siguiente forma:

> La arquitectura de software de un sistema de programa o computación es la estructura de las estructuras del sistema, la cual comprende los componentes del software, las propiedades de esos componentes visibles externamente, y las relaciones entre ellos.

La arquitectura no es el software operacional. Más bien, es la representación que capacita al ingeniero del software para: (1) analizar la efectividad del diseño para la consecución de los requisitos fijados, (2) considerar las alternativas arquitectónicas en una etapa en la cual hacer cambios en el diseño es relativamente fácil, y (3) reducir los riesgos asociados a la construcción del software.



La definición presentada anteriormente enfatiza el papel de los «componentes del software» en cualquier representación arquitectónica. En el contexto del diseño arquitectónico, un componente del software puede ser tan simple como un módulo de programa, pero también puede ser algo tan complicado como incluir bases de datos y software intermedio («*middleware*») que permiten la configuración de una red de clientes y servidores. Las propiedades de los componentes son aquellas características necesarias para entender cómo los componentes interactúan con otros componentes. A nivel arquitectónico, no se especifican las propiedades internas (por ejemplo, detalles de un algoritmo). Las relaciones entre los componentes pueden ser tan sencillas como una llamada de procedimiento de un módulo a otro, o tan complicadas como el protocolo de acceso a bases de datos.

En este libro, el diseño de la arquitectura de software tiene en cuenta dos niveles de la pirámide del diseño (Fig. 13.1)—el diseño de datos y arquitectónico—. En este sentido, el diseño de datos nos facilita la representación de los componentes de datos de la arquitectura. El diseño arquitectónico se centra en la representación de la estructura de los componentes del software, sus propiedades e interacciones.

14.1.2. ¿Por qué es importante la arquitectura?

En su libro dedicado a la arquitectura de software, Bass y sus colegas [BAS98] identifican tres razones clave por las que la arquitectura de software es importante:

- las representaciones de la arquitectura de software facilitan la comunicación entre todas las partes (partícipes) interesadas en el desarrollo de un sistema basado en computadora;
- la arquitectura destaca decisiones tempranas de diseño que tendrán un profundo impacto en todo el trabajo de ingeniería del software que sigue, y es tan importante en el éxito final del sistema como una entidad operacional;
- la arquitectura «constituye un modelo relativamente pequeño e intelectualmente comprensible de cómo está estructurado el sistema y de cómo trabajan juntos sus componentes» [BAS98].

PUNTO CLAVE

El modelo arquitectónico proporciono uno visión «Gestalt» del sistema, permitiendo al ingeniero del software examinarlo como un todo.

El modelo de diseño arquitectónico y los patrones arquitectónicos contenidos dentro son transferibles. Esto es, los estilos y patrones de arquitectura (Sección **14.3.1)** pueden ser aplicados en el diseño de otros sistemas y representados a través de un conjunto de abstracciones que facilitan a los ingenieros del software la descripción de la arquitectura de un modo predecible.

14.2 DISEÑO DE DATOS

Al igual que otras actividades de la ingeniería del software, el *diseño de datos* (a veces llamado *arquitectura de datos*) crea un modelo de datos y/o información que se representa con un alto nivel de abstracción (la visión de datos del cliente/usuario). Este modelo de datos, es entonces refinado en progresivas representaciones especificas de la implementación, que pueden ser procesadas por un sistema basado en computadora. En muchas aplicaciones de software, la arquitectura de datos tendrá una gran influencia sobre la arquitectura del software que debe procesarlo.

La estructura de datos ha sido siempre una parte importante del diseño de software. Al nivel de los componentes del programa, el diseño de las estructuras de datos y de los algoritmos asociados requeridos para su manipulación, son la parte esencial en la creación de aplicaciones de alta calidad. A nivel de aplicación, la traducción de un modelo de datos (derivado como parte de la ingeniería de requisitos) en una base de datos es el punto clave para alcanzar los objetivos de negocio del sistema. **A** nivel de negocios, el conjunto de información almacenada en las diferentes bases de datos y reorganizada en el almacén de datos facilita la minería de datos o el descubrimiento de conocimiento que puede influir en el propio éxito del negocio. De cualquier modo, el diseño de datos juega un papel muy importante.

14.2.1. Modelado de datos, estructuras de datos, bases de datos y almacén de datos

Los objetos de datos definidos durante el análisis de requisitos del software son modelados utilizando diagramas de entidad-relación y el diccionario de datos (Capítulo 12). La actividad de diseño de datos traduce esos elementos del modelo de requisitos en estructuras de datos a nivel de los componentes del software y, cuando es necesario, a arquitectura de base de datos a nivel de aplicación.

Cita:

"La calidad de los datos es lo que marca la diferencia entre un almacén de datos y un basurero de datos."
Jarrett Rosenberg

Durante muchos años, la arquitectura de datos estuvo limitada, generalmente, a las estructuras de datos a nivel del programa y a las bases de datos a nivel de aplicación. Pero hoy, las empresas grandes y pequeñas están inundadas de datos. No es inusual, incluso para una mediana empresa, tener docenas de bases de datos sirviendo diferentes aplicaciones de cientos de gigabytes de datos. El desafío de las empresas ha sido extraer información útil de su entorno de datos, particularmente cuando la información deseada está funcionalmente interrelacionada (por ejemplo, información que sólo puede obtenerse si los datos de marketing específicos se cruzan con los datos de la ingeniería del producto).

Para resolver este desafío, la comunidad de empresas de TI ha desarrollado las técnicas de *minería de datos*, también llamadas *descubrimiento de conocimiento en bases de datos (DCBC),* que navegan a través de las bases de datos en un intento por extraer el nivel de información de negocio apropiado. Sin embargo, la existencia de múltiples bases de datos, sus diferentes estructuras, el grado de detalle contenido en las bases de datos, y muchos otros factores hacen difícil la minería de datos dentro de un entorno con bases de datos existentes. Una solución alternativa, llamada almacén de datos, añade una capa adicional a la arquitectura de datos.

Referencia Web

Para obtener información actualizada sobre tecnologías de olmocén de datos visitar:
www.datawarehouse.com

Un *almacén de datos* es un entorno de datos separado, que no está directamente integrado con las aplicaciones del día a día, pero que abarca todos los datos utilizados por una empresa [MAT96]. En cierto sentido, un almacén de datos es una gran base de datos independiente, que contiene algunos, pero no todos los datos almacenados en las bases de datos que sirven al conjunto de aplicaciones requeridas en un negocio. Sin embargo, hay muchas características diferenciales entre un almacén de datos y una base de datos típica [INM95]:

Orientación por materia. Un almacén de datos está organizado por las materias importantes del negocio, más que por los procesos o funciones del negocio. Esto nos lleva a una exclusión de datos que podrían ser necesarios para una función particular del negocio, pero que generalmente no son necesarios para la minería de datos.

Integración. Sin tener en cuenta la fuente de datos, da consistencia nombrar convenciones, unidades y medidas, estructuras de codificación y atributos físicos incluso cuando la inconsistencia existe a través de las diferentes bases de datos orientadas a aplicaciones.

Restricciones de tiempo. Para un entorno de aplicación orientado a transacción, los datos son precisos en el momento del acceso y por un periodo de tiempo relativamente pequeño (normalmente de 60 a 90 días) antes del acceso. Sin embargo, en un almacén de datos, se accede a los datos en un momento específico del tiempo (por ejemplo, los clientes con los que se ha contactado el mismo día que el nuevo producto ha sido anunciado en la prensa comercial). El horizonte típico de tiempo de un almacén de datos es de 5 a 10 años.

No volatilidad. A diferencia de las típicas bases de datos de aplicaciones de negocios que atraviesan una continua corriente de cambios (insertar, borrar, actualizar), en el almacén de datos los datos se cargan, pero después de la primera transferencia, los datos no cambian.

PUNTO CLAVE

Un almacén de datos contiene todos los datos utilizados en una empresa. Su objetivo es facilitar el acceso a ((conocimiento)) que de otro modo no se dispondría.

Estas características presentan un cuadro Único de desafíos de diseño para el arquitecto de datos.

14.2.2. Diseño de datos a nivel de componentes

El diseño de datos a nivel de componentes se centra en la representación de estructuras de datos a las que se accede directamente a través de uno o más componentes del software. Wasserman [WASSO] ha propuesto un conjunto de principios que pueden emplearse para especificar y diseñar dicha estructura de datos. En realidad, el diseño de datos empieza durante la creación del modelo de análisis. Recordando que el análisis de requisitos y el diseño a menudo se solapan, consideramos el siguiente conjunto de principios [WASSO] para la especificación de datos:

1. *Los principios del análisis sistemático aplicados a la función y al comportamiento deberían aplicarse también a los datos.* Invertimos mucho tiempo y esfuerzo en obtener, revisar y especificar los requisitos funcionales y el diseño preliminar. Las representaciones de flujo de datos y contenido deberían desarrollarse y revisarse, las de objetos de datos deberían identificarse, se deberían estudiar las organizaciones alternativas de datos y debería evaluarse el impacto del modelado de datos sobre el diseño del software. Por ejemplo, la especificación de una lista enlazada con múltiples anillos puede satisfacer los requisitos de los datos pero puede llevar a un diseño del software poco flexible. Una organización de datos alternativa nos podría llevar a obtener mejores resultados.

¿Qué principios son aplicables para el diseño de datos?

2. *Todas las estructuras de datos y las operaciones a llevar a cabo en cada una de ellas deberían estar claramente identificadas.* El diseño de una estructura de datos eficaz debe tener en cuenta las operaciones que se van a llevar a cabo sobre dicha estructura de datos (vea [AHO83]). Por ejemplo, imagínese una estructura de datos hecha con un conjunto de elementos de datos diversos. La estructura de datos se va a manipular con varias funciones principales del software. Después de la evaluación de la operación realizada sobre la estructura de datos, se define un tipo de datos abstracto para usarlo en el diseño del software subsiguiente. La especificación de los tipos de datos abstractos puede simplificar considerablemente el diseño del software.
3. *Se debería establecer un diccionario de datos y usarlo para definir el diseño de los datos y del programa.* El concepto de diccionario de datos se introdujo en el Capítulo 12. Un diccionario de datos representa explícitamente las relaciones entre los objetos de datos y las restricciones de los elementos de una estructura de datos. Los algoritmos que deben aprovecharse de estas relaciones específicas pueden definirse más fácilmente si existe una especificación de datos tipo diccionario.
4. *Las decisiones de diseño de datos de bajo nivel deberían dejarse para el final del proceso de diseño.* Se puede usar un proceso de refinamiento paso a paso para el diseño de datos. Es decir, la organización general de datos puede definirse durante el análisis de requisitos; refinarse durante los trabajos de diseño de datos y especificarse en detalle durante el diseño a nivel de componentes. El enfoque descendente del diseño de data proporciona ventajas análogas al enfoque descendente del diseño de software: se diseñan y evalúan primeramente los atributos estructurales principales de manera que se pueda establecer la arquitectura de los datos.
5. *La representación de las estructuras de datos deberían conocerla sólo aquellos módulos que deban hacer uso directo de los datos contenidos dentro de la estructura.* El concepto de ocultación de información y el concepto relacionado de acoplamiento (Capítulo 13) proporciona una importante visión de la calidad del diseño del software. Este principio alude a la importancia de estos conceptos así como

a «la importancia de separar la visión lógica de un objeto de datos de su visión física» [WASSO].

6. *Debería desarrollarse una biblioteca de estructuras de datos útiles y de las operaciones que se les pueden aplicar.* Las estructuras de datos y las operaciones deberían considerarse como recursos para el diseño del software. Las estructuras de datos pueden diseñarse para que se puedan reutilizar. Una biblioteca de plantillas de estructuras de datos (tipos abstractos de datos) puede reducir el esfuerzo del diseño y de la especificación de datos.
7. *Un diseño del software y un lenguaje de programación debería soportar la especificación y realización de los tipos abstractos de datos.* La implementación de una estructura de datos sofisticada puede hacerse excesivamente difícil si no existen los medios de especificación directa de la estructura en el lenguaje de programación escogido para dicha implementación.

Los principios descritos anteriormente forman una base para un enfoque de diseño de datos a nivel de componentes que puede integrarse en las fases de análisis y diseño.

14.3 ESTILOS ARQUITECTÓNICOS

Cuando un constructor utiliza el término *«centre hall colonial»* para describir una casa, la mayoría de la gente familiarizada con las casas en los Estados Unidos sería capaz de recrear una imagen general de cómo sería esa casa y de cómo sena su diseño de plantas. El constructor ha utilizado un *estilo arquitectónico* a modo de mecanismo descriptivo para diferenciar esa casa de otros estilos (por ejemplo, *A--ame, raised ranch, Cape Cod*). Pero, lo más importante, es que el estilo arquitectónico es también un patrón de construcción. Más adelante se deberán definir los detalles de la casa, se especificarán sus dimensiones finales, se le añadirán características personalizadas y se determinarán los materiales de construcción, pero el patrón --*«centre hall colonial»*— guía al constructor en su trabajo.

Referencia Web

El proyecto ABLE de lo CMU cuenta con trabajos y ejemplos muy útiles sobre estilos arquitectónicos: **tom.cs.cmu.edu/able/**

El software construido para sistemas basados en computadoras también cuenta con diversos estilos arquitectónicos[1]. Cada estilo describe una categoría del sistema que contiene: (1) un conjunto de *componentes* (por ejemplo, una base de datos, módulos computacionales) que realizan una función requerida por el sistema; (2) un conjunto de *conectores* que posibilitan la «comunicación, la coordinación y la cooperación» entre los componentes; (3) *restricciones* que definen cómo se pueden integrar los componentes que forman el sistema; y (4) *modelos semánticos* que permiten al diseñador entender las propiedades globales de un sistema para analizar las propiedades conocidas de sus partes constituyentes [BAS98]. En la siguiente sección, abordamos los patrones arquitectónicos comúnmente utilizados para el software.

¿Qué es un estilo arquitectónico

14.3.1. Una breve taxonomía de estilos y patrones

Aunque durante los pasados 50 años se han creado cientos de miles de sistemas basados en computadora, la gran mayoría pueden ser clasificados (ver [SHA96], [BAS98], [BUS98]) dentro de uno de los estilos arquitectónicos:

Arquitecturas centradas de datos. En el centro de esta arquitectura se encuentra un almacén de datos (por ejemplo, un documento o una base de datos) al que otros componentes acceden con frecuencia para actualizar, añadir, borrar o bien modificar los datos del almacén. La figura **14.1** representa un estilo típico basada en los datos. El software de cliente accede a un almacén centrai. En algunos casos, el almacén de datos es *pasivo.* Esto significa que el software de cliente accede a los datos independientemente de cualquier cambio en los datos o de las acciones de otro software de cliente. Una variación en este acceso transforma el almacén en una «pizarra» que envía notificaciones al software de cliente cuando los datos de interés del cliente cambian.

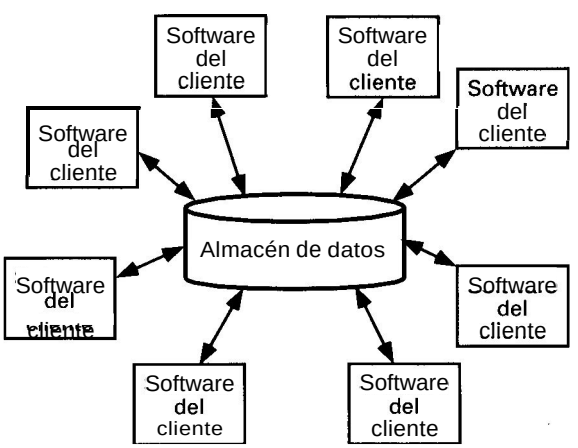


FIGURA 14.1. Arquitectura basada en los datos.

[1] Los términos *estilos* y *patrones* se utilizan indistintamente en este capítulo.

Las arquitecturas basadas en **los** datos promueven la capacidad de integración (integrability) [BAS98]. Por consiguiente, **los** componentes existentes pueden cambiarse o **los** componentes del nuevo cliente pueden añadirse a la arquitectura sin involucrar a otros clientes (porque **los** componentes del cliente operan independientemente). Además, **los** datos pueden ser transferidos entre **los** clientes utilizando un mecanismo de pizarra (por ejemplo, el componente de pizarra sirve para coordinar la transferencia de información entre clientes). Los componentes cliente son procesos ejecutados independientemente.

Cita:
El uso de patrones y estilos de diseño está integrado en las disciplinas de ingeniería.
Mary Shaw y David Garlan

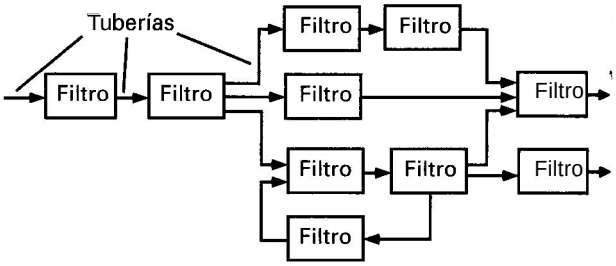
Arquitecturas de flujo de datos. Esta arquitectura se aplica cuando **los** datos de entrada son transformados a través de una serie de componentes *computacionales* o *manipulativos* en los datos de salida. Un patrón tubería y filtro (Fig. 14.2.a) tiene un grupo de componentes, llamados filtros, conectados por tuberías que transmiten datos de un componente al siguiente. Cada filtro trabaja independientemente de aquellos componentes que se encuentran en el flujo de entrada o de salida; está diseñado para recibir la entrada de datos de una cierta forma y producir una salida de datos (hacia el siguiente filtro) de una forma específica. Sin embargo, el filtro no necesita conocer el trabajo de **los** filtros vecinos.

Si el flujo de datos degenera en una simple línea de transformadores (Fig. 14.2.b) se le denomina *secuencia1 por lotes.* Este patrón aplica una serie de componentes secuenciales (filtros) para transformarlos.

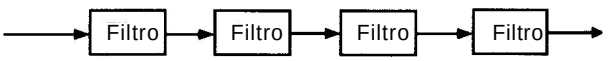
Cita:
Un buen arquitecto es el principal mantenedor de la visión del usuario del producto final.
Norman Simenson

Arquitecturas de llamada y retorno. Este estilo arquitectónico permite al diseñador del software (arquitecto del sistema) construir una estructura de programa relativamente fácil de modificar y ajustar a escala. Existen dos subestilos [BAS98] dentro de esta categoría:

- *arquitecturas de programa principallsuhprograma:* esta estructura clásica de programación descompone las funciones en una jerarquía de control donde un programa «principal» llama a un número de componentes del programa, **los** cuales, en respuesta, pueden también llamar a otros componentes. La Figura 13.3 representa una arquitectura de este tipo.
- *arquitecturas de llamada de procedimiento remoto:* los componentes de una arquitectura de programa principal/subprograma, están distribuidos entre varias computadoras en una red.



(a) Tuberías y filtros



(b) Secuencial por lotes

FIGURA 14.2. Arquitecturas de flujo de datos.

Referencia cruzada
En la Parte 4 se presenta un estudio detallado sobre las arquitecturas orientadas a objetos.

Arquitecturas orientadas a objetos. Los componentes de un sistema encapsulan **los** datos y las operaciones que se deben realizar para manipular **los** datos. La comunicación y la coordinación entre componentes se consigue a través del paso de mensajes.

Arquitecturas estratificadas. La estructura básica de una arquitectura estratificada se representa en la Figura 14.3. Se crean diferentes capas y cada una realiza operaciones que progresivamente se aproximan más al cuadro de instrucciones de la máquina. En la capa externa, **los** componentes sirven a las operaciones de interfaz de usuario. En la capa interna, **los** componentes realizan operaciones de interfaz del sistema. Las capas intermedias proporcionan servicios de utilidad y funciones del software de aplicaciones.

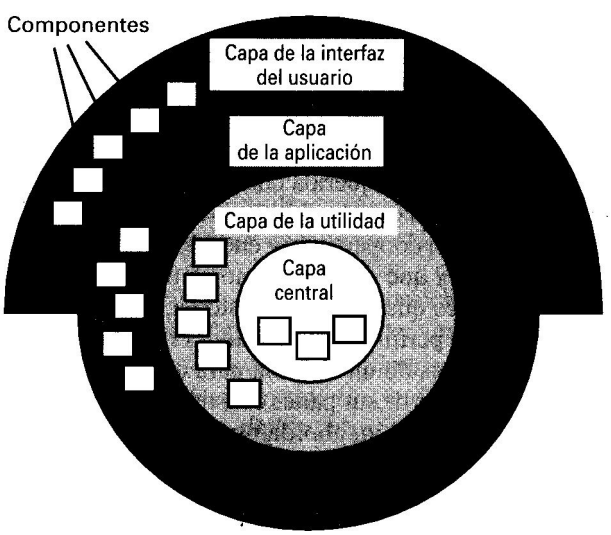


FIGURA 14.3. Arquitectura ectratificada.

Los estilos arquitectónicos citados anteriormente son sólo una pequeña parte de los que dispone el diseñador de software[2]. Una vez que la ingeniería de requisitos define las características y las restricciones del sistema que ha de ser construido, se escoge el patrón arquitectónico (estilo) o la combinación de patrones (estilos) que mejor encajan con las características y restricciones. En muchos casos, puede ser apropiado más de un patrón y se podrían diseñar y evaluar estilos arquitectónicos alternativos.

14.3.2. Organización y refinamiento

Puesto que el proceso de diseño deja a menudo al ingeniero con un número de alternativas arquitectónicas, es importante establecer un conjunto de criterios de diseño que puedan ser utilizados para evaluar un diseño arquitectónico derivado. Las siguientes cuestiones [BAS98] proporcionan una idea del estilo arquitectónico que ha sido derivado:

Control. ¿Cómo se gestiona el control dentro de la arquitectura? ¿Existe una jerarquía de control diferente, y si es así, cuál es el papel de los componentes dentro de esa jerarquía de control? ¿Cómo transfieren el control los componentes dentro del sistema? ¿Cómo se comparte el control entre componentes? ¿Cuál es la topología de control (por ejemplo, la forma geométrica[3] que adopta el control)?, ¿Está el control sincronizado o los componentes actúan asincrónicamente?

¿Cómo se evalúa un estilo arquitectónico que se ha derivado?

Datos. ¿Cómo se comunican los datos entre componentes? ¿El flujo de datos es continuo o son objetos de datos que pasan de un componente a otro, o los datos están disponibles globalmente para ser compartidos entre **los** componentes del sistema? ¿Existen los componentes de datos (por ejemplo, una pizarra o almacén), y **si** es así, cuál es su papel? ¿Cómo interactúan los componentes funcionales con **los** componentes de datos? ¿Los componentes de datos son activos o pasivos (por ejemplo, los componentes de datos interactúan activamente con otros componentes del sistema)? ¿Cómo interactúan **los** datos y el control dentro del sistema?

Estas preguntas proporcionan al diseñador una evaluación temprana de la calidad del diseño y sientan las bases para un análisis más detallado de la arquitectura.

14.4 ANÁLISIS DE DISEÑOS ARQUITECTÓNICOS ALTERNATIVOS

Las preguntas citadas en la sección anterior proporcionan una evaluación preliminar del estilo arquitectónico escogido para un sistema concreto. Sin embargo, para la consecución efectiva del diseño, se necesita un método de evaluación de la calidad de la arquitectura más completo. En las siguientes secciones, consideramos dos enfoques diferentes para el análisis de diseños arquitectónicos alternativos. El primer enfoque utiliza un método iterativo para evaluar el diseño de los compromisos. El segundo enfoque aplica una pseudotécnica cuantitativa para evaluar la calidad del diseño.

Cita:
Igual está en el sótano, déjame ir arriba y mirar.
M.C. Escher

14.4.1. Un método de análisis de compromiso para la arquitectura

El *Instituto de Ingeniería de Software (SEI)* ha desarrollado un *Método de análisis de compromiso para la arquitectura* (MACA) [KAZ98] que establece un proceso de evaluación iterativo para las arquitecturas de software. Las actividades de análisis de diseño mencionadas abajo se realizan iterativamente:

1. *Recopilación de escenarios.* En el Capítulo 11 se recogen un grupo de casos de uso para representar el sistema desde el punto de vista del usuario.
2. *Elicitación de requisitos.* Esta información es requerida como parte de los requisitos de ingeniería y se utiliza para asegurarse de que todos los clientes, usuarios y partícipes implicados han sido atendidos.

Referencia Web
Se puede encontrar información detallada sobre análisis de compromiso de software arquitectónico en:
www.sei.cmu.edu/ata/ata_method.html

3. *Describir los patrones/estilos arquitectónicos escogidos para derivar los escenarios y requisitos.* El(los) estilo(s) debería describirse utilizando vistas arquitectónicas como:
   - *vista de módulo:* para analizar el trabajo asignado por componente y el grado de ocultación de información que se ha alcanzado
   - *vista de proceso:* para analizar la actuación del sistema
   - *vista de flujo de datos:* para analizar el grado en el que la arquitectura cumple los requisitos funcionales

Referencia cruzada
En los Capítulos 8 y 19 se estudian los atributos de calidad.

[2] Ver [SHA97], [BAS98] y [BUS98] para un estudio detallado de estilos y patrones arquitectónicos.

[3] Una jerarquía es una forma geométrica, pero también podemos encontrar mecanismos de control semejantes en un sistema cliente/servidor.

4. *Evaluar los atributos de calidad considerando cada atributo de forma aislada.* El número de atributos de calidad escogidos para el análisis depende del tiempo disponible para la revisión y del grado de relevancia de dichos atributos para el sistema actual. Los atributos de calidad para la evaluación del diseño arquitectónico incluyen: fiabilidad, rendimiento, seguridad, facilidad de mantenimiento, flexibilidad, capacidad de prueba, movilidad, reutilización e interoperatibilidad.

5. *Identificar la sensibilidad de los atributos de calidad con los diferentes atributos arquitectónicos en un estilo arquitectónico específico.* Esto se puede conseguir realizando pequeños cambios en la arquitectura y determinando cuán sensibles al cambio son los atributos de calidad, como el rendimiento. Aquellos atributos afectados significativamente por un cambio en la arquitectura son denominados *puntos sensibles.*

6. *Análisis de las arquitecturas candidatas (desarrolladas en el paso 3) utilizando el análisis de sensibilidad del paso 5.* El SEI describe este enfoque de la siguiente forma [KAZ98]:

> Una vez determinados los puntos sensibles de la arquitectura, encontrar los puntos de compromiso consiste simplemente en la identificación de los elementos arquitectónicos en los cuales **varios** atributos son sensibles. **Por** ejemplo, el rendimiento de una arquitectura cliente/servidor sena muy sensible al número de servidores (el rendimiento aumenta, dentro de un grado, aumentando en número de servidores). La disponibilidad de esa arquitectura también variaría directamente con el número de servidores. Sin embargo, la seguridad del sistema variaría inversamente al número de servidores (porque el sistema contiene más puntos de ataque potenciales). El número de servidores, entonces, es un punto de compromiso con respecto a la arquitectura. Este es **un** elemento, potencialmente uno de muchos, donde se harán los compromisos arquitectónicos, consciente o inconscientemente.

Estos seis pasos representan la primera iteración MACA. Tras los resultados de los pasos 5 y 6 algunas arquitecturas alternativas se eliminarían, una o varias de las arquitecturas restantes serían modificadas y presentadas con mayor detalle, y después los pasos MACA se aplicarían de nuevo.

14.4.2. Guía cuantitativa para el diseño arquitectónico

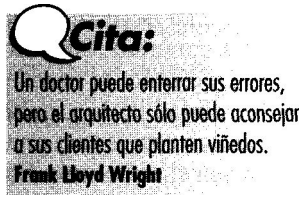
Uno de los muchos problemas con los que se enfrentan los ingenieros del software durante el proceso de diseño es la carencia general de métodos cuantitativos para la evaluación de la calidad de los diseños propuestos. El enfoque MACA recogido en la Sección 14.4.1 representa un enfoque innegablemente cualitativo y útil para el análisis del diseño.

Asada [SHA96] propone varios modelos simples **para** ayudar al diseñador a determinar el grado al cual una arquitectura particular alcanza los criterios de «bondad» predefinidos. Estos criterios, a veces llamados *dimensiones del diseño,* a menudo abarcan los atributos de calidad definidos en la sección anterior: fiabilidad, rendimiento, seguridad, facilidad de mantenimiento, flexibilidad, capacidad de prueba, movilidad, reutilización e interoperatibilidad, entre otros.

El primer modelo, denominado *análisis del espectro,* evalúa un diseño arquitectónico en un espectro de «bondad» desde el mejor diseño posible hasta el peor. Una vez que la arquitectura del software ha sido propuesta, se analiza asignando una «puntuación» a cada una de las dimensiones del diseño. Estas notas de las dimensiones se suman para determinar la calificación total, S, del diseño como un todo. Los casos de notas más bajas[4] son asignados a un diseño hipotético, y se computa la **suma** de notas de la peor arquitectura, S_w. La mejor nota, S_b, se computa para un diseño óptimo[5]. Entonces calculamos el *índice del espectro,* I_s, mediante la ecuación:

$$I_s = [(S - S_w)/(S_b - S_w)] \times 100$$

El índice del espectro indica el grado al cual la arquitectura propuesta se aproxima al sistema óptimo dentro del espectro de alternativas razonables para el diseño.



Si se realizan modificaciones del diseño propuesto o si se propone un nuevo diseño entero, los índices de espectro de ambos deben ser comparados y se computará un *índice de mejora,* I_{me}:

$$I_{me} = I_{s1} - I_{s2}$$

Esto proporciona al diseñador una indicación relativa a la mejora asociada con los cambios arquitectónicos realizados o con la nueva arquitectura propuesta. Si I_{me} es positivo, entonces podemos concluir que el sistema 1 ha sido mejorado en relación al sistema 2.

El *análisis de selección del diseño* es otro modelo que requiere un cuadro de dimensiones de diseño **para** ser definido. La arquitectura propuesta es entonces evaluada para determinar el número de dimensiones del diseño que se logran cuando se compara con un sistema ideal (el mejor caso). Por ejemplo, si una arquitectura propuesta alcanzara una reutilización de componentes excelente, y esta dimensión es requerida para el sistema ideal, la dimensión de reutilización ha

[4] El diseño debe ser todavía aplicable al problema actual, incluso si la solución no es particularmente buena.

[5] El diseño seria Óptimo, pero las restricciones, los costes u otros factores no permitirán su construcción.

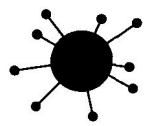
sido lograda. Si la arquitectura propuesta tiene poca seguridad, y se necesita una gran seguridad, no se ha alcanzado la dimensión del diseño.

Calculamos el *índice de selección del diseño, d*, como:

$$d=(N_s/N_a)\times 100$$

donde N_s es el número de dimensiones de diseño logradas en la arquitectura propuesta y N_a es el número total de dimensiones en el espacio de diseño. A mayor índice de selección del diseño, más cerca estamos de que la arquitectura propuesta alcance el sistema ideal.

El *análisis de contribución* «identifica las razones por las que un grupo de alternativas de diseño obtiene unas notas menores que otro». [SHA96] Retomando el estudio del *despliegue de la función de calidad (DFC)* visto en el Capítulo 11, el análisis de valor se realiza para determinar la prioridad relativa de los requisitos determinados durante el despliegue de funciones, el despliegue de información y el despliegue de tareas. Se identifica un conjunto de «mecanismos de realización» (características de la arquitectura). Se crea un listado con todos los requisitos del cliente (determinados a través del DFC) y una matriz de referencias cruzadas. Las celdas de la matriz indican (en una escala del 1 al 10) la fuerza relativa de la relación entre un mecanismo de realización y un requisito para cada arquitectura propuesta. A esto se le conoce como *espacio de diseño cuantificado (EDC)*.



Hoja de cálculo DFC

El EDC es bastante fácil de implementar como un modelo de hoja de cálculo y puede ser utilizado para aislar porqué un cuadro de alternativas de diseño obtiene unas notas menores que otro.

14.4.3. Complejidad arquitectónica

Para evaluar la complejidad total de una arquitectura dada, una técnica útil consiste en considerar las relaciones de dependencia entre los componentes de la arquitectura. Estas relaciones de dependencia son conducidas a través de los flujos de información/control dentro del sistema.

Zhao [ZHA98] propone tres tipos de dependencia:

Dependencias de compartimiento, representan las relaciones de dependencia entre los consumidores que utilizan los mismos recursos o los productores que producen para los mismos consumidores. Por ejemplo, tenemos dos componentes u y v, si *u* y v se refieren a los mismos datos globales, entonces existe una dependencia de compartimiento entre ambos.

Dependencias de flujo, representan las relaciones de dependencia entre los productores y los consumidores de recursos. Por ejemplo, entre dos componentes u y v, si u debe completarse antes de que el control fluya a *v* (prerrequisito), o *si* u y *v* se comunican a través de parámetros, entonces existirá una relación de dependencia de flujo entre ambos.

Dependencias restrictivas, representan las restricciones de un relativo flujo de control entre un cuadro de actividades. Por ejemplo, dos componentes u y v, si u *y v* no se pueden ejecutar al mismo tiempo (por exclusión mutua), entonces existirá una dependencia restrictiva entre u y *v*.

Las dependencias de compartimiento y de flujo recogidas por Zhao se parecen de alguna forma al concepto de acoplamiento visto en el Capítulo 13. En el Capítulo 19 se explicarán mediciones sencillas para la evaluación de las dependencias.

14.5 CONVERSIÓN DE LOS REQUISITOS EN UNA ARQUITECTURA DEL SOFTWARE

En el Capítulo 13 ya dijimos que los requisitos del software pueden ser convertidos en varias representaciones del modelo de diseño. Los estilos arquitectónicos vistos en la Sección 14.3.1 representan arquitecturas radicalmente diferentes, por lo cual no deberíamos sorprendemos que no exista una única conversión que logre una transición del modelo de requisitos al modelo de diseño. De hecho, todavía no existe una conversión para algunos modelos arquitectónicos, y el diseñador debe lograr la traducción de los requisitos del diseño para esos estilos de una forma *ad hoc*.

Para ilustrar un enfoque de conversión arquitectónica, consideraremos la arquitectura de llamada y retorno —una estructura muy común para diferentes tipos de sistemas[6]—. La técnica de conversión que se presenta capacita al diseñador para derivar arquitecturas de llamada y retorno razonablemente complejas en diagramas de flujo de datos dentro del modelo de requisitos.

La técnica, a veces denominada *diseño estructurado*, tiene sus orígenes en antiguos conceptos de diseño que defendían la modularidad [DEN73], el diseño descendente [WIR71] y la programación estructurada

[6] También es importante mencionar que las arquitecturas de llamada y retorno pueden residir dentro de otras arquitecturas mas sofisticadas vistas anteriormente en este capitulo. Por ejemplo, la arquitectura de uno o vanos componentes de una arquitectura cliente/servidor podría ser de llamada y retorno.

[DAH72, LIN79]. Stevens, Myers y Constantine [STE74] propusieron el diseño del software basado en el fluyo de datos a través de un sistema. Los primeros trabajos se refinaron y se presentaron en libros de Myers [MYE78], Yourdon y Constantine [YOU79].

¿Cuáles son los pasos a seguir en la evaluación del DFD en una arquitectura de llamada y retorno?

El diseño estructurado suele caracterizarse como un método de diseño orientado al flujo de datos porque permite una cómoda transición desde el *diagrama de flujo de datos (DFD)* a la arquitectura de software[7]. La transición desde el flujo de información (representado como un diagrama de flujo de datos) a una estructura del programa se realiza en un proceso de seis pasos: (1) se establece el tipo de flujo de información; (2) se indican los límites del flujo; **(3)** se convierte el DFD en la estructura del programa; (4) se define la jerarquía de control; *(5)* se refina la estructura resultante usando medidas y heur'sticas de diseño, y (6) se refina y elabora la descripción arquitectónica.

El tipo de flujo de información es lo que determina el método de conversión requerido en el paso **3.** En las siguientes secciones examinamos los dos tipos de flujo.

14.5.1. Flujo de transformación

Retomando el modelo fundamental de sistema (nivel 0 del diagrama de flujo de datos), la información debe introducirse y obtenerse del software en forma de «mundo exterior». Por ejemplo, los datos escritos con un teclado, los tonos en una línea telefónica, y las imágenes de vídeo en una aplicación multimedia son todas formas de información del mundo exterior. Tales datos internos deben convertirse a un formato interno para el procesamiento. La información entra en el sistema a lo largo de caminos que transforman los datos externos a un formato interno. Estos caminos se identifican comoflujo *de entrada.* En el interior del software se produce una transición. La información entrante se pasa a través de un *centro de transformación* y empieza a moverse a lo largo de caminos que ahora conducen hacia «fuera» del software. Los datos que se mueven a lo largo de estos caminos se denominan *flujo de salida.* **El** flujo general de datos ocurre de manera secuencial y sigue un, o unos pocos, caminos[8] «directos». Cuando un segmento de un diagrama de flujo de datos presenta estas características, lo que tenemos presente es un *flujo de transformación.*

Referencia cruzada

Los diagramas de flujo de datos se estudian en detalle en el Capítulo 12.

14.5.2. Flujo de transacción

El modelo fundamental de sistema implica un flujo de transformación; por tanto, es posible caracterizar todo el flujo de datos en esta categoría. Sin embargo, el flujo de información está caracterizado a menudo por un único elemento de datos, denominado *transacción,* que desencadena otros flujos de información a lo largo de uno de los muchos caminos posibles. Cuando un **DFD** toma la forma mostrada en la Figura 14.4, lo que tenemos es un *flujo de transacción.*

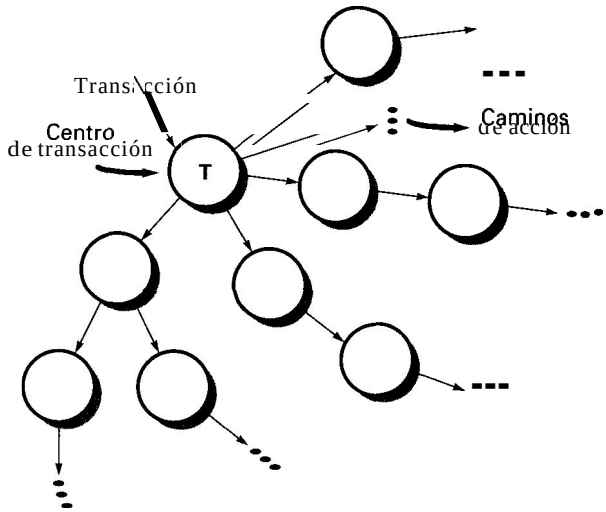


FIGURA 14.4. Flujo de transacción.

El flujo de transacción se caracteriza por datos que se mueven a lo largo de un camino de entrada que convierte la información del mundo exterior en una transacción. La transacción se evalúa y, basándose en ese valor, se inicia el flujo a lo largo de uno de muchos *caminos de acción.* El centro del flujo de información del que parten muchos de los caminos de acción se denomina *centro de transacción.*

Debería recalcarse que dentro del DFD de un sistema grande, ambos flujos de transacción y de transformación pueden presentarse. Por ejemplo, en un flujo orientado a transacción, el flujo de información a lo largo de un camino de acción puede tener características de flujo de transformación.

[7] Debe recordarse que también durante el modelado de análisis se utilizan otros elementos del método de análisis (por ejemplo; el diccionario de datos, EP, EC).

[8] Un análisis obvio de este tipo de flujo de información lo encontramos en la Sección 14.3.1 en la arquitectura de flujo de datos. **Sin** embargo, hay muchos casos en los que la arquitectura de flujo de datos no sería la mejor elección para un sistema complejo. Los ejemplos incluyen sistemas que sufrirían cambios sustanciales en el tiempo, o sistemas en los cuales el procesamiento asociado con el flujo de datos no es necesariamente secuencial.

14.6 ANÁLISIS DE LAS TRANSFORMACIONES

El *análisis de las transformaciones* es un conjunto de pasos de diseño que permite convertir un DFD, con características de flujo de transformación, en un estilo arquitectónico específico. En esta sección se describe el análisis de las transformaciones aplicando los pasos de diseño a un sistema de, por ejemplo, una parte del software de *Hogarseguro* presentado en capítulos anteriores.

14.6.1. Un ejemplo

El sistema de seguridad *Hogarseguro*, presentado anteriormente en este libro, es representativo de muchos productos y sistemas basados en computadora actualmente en uso. El producto vigila el mundo real y reacciona ante cambios que encuentra. También interacciona con el usuario a través de una serie de entradas por teclado y visualizaciones alfanuméricas. El nivel 0 del diagrama de flujo de datos de *Hogarseguro*, reproducido del Capítulo 12, se muestra en la Figura 14.5.

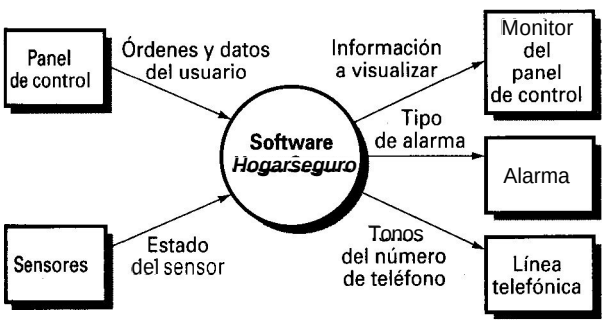


FIGURA 14.5. DFD a nivel de contexto **para** *Hogarseguro.*

Durante el análisis de requisitos, se habrán creado más modelos detallados del flujo para *Hogarseguro*. Además, se crearán las especificaciones de control y proceso, un diccionario de datos y varios modelos de comportamiento.

14.6.2. Pasos del diseño

El ejemplo anterior se usará para ilustrar cada paso en el análisis de las transformaciones. Los pasos empiezan con una reevaluación del trabajo hecho durante el análisis de requisitos y después evoluciona hacia el diseño de la arquitectura del software.

Paso 1. Revisar el modelo fundamental del sistema. El modelo fundamental del sistema comprende el DFD de nivel 0 y la información que lo soporta. En realidad, el paso de diseño empieza con una evaluación de la *especificación del sistema* y de la *especificación de requisitos del software.* Ambos documentos describen el flujo y la estructura de la información en la interfaz del software. Las Figuras 14.5 y 14.6 muestran el nivel 0 y 1 del flujo de datos del software *Hogarseguro.*

Paso 2. Revisar y refinar los diagramas de flujo de datos del software. La información obtenida de los modelos de análisis contenidos en la *Especificación de Requisitos del Software* se refina para obtener mayor detalle. Por ejemplo, se examina el DFD del nivel 2 de *monitorizar sensores* (Fig. 14.7) y se obtiene un diagrama de flujo de datos de nivel 3 como se muestra en la Figura 14.8. En el nivel 3, cada transformación en el diagrama de flujo de datos presenta una cohesión relativamente alta (Capítulo 13). Es decir, el proceso implicado por una transformación realiza una función única y distinta que puede implementarse como un módulo[9] en el software *HogarSeguro.* Por tanto, el DFD de la Figura 14.8 contiene suficiente detalle para establecer un diseño «inicial» de la arquitectura del subsistema de monitorizar sensores y continuamos sin más refinamiento.

Si el *DFD* es *refinado más* de uno vez, se *esforzará* por *derivar burbujas* que *presentan gran* cohesión.

Paso 3. Determinar si el DFD tiene características de flujo de transformación o de transacción. En general, el flujo de información dentro de un sistema puede representarse siempre como una transformación. Sin embargo, cuando se encuentra una característica obvia de transacción (Fig. 14.4), se recomienda una estructura de diseño diferente. En este paso, el diseñador selecciona la característica general del flujo (de la amplitud del software) basándose en la propia naturaleza del DFD. Además, se aíslan regiones locales de flujo de transformación o de transacción. Estos *subflujos* pueden usarse para refinar la estructura del programa obtenida por la característica general que prevalece. Por ahora, concentraremos nuestra atención solamente en el flujo de datos del subsistema de monitorización de sensores mostrado en la Figura 14.7.

A menudo se podrán encontrar ambos tipos de flujo de datos dentro del mismo modelo de análisis. tos flujos se dividen y la estructura del programa se deriva utilizando el análisis adecuado.

9 La utilización del término módulo en este capítulo equivale al término componente usado anteriormente en el estudio de la arquitectura de software.

Evaluando el DFD (Fig. 14.8), vemos que los datos entran al software por un camino de entrada y salen por tres caminos de salida. No se distingue ningún centro de transacción (aunque la transformación *establecer las condiciones de alarma* podría percibirse como tal). Por tanto, se asumirá una característica general de transformación para el flujo de información.

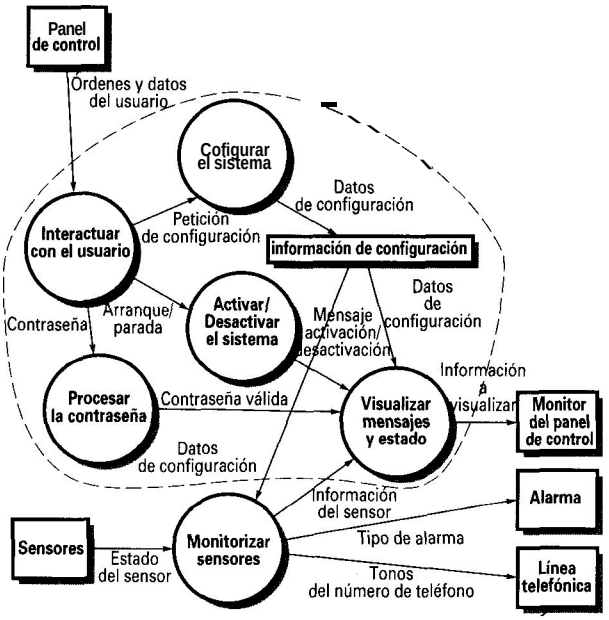


FIGURA 14.6. Nivel 1 del DFD del *HogarSeguro.*

Paso 4. Aislar el centro de transformación especificando los límites de los flujos de entrada y salida. En la sección precedente el flujo de entrada se describió como un camino en el que la información se convierte de formato externo en interno; el flujo de salida convierte de formato interno a externo. Los límites del flujo de entrada y el de salida son interpretables. Es decir, los diseñadores pueden elegir puntos ligeramente diferentes como límites de flujo. De hecho, se pueden obtener soluciones de diseño alternativas variando la posición de los límites de flujo. Aunque hay que tener cuidado cuando se seleccionan los límites, una variación de una burbuja a lo largo de un camino de flujo tendrá generalmente poco impacto en la estructura final del programa.

Cambia la situación de los fronteras de flujo en un esfuerzo por explorar los estructuras de programa alternativas. No llevo mucho tiempo y puede proporcionarnos importantes ideas.

Los límites de flujo del ejemplo se ilustran como curvas sombreadas que van verticales a través del flujo en la Figura 14.8. Las transformaciones (burbujas) que forman el centro de transformación se encuentran entre los dos límites sombreados que van de arriba abajo en el dibujo. Se podría argumentar algún cambio para reajustar los límites (por ejemplo, se podría proponer un límite del flujo de entrada que separe *leer sensores,* de *adquirir información de respuesta).* El énfasis en este paso del diseño debería ponerse en seleccionar límites razonables, en vez de largas disquisiciones sobre la colocación de las separaciones.

Paso 5. Realizar una «descomposición de primer nivel». La estructura del programa representa una distribución descendente del control. La descomposición en partes provoca una estructura de programa en la que los módulos del nivel superior realizan la toma de decisiones y los módulos del nivel inferior realizan la mayoría del trabajo de entrada, cálculos y salida. Los módulos de nivel intermedio realizan algún control y cantidades moderadas de trabajo.

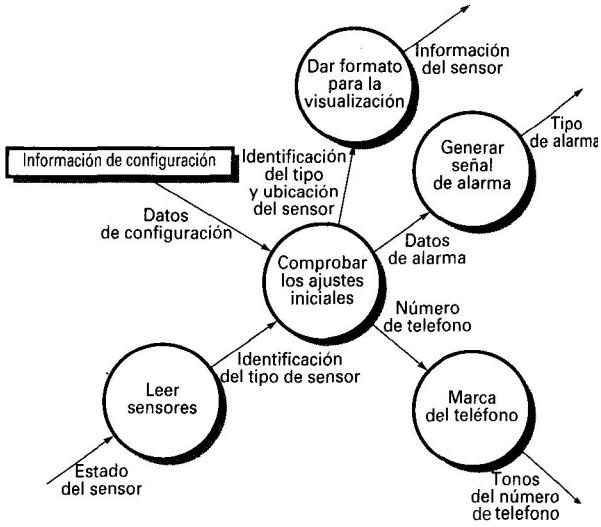


FIGURA 14.7. Nivel 2 del DFD que refina el proceso de monitorizar sensores

Cuando encontramos un flujo de transformación, un DFD se organiza en una estructura específica (una arquitectura de llamada y retorno) que proporciona control para el procesamiento de información de entrada, transformación y de la salida. Esta descomposición en factores o primer nivel del subsistema de monitorizar sensores se ilustra en la Figura **14.9.** Un controlador principal, denominado **gestor de monitorización de sensores,** reside en la parte superior de la estructura del programa y sirve para coordinar las siguientes funciones de control subordinadas:

- un controlador de procesamiento de la información de entrada denominado controlador de la entrada del sensor, coordina la recepción de todos los datos de entrada;
- un controlador del flujo de transformación, denominado controlador de las condiciones de alarma, supervisa todas las operaciones sobre los datos en su forma interna (por ejemplo; un módulo **que** invoca varios procedimientos de transformación de datos);

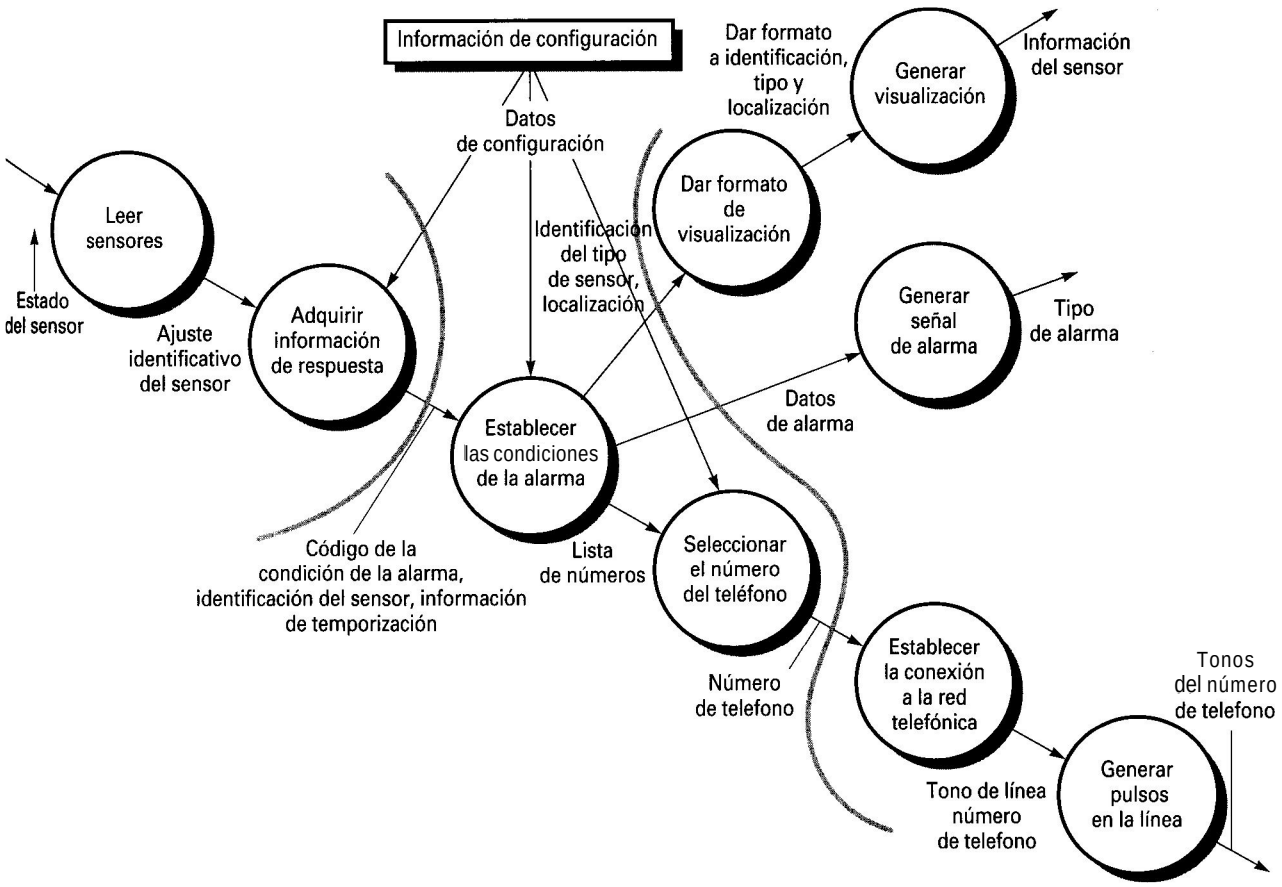


FIGURA 14.8. Nivel 3 del DFD de *monitorizar sensores* con los límites de flujo.

- un controlador de procesamiento de información de salida, denominado controlador de la salida de alarma, coordina la producción de la información de salida.

No seas dogmático en esta parte. Podría ser necesario establecer dos o más controladores de procesamiento de entrada o computación, a causa de la complejidad del sistema a construir. Si el sentido común así te lo dicta, hazlo.

Aunque la Figura 14.9 implica una estructura con tres ramas, en grandes sistemas la complejidad del flujo puede hacer que existan dos o más módulos de control, uno para cada una de las funciones genéricas descritas anteriormente. El número de módulos del primer nivel debería limitarse al mínimo que pueda realizar las funciones de control y mantener al mismo tiempo unas buenas características de acoplamiento y cohesión.

Paso 6. Realizar «descomposición de segundo nivel». La descomposición de segundo nivel se realiza mediante la conversión de las transformaciones individuales (burbujas) de un DFD en los módulos correspondientes dentro de la arquitectura. Empezando desde el límite del centro de transformación y moviéndonos

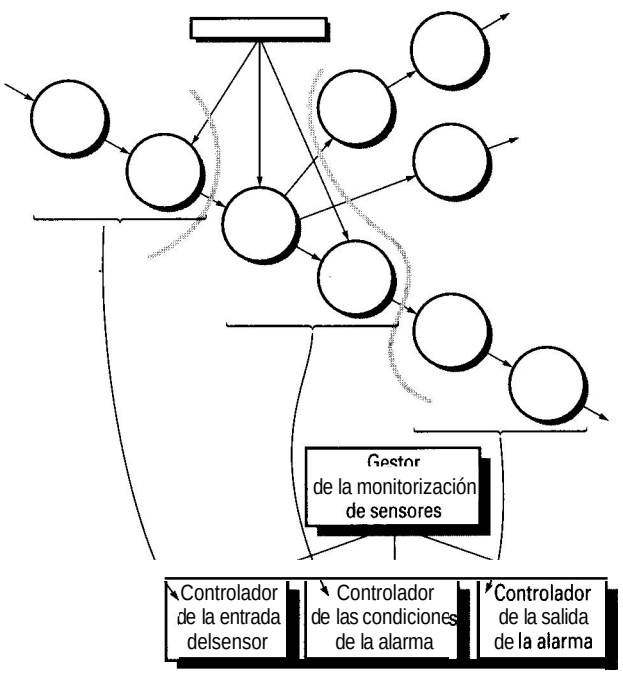


FIGURA 14.9. Descomposición de primer nivel para la monitorización de sensores.

hacia fuera a lo largo de los caminos de entrada, y luego de salida, las transformaciones se convierten en niveles subordinados de la estructura del software. El enfoque general del segundo nivel de descomposición del flujo de datos *HogarSeguro* se ilustra en la Figura **14.10.**

Aunque la Figura 14.10 ilustra una correspondencia una a uno entre las transformaciones del DFD y los módulos del software, ocurren frecuentemente diferentes conversiones. Se pueden combinar dos o incluso tres burbujas y representarlas como un solo módulo (teniendo presente los problemas potenciales de la cohesión), o una sola burbuja puede dividirse en dos o más módulos. Las consideraciones prácticas y las medidas de la calidad dictan el resultado de la descomposición en factores de segundo nivel. La revisión y el refinamiento pueden llevar a cambios en la estructura, pero puede servir como una primera iteración del diseño.

*Mantén bajos los **controladores** de **trabajo** en la estructura del programa. **Así,** la arquitectura será más fácil de **modificar.***

La descomposición de factores de segundo nivel del flujo de entrada sigue de igual manera. La descomposición en factores se realiza de nuevo moviéndose hacia fuera desde el límite del centro de transformación correspondiente al flujo de entrada. El centro de transformación del software del subsistema de monitorizar sensores se direcciona de manera algo diferente. Cada

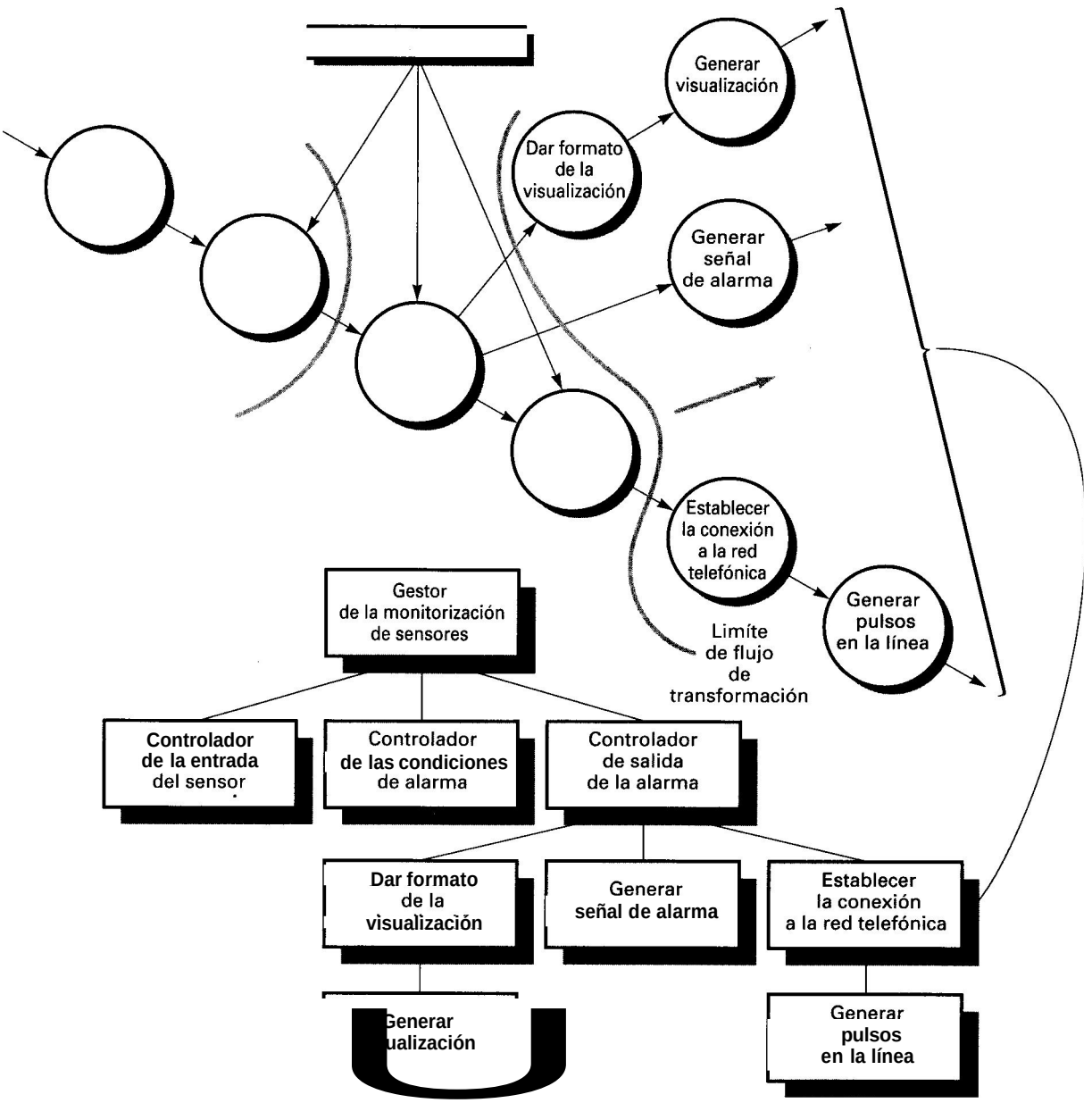


FIGURA 14.10. Descomposición de factores de segundo nivel de monitorización de sensores.

conversión de datos o transformaciones de cálculo de la porción de transformación del DFD se convierte en un módulo subordinado al controlador de transformación. En la Figura 14.11 se muestra una estructura de programa completa de primera iteración.

Los módulos direccionados de la manera descrita anteriormente y mostrados en la Figura 14.11 representan un diseño inicial de la estructura del programa. Aunque los módulos se nombran de manera que indiquen su función, se debería escribir para cada uno un breve texto que explique su procesamiento (adaptado del EP creado durante la modelación de análisis). El texto describe:

- la información que entra y la que sale fuera del módulo (una descripción de la interfaz);
- la información que es retenida en el módulo, por ejemplo, datos almacenados en una estructura de datos local;
- una descripción procedimental que indique los principales puntos de decisión y las tareas;
- un pequeño estudio de las restricciones y características especiales (por ejemplo, archivo I/O, características dependientes del hardware, requisitos especiales de tiempo).

El texto explicativo sirve como una primera generación de la especificación de diseño. Sin embargo, se dan más refinamientos y adiciones regularmente durante este periodo de diseño.

Paso 7. Refinar la estructura inicial de la arquitectura usando heurísticas para mejorar la calidad del software. Una primera estructura de arquitectura siempre puede refinarse aplicando los conceptos de independencia de módulos (Capítulo 13). **Los** módulos se incrementan o reducen para producir una descomposición razonable, buena cohesión, acoplamiento mínimo **y** lo más importante, una estructura que se pueda implementar sin dificultad, probarse sin confusión y mantenerse **sin** problemas.

Los refinamientos se rigen por el análisis y los métodos de evaluación descritos brevemente en la Sección 14.4, así como por consideraciones prácticas **y** por el sentido común. Hay veces, por ejemplo, que el controlador del flujo de datos de entrada es totalmente innecesario, o se requiere cierto procesamiento de entrada en un módulo subordinado al controlador de transformación, o no se puede evitar un alto acoplamiento debido a datos generales, o no se pueden lograr las características estructurales óptimas (vea la Sección 13.6). **Los** requisitos del software junto con el buen juicio son los árbitros finales.

CONSEJO

Elimina los módulos de control redundantes. Es decir, si un módulo de control no hace otro caso que controlar otro módulo, esto función de control debería explotarse a mayor nivel.

CONSEJO

Céntrese en lo independencia funcional de los módulos que derive. Su objetivo debe ser uno cohesión alta y un acoplamiento débil.

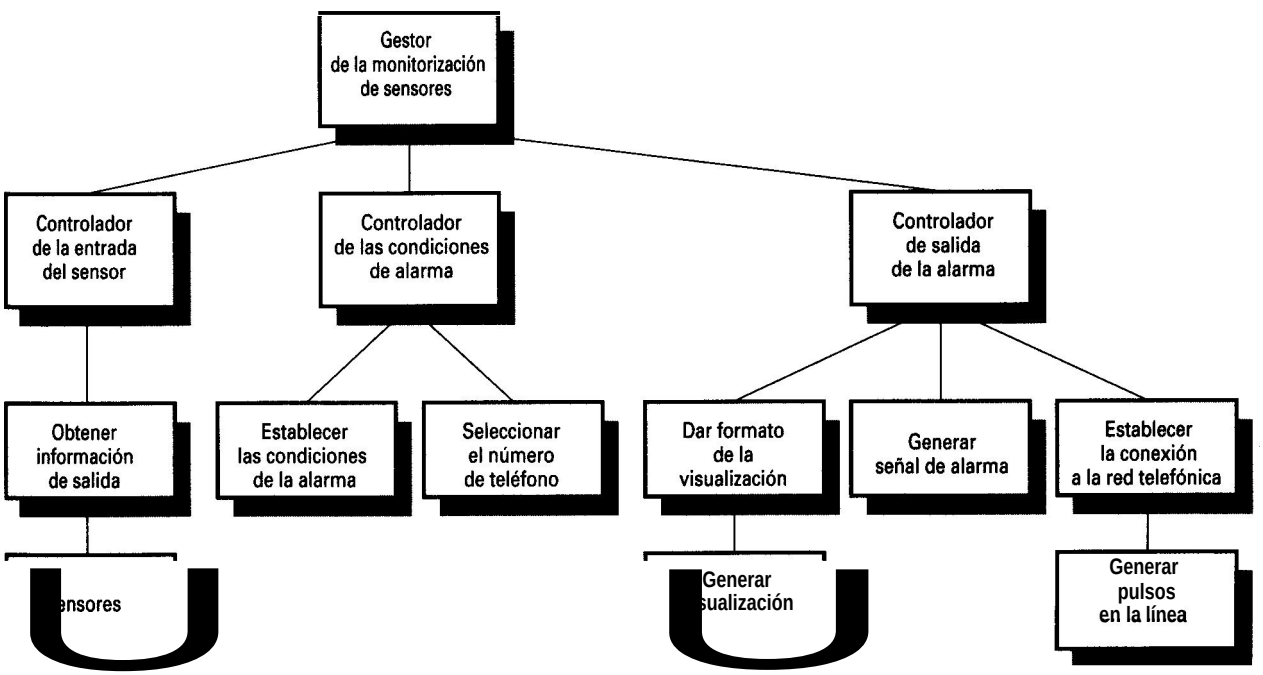


FIGURA 14.11. Primera iteración de la estructura del programa para *monitorizar* sensores.

Se pueden hacer muchas modificaciones a la primera estructura desarrollada para el subsistema de *monitorizar sensores* de *HogarSeguro.* Algunas de ellas son:

1. El controlador del flujo de entrada se puede eliminar porque es innecesario cuando sólo hay que manejar un solo camino de flujo de entrada.
2. La subestructura generada del flujo de transformación puede reducirse en el módulo **establecer las condiciones de alarma** (que incluirá ahora el procesamiento implicado por **seleccionar número de teléfono).** El controlador de transformación no será necesario y la pequeña disminución en la cohesión es tolerable.
3. Los módulos **dar formato de visualización** y **generar la visualización** pueden unirse (asumimos que dar formato de visualización es bastante simple) en un nuevo módulo denominado **producir visualización.**

En la Figura 14.12 se muestra la estructura del software refinada del subsistema de monitorizar sensores.

El objetivo de los siete puntos anteriores es desarrollar una representación general del software. Es decir, una vez que se ha definido la estructura, podemos evaluar y refinar la arquitectura del software viéndola en su conjunto. Las modificaciones hechas ahora requieren poco trabajo adicional, pero pueden tener un gran impacto en la calidad del software.

El lector debería reflexionar un momento y considerar la diferencia entre el enfoque de diseño descrito anteriormente y el proceso de «escribir programas)). Si el código es la única representación del software, el desarrollador tendrá grandes dificultades en evaluar o refinar a nivel general u holístico y, de hecho, tendrá dificultad para que «los árboles dejen ver el bosque».

14.7 ANÁLISIS DE LAS TRANSACCIONES

En muchas aplicaciones software, un solo elemento de datos inicia uno o varios de los flujos de información que llevan a cabo una función derivada por el elemento de datos iniciador. El elemento de datos, denominado transacción, y sus características de flujo correspondientes se trataron en la Sección 14.5.2. En esta sección consideramos los pasos de diseño usados para tratar el flujo de transacción.

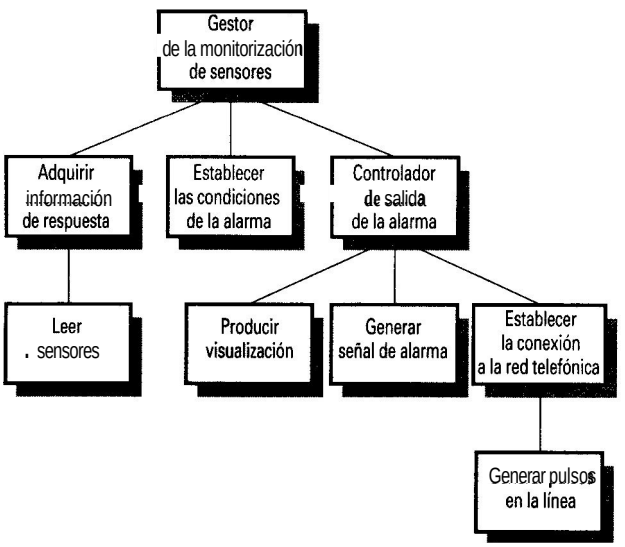


FIGURA 14.12. Estructura refinada del programa para *monitorizar sensores.*

14.7.1. Un ejemplo

El análisis de las transacciones se ilustrará considerando el subsistema de *interacción con el usuario* del software *HogarSeguro.* El nivel 1 del flujo de datos de este subsistema se muestra como parte de la Figura 14.6. Refinando el flujo, se desarrolla un nivel 2 del diagrama de flujo (también se crearían un diccionario de datos correspondiente, EC y EP) que se muestra en la Figura 14.13.

Como se muestra en la figura, **Órdenes y datos del usuario** fluye al sistema y provoca un flujo de información adicional a lo largo de uno de tres caminos de acción. Un elemento de datos, **tipo de orden,** hace que el flujo de datos se expanda del centro. Por tanto, la característica general del flujo de datos es de tipo transacción.

Debería recalcarse que el flujo de información a lo largo de dos de los tres caminos de acción acomodan flujos de entrada adicionales (por ejemplo, parámetros y datos del sistema son entradas en el camino de acción a «configurar»). Todos los caminos de acción fluyen a una sola transformación, *mostrar mensajes* y *estado.*

14.7.2. Pasos del diseño

Los pasos del diseño para el análisis de las transacciones son similares y en algunos casos idénticos a los pasos para el análisis de las transformaciones (Sección 14.6). La principal diferencia estriba en la conversión del DFD en la estructura del software.

Paso 1. Revisar el modelo fundamental del sistema.

Paso 2. Revisar y refinar los diagramas de flujo de datos para el software.

Paso 3. Determinar si el DFD tiene características de flujo de transformación o de transacción. Los pasos 1, 2 y 3 son idénticos a los correspondientes pasos del análisis de las transformaciones. El DFD mostrado en la Figura 14.13 tiene la clásica característica de flujo de tran-

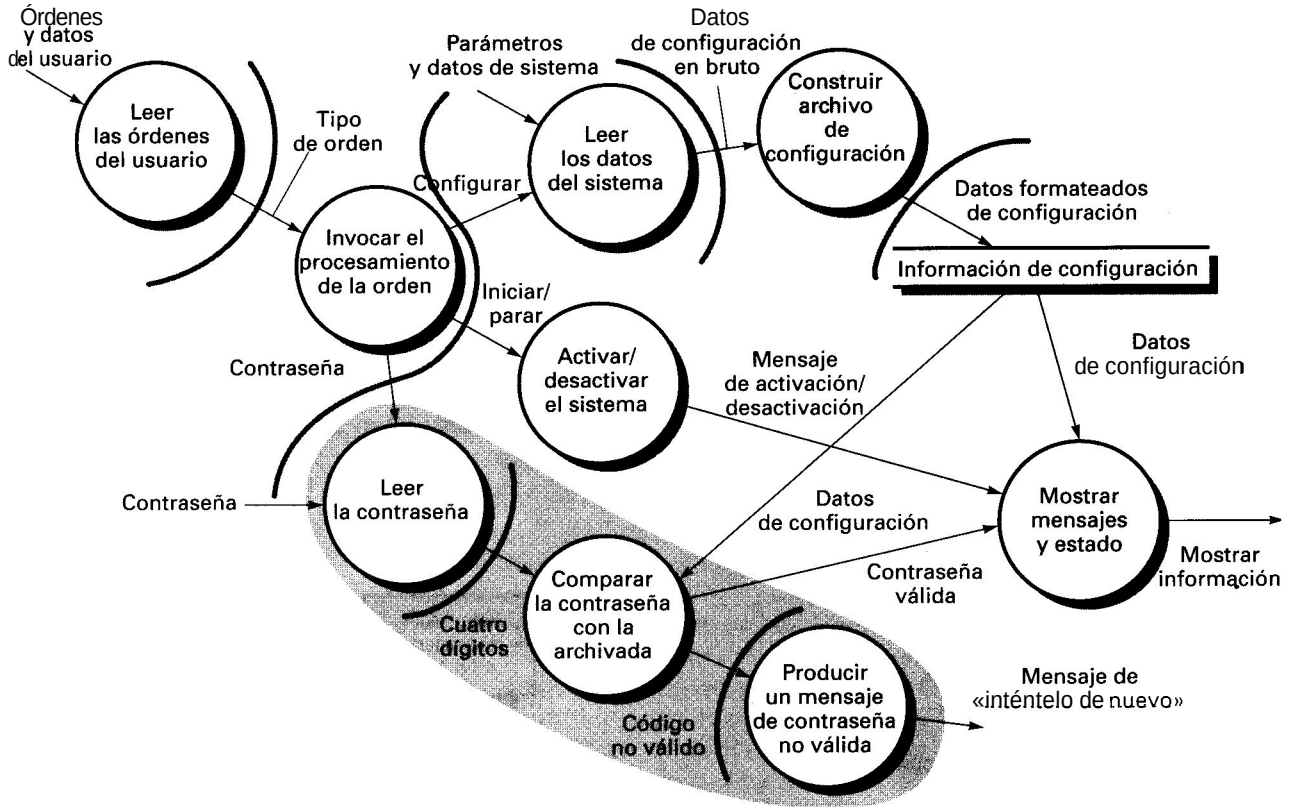


FIGURA 14.13. Nivel 2 de DFD para el subsistema de interación del usuario con límites del flujo.

sacción. Sin embargo, el flujo a lo largo de dos de los caminos de acción que emanan desde la burbuja *invocar el procesamiento de la orden* parece tener características de flujo de transformación. Por tanto, se deben establecer los límites de flujo para ambos tipos de flujos.

Paso 4. Identificar el centro de transacción y las características de flujo a lo largo de cada camino de acción. La posición transacción se puede discernir inmediatamente del DFD. El centro de transacción está en el origen de varios caminos de acción que fluyen radialmente desde él. Para el flujo mostrado en la Figura 14.13, la burbuja *invocar el procesamiento de la orden* es el centro de transacción.

El camino de entrada (por ejemplo el camino de flujo a lo largo del que se recibe una transacción) y todos los caminos de acción deben aislarse. Los límites que definen un camino de recepción y los caminos de acción también se muestran en la figura. Se debe evaluar las características individuales de flujo de cada camino de acción. Por ejemplo, el camino «de la contraseña» (mostrado incluido en un área sombreada en la Fig. 14.13) tiene características de transformación. Los flujos de entrada, de transformación y de salida se indican con límites.

Paso 5. Transformar el DFD en una estructura de programa adecuada al procesamiento de la transacción. El flujo de transacción se convierte en una arquitectura que contiene una rama de entrada y una rama de distribución. La estructura de la rama de entrada se desarrolla de la misma manera que para un análisis de transformación. Empezando por el centro de transacción, las burbujas que hay a lo largo del camino de entrada se convierten en módulos. La estructura de la rama de distribución contiene un módulo distribuidor que controla todos los módulos de acción subordinados. Cada camino de flujo de acción del DFD se convierte en una estructura que corresponde a sus características específicas de flujo. Este proceso se ilustra esquemáticamente en la Figura 14.14.

La descomposición de primer nivel tiene como resultado la derivación de la jerarquía de control poro el software. La descomposición de segundo nivel distribuye los módulos de trabajo a los controladores opropiodos.

Considerando el flujo de datos del subsistema de *interacción del usuario,* la descomposición de primer nivel del paso 5 se muestra en la Figura 14.15. Las burbujas *leer órdenes del usuario y activarldesactivar el sistema* se convierten directamente en la arquitectura, sin la necesidad de módulos intermedios de control. El centro de transacción, *invocar el procesamiento de la orden,* se convierte directamente en un módulo distribuidor con el mismo nombre. **Los** controladores de la configuración del sistema y procesamiento de la contraseña se obtienen como se indica en la Figura 14.16.

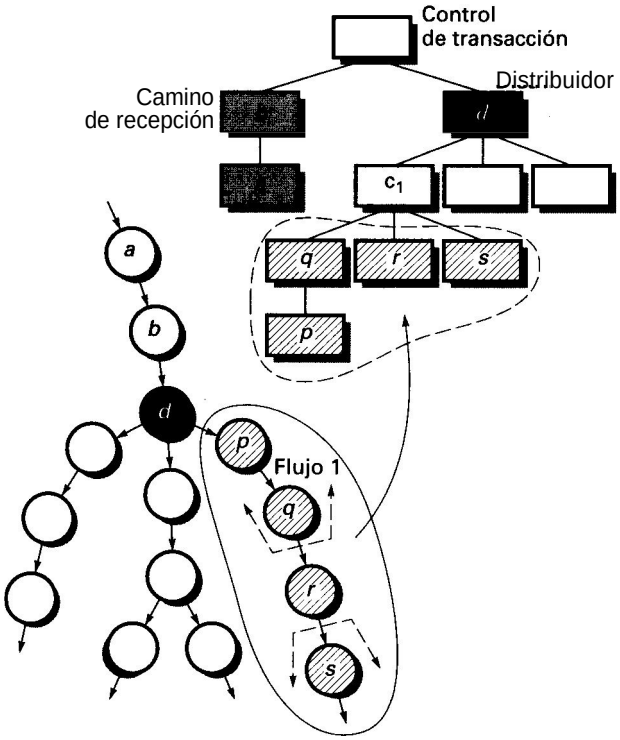


FIGURA 14.14. Análisis de transacción.

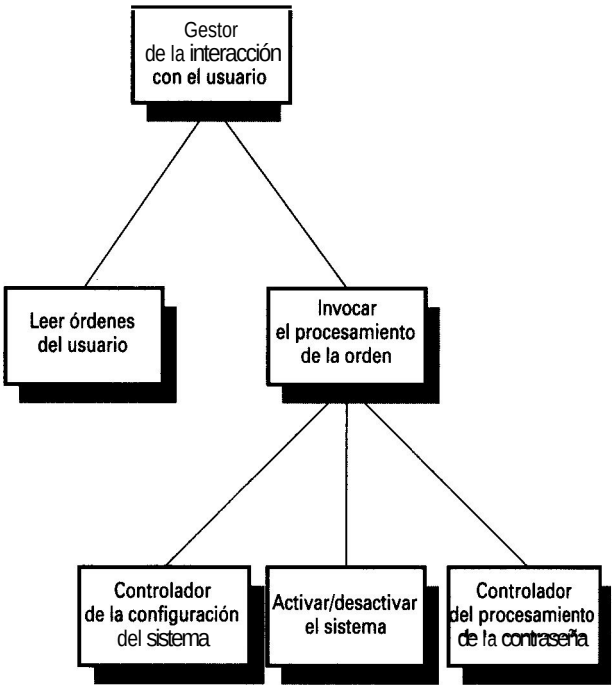


FIGURA 14.15. Descomposición en factores de primer nivel del subsistema interacción del usuario.

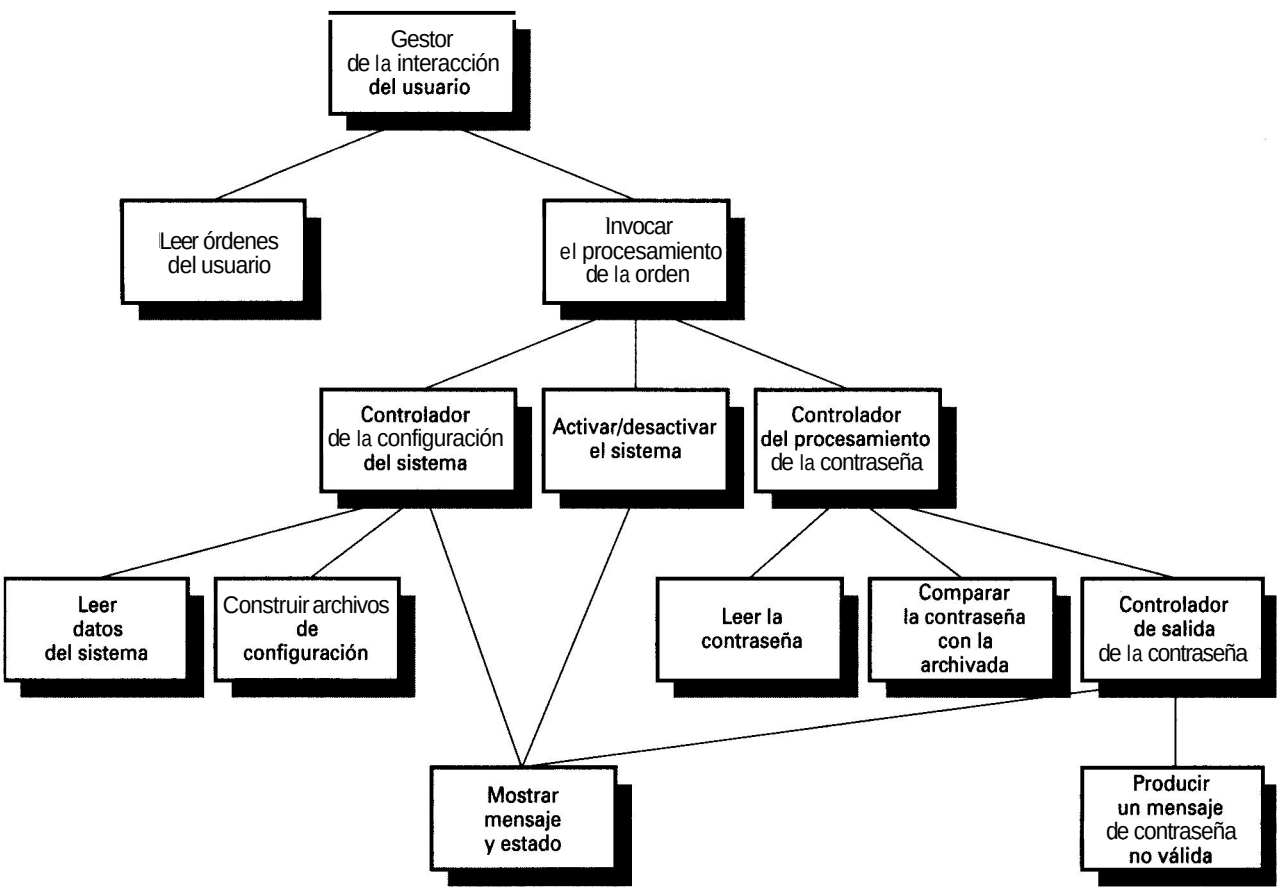


FIGURA 14.16. Primera iteración de la estructura del programa del subsictema ***interacción del*** usuario.

Paso 6. Descomponer y refinar la estructura de transacción y la estructura de todos los caminos de acción. Cada camino de acción del diagrama de flujo de datos tiene sus propias características de flujo de información. Ya hemos dicho anteriormente que se pueden encontrar flujos de transformación o de transacción. La «subestructura» relacionada con el camino de acción se desarrolla usando los pasos estudiados en esta sección y en la anterior.

Por ejemplo, considere el flujo de información del procesamiento de contraseña mostrado (dentro del área sombreada) en la Figura **14.13.** El flujo presenta las características clásicas de transformación. Se introduce una contraseña (flujo de entrada) y se transmite a un centro de transformación donde se compara con las contraseñas almacenadas. Se produce una alarma y un mensaje de advertencia (flujo de salida) si no coincide con ninguna.

El camino «configurar» se dibuja similarmente usando el análisis de transformación. La arquitectura de software resultante se muestra en la Figura 14.16.

Paso 7. Refinar la primera arquitectura del programa usando heurísticas de diseño para mejorar la calidad del software. Este paso del análisis de transacción es idéntico al correspondiente paso del análisis de transformación.

En ambos métodos de diseño se deben considerar cuidadosamente criterios tales como independencia del módulo, conveniencia (eficacia de implementación y prueba) y facilidad de mantenimiento a medida que se proponen modificaciones estructurales.

14.6 REFINAMIENTO DEL DISEÑO ARQUITECTÓNICO

El éxito de la aplicación del análisis de transformación o de transacción se complementa añadiendo la documentación adicional requerida como parte del diseño arquitectónico. Después de haber desarrollado y refinado la estructura, se deben completar las siguientes tareas:

- se debe desarrollar una descripción del procesamiento para cada módulo.
- se aporta una descripción de la interfaz para cada módulo.
- se definen las estructuras de datos generales y locales.
- se anotan todas las limitaciones/restricciones del diseño.
- se lleva a cabo una revisión del diseño.
- se considera un «refinamiento» (si es necesario y está justificado).

¿Qué pasa una vez que la arquitectura ha sido creada?

Un texto explicativo del procesamiento es (idealmente) una delimitada descripción sin ambigüedades del procesamiento que ocurre dentro de un módulo. La narrativa describe el procesamiento, las tareas, las decisiones y la entrada/salida. La descripción de la interfaz requiere el diseño de interfaces internas de módulo, interfaces externas del sistema y la interfaz hombre-computadora (Capítulo 15). El diseño de estructuras de datos puede tener un profundo impacto en la arquitectura y en los detalles procedimentales de cada componente del software. También se documentan las restricciones/limitaciones de cada módulo. Algunos aspectos típicos que pueden tratarse incluyen: la restricción de tipo o formato de datos, las limitaciones de memoria o de tiempo, la delimitación de los valores o cantidades de las estructuras de datos, los casos especiales no considerados y las características específicas de un módulo individual. El propósito de una sección restricciones/limitaciones es reducir el número de errores debidos a características funcionales asumidas.

Una vez que se ha desarrollado la documentación de diseño para todos los módulos, se lleva a cabo una revisión (vea las directrices de revisión en el Capítulo **8).** LA revisión hace hincapié en el seguimiento de los requisitos del software, la calidad de la arquitectura del programa, las descripciones de las interfaces, las descripciones de las estructuras de datos, los datos prácticos de la implementación, la capacidad de prueba y la facilidad de mantenimiento.

Documento del diseño de software.

Debe fomentarse el refinamiento de la arquitectura del software durante las primeras etapas del diseño. Como ya dijimos anteriormente en este capítulo, los estilos arquitectónicos alternativos deben ser derivados, refinados y evaluados para un «mejor» enfoque. Este enfoque de optimización es uno de los verdaderos beneficios derivados del desarrollo de la representación de la arquitectura del software.

Es importante anotar que la simplicidad estructural es, a menudo, reflejo de elegancia y eficiencia. El refinamiento del diseño debería luchar por obtener un pequeño número de módulos consecuentes a la modularidad operativa y la estructura de datos menos compleja que sirva adecuadamente a los requisitos de información.

RESUMEN

La arquitectura del software nos proporciona una visión global del sistema a construir. Describe la estructura y la organización de los componentes del software, sus propiedades y las conexiones entre ellos. Los componentes del software incluyen módulos de programas y varias representaciones de datos que son manipulados por el programa. Además, el diseño de datos es una parte integral para la derivación de la arquitectura del software. La arquitectura marca decisiones de diseño tempranas y proporciona el mecanismo para evaluar los beneficios de las estructuras de sistema alternativas.

El diseño de datos traduce los objetos de datos definidos en el modelo de análisis, en estructuras de datos que residen dentro del software. Los atributos que describe el objeto, las relaciones entre los objetos de datos y su uso dentro del programa influyen en la elección de la estructura de datos. A mayor nivel de abstracción, el diseño de datos conducirá a lo que se define como una arquitectura para una base de datos o un almacén de datos.

El ingeniero del software cuenta con diferentes estilos y patrones arquitectónicos. Cada estilo describe una categoría de sistema que abarca un conjunto de componentes que realizan una función requerida por el sistema, un conjunto de conectores que posibilitan la comunicación, la coordinación y cooperación entre los componentes, las restricciones que definen cómo se integran los componentes para conformar el sistema, y los modelos semánticos que facilitan al diseñador el entendimiento de todas las partes del sistema.

Han sido propuestos uno o varios estilos arquitectónicos por sistema, y el método de análisis de compromisos para la arquitectura podría utilizarse para evaluar la eficacia de cada arquitectura propuesta. Esto se consigue determinando la sensibilidad de los atributos de calidad seleccionados (también llamados dimensiones del diseño) con diferentes mecanismos de realización que reflejan las propiedades de la arquitectura.

El método de diseño arquitectónico presentado en este capítulo utiliza las características del flujo de datos descritas en el modelo de análisis que derivan de un estilo arquitectónico utilizado comúnmente. El diagrama de flujo de datos se descompone dentro de la estructura del sistema a través de dos enfoques de análisis—el análisis de las transformaciones y/o el análisis de las transacciones—. Se aplica el análisis de las transformaciones a un flujo de información que presenta diferentes límites entre los datos de entrada y de salida. El DFD se organiza en una estructura que asigna los controles de entrada, procesamiento y salida a través de tres jerarquías separadas de módulos de descomposición en factores. El análisis de las transformaciones se aplica cuando un Único ítem de información bifurca su flujo a través de diferentes caminos. El DFD se organiza en una estructura que asigna el control a una subestructura que adquiere y evalúa la transacción. Otra subestructura controla todas las acciones potenciales de procesamiento basadas en la transacción. Una vez que la arquitectura ha sido perfilada se elabora y se analiza contrastándola con los criterios de calidad.

El diseño arquitectónico agrupa un grupo inicial de actividades de diseño que conducen a un modelo completo del diseño del software. En los siguientes capítulos, se estudiará el diseño de las interfaces y de los componentes.

REFERENCIAS

[AHO83] Aho, A.V., J. Hopcroft y J.Ullmann, *Data Structures and Algorithms,* Addison-Wesley, 1983.

[BAS98] Bass, L., P. Clements y R. Kazman, *Software Architecture in Practice,* Addison-Wesley, 1998.

[DAH72] Dah, O., E. Dijkstra y C. Hoare, *Structured Programming,* Academic Press, 1972.

[DAT95] Date, C.J., *An Introduction to Database Systems,* Sexta Edición, Addison-Wesley, 1995.

[DEN73] Dennis, J.B., «Modularity», en *Advanced Course on Software Engineering,* F.L. Bauer (ed.), Springer-Verlag, Nueva York, 1973, pp. 128-182.

[FRE80] Freeman, P., «The Context of Design», in *Software Design Techniques* (L.P. Freeman y A. Wasserman, eds.), IEEE Computer Society Press, 3."ed., 1980, pp. 2-4.

[INM95] Inmon, W.H., «What is a Data Warehouse?» Prism Solutions, Inc. 1995, presentada en: **http://www.cait.wustl.edu/cait/papers/prism/vol1_no1**.

[KAZ98] Kazman, R. *The Architectural Tradeoff Analysis Method,* Software Engineering Institute, CMU/SEI-98-TR-008. Julio 1998.

[KIM98] Kimball, R., L. Reeves, M. Ross y W. Thornthwaite, *The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing, and Deploying,* Data Warehouses, Willey, 1998.

[LIN79] Linger, R.C., H.D. Mills y B.I. Witt, *Structured Programming,* Addison-Wesley, 1979.

[MAT96] Mattison, R., *Data Warehousing: Strategies, Technologies and Techniques,* McGraw-Hill, 1996.

[MOR80] Morris, J., «Programming by Successive Refinement of Data Abstractions», *Software-Practice and Experience,* vol. 10, núm. 4, abril 1980, pp. 249-263.

[MYE78] Myers, G., *Composite Structures Design,* Van-Nostrand, 1978.

[PET81] Peters, L.J., ***Software Design: Methods and Techniques,*** Yourdon Press, Nueva York, 1981.

[PRE98] Preiss, B.R. ***Data Structures and Algorithms: With Ohject-Oriented Design Patterns in C++***, Wiley, 1998.

[SHA96] Shaw, M., y D. Garlan, ***Software Arquitecture,*** Prentice Hall, 1996.

[SHA97] Shaw, M., y P. Clements, «A Field Guide to Boxology: Preliminary Classification of Architectural Styles for Software Systems», ***Proc. COMPSAC,*** Washington DC, Agosto 1997.

[STE74] Stevens, W., G. Myers y L. Constantine, «Structured Design», ***IBM System Journal,*** vol.13, n.° 2, 1974, pp. 115-139.

[WAS80] Wasserman, A., «Principles of Systematic Data Design and Implementation», en ***Software Design Techniques*** (P. Freeman y A. Wasserman, eds.), 3.ª ed., IEEE Computer Society Press, 1980, pp. 287-293.

[WIR71] Wirth, N., «Program Development by Stepwise Refinement», CACM, vol. 14, n.° 4, 1971, pp. 221-227.

[YOU79] Yourdon, E., y L. Constantine, ***Structures Design,*** Prentice-Hall, 1979.

[ZHA98] Zhao, J., «On Assessing the Complexity of Software Architectures», ***Proc. Intl. Software Architecture Workshop***, ACM, Orlando, Florida, 1998, pp. 163-167.

PROBLEMAS Y PUNTOS A CONSIDERAR

14.1. Usando la arquitectura de una casa o un edificio a modo de metáfora, dibuje comparaciones con la arquitectura del software. ¿En qué se parecen la disciplina de la arquitectura clásica y la de la arquitectura del software? ¿En qué se diferencian?

14.2. Escriba un documento de tres a cinco páginas que contenga directrices para la selección de estructuras de datos basándose en la naturaleza del problema. Empiece delimitando las clásicas estructuras de datos que se encuentran en el software y después describiendo los criterios para la selección de éstos para tipos particulares de problemas.

14.3. Explique la diferencia entre una base de datos que sirve a una o más aplicaciones de negocios convencionales y un almacén de datos.

14.4. Escriba un documento de tres a cinco página que describa cómo se utilizan las técnicas de minería de datos en un entorno de negocio y el estado actual de las técnicas DCBC.

14.5. Presente dos o tres ejemplos de aplicaciones para cada estilo arquitectónico citado en la Sección 14.3.1.

14.6. Algunos de los estilos arquitectónicos citados en la Sección 14.3.1. son jerárquicos por naturaleza y otros no. Haga una lista de cada tipo. ¿Cómo se implementan los estilos arquitectónicos no jerárquicos?

14.7. Seleccione una aplicación que le sea familiar. Conteste cada una de las preguntas propuestas para el control y los datos de la Sección 14.3.2.

14.8. Estudie el MACA (utilizando el libro de [KAZ98]) y presente un estudio detallado de los seis pasos presentados en la Sección 14.4.1.

14.9. Seleccione una aplicación que le sea familiar. Utilizando, donde sea requerido, su mejor intuición, identifique el conjunto de dimensiones del diseño y después realice el análisis del espectro y el análisis de la selección del diseño.

14.10. Estudie el EDC (utilizando el libro de [SHA96]) y desarrolle un espacio de diseño cuantificado para una aplicación que le sea familiar.

14.11. Algunos diseñadores defienden que todo el flujo de datos debe ser tratado como orientado a transformaciones. Estudie como afectará esta opinión a la estructura del software que se obtiene cuando un flujo orientado a transacción es tratado como de transformación. Utilice un flujo de ejemplo para ilustrar los puntos importantes.

14.12. Si no lo ha hecho, complete el Problema 12.12. Utilice los métodos de diseño descritos en este capítulo para desarrollar una estructura de programa para el SSRB.

14.13. Mediante un diagrama de flujo de datos y una descripción del procesamiento, describa un sistema basado en computadora que tenga unas características de flujo de transformación singulares. Defina los límites del flujo y transforme el DFD en la estructura software usando una técnica de las descritas en la Sección 14.6.

14.14. Mediante un diagrama de flujo de datos y una descripción de procesamiento, describa un sistema basado en computadora que tenga unas características de flujo de transacción claras. Defina los límites del flujo y direcciones el DFD en una estructura software utilizando la técnica descrita en la Sección 14.7.

14.15. Usando los requisitos obtenidos en un estudio hecho en clase, complete los DFD y el diseño arquitectónico para el ejemplo *HogarSeguro* presentado en las Secciones 14.6 y 14.7. Valore la independencia funcional de todos los módulos. Documente su diseño.

14.16. Estudie las ventajas y dificultades relativas de aplicar un diseño orientado al flujo de datos en las siguientes áreas: (a) aplicaciones de microprocesador empotrado, (b) análisis de ingeniería/científico, (c) gráficos por computadora, (d) diseño de sistemas operativos, (e) aplicaciones de negocio, (f) diseño de sistemas de gestión de bases de datos, (g) diseño de software de comunicaciones, (h) diseño de compiladores, (i) aplicaciones de control de proceso y (j) aplicaciones de inteligencia artificial.

14.17. Dado un conjunto de requisitos que le proporcione su profesor (o un conjunto de requisitos de un problema en el que esté trabajando actualmente) desarrolle un diseño arquitectónico completo. Lleve a cabo una revisión del diseño (Capítulo 8) para valorar la calidad de su diseño. Este problema debe asignarse a un equipo en vez de a un solo individuo.

LECTURAS Y FUENTES DE INFORMACIÓN

Durante la Última década han aparecido muchísimos libros sobre arquitectura de software. Los libros de Shaw y Gannon [SHA96], Bass, Clements y Kazman [BAS98] y Buschmann y colaboradores[BUS98], proporcionan un tratamiento en profundidad sobre la materia. El primer trabajo de Garlan ***(An Introduction to Software Architecture,*** Software Engineering Institute, CMU/SEI-94-TR-02 1,1994) proporciona una excelente introducción al tema.

Los libros específicos de implementación de arquitectura, sitúan el diseño arquitectónico dentro de un entorno específico de desarrollo ***o*** tecnología. Mowray ***(CORBA Design Patterns,*** Wiley, 1997) y Mark y colaboradores ***(Object Management Architecture Guide,*** Wiley, 1996) proporcionan parámetros de diseño detallados para el marco de soporte de las aplicaciones distribuidas de CORBA. Shanley ***(Protected Mode Software Architecture,*** Addison-Wesley, 1996) proporciona una guía de diseño arquitectónico para aquellos que diseñen sistemas operativos a tiempo real basados en computadora, sistemas operativos multitarea ***o*** controladores de dispositivos.

Las investigaciones actuales sobre arquitectura de software están recogidas en el anuario ***Proceedings of the International Workshop on Software Architecture,*** patrocinado por la ACM y otras organizaciones de computadoras y en el ***Proceedings of the International Conference on Software Engineering.***

El modelado de datos es un prerrequisito para un buen diseño de datos. **Los** libros de Teory ***(Database Modeling & Design,*** Academic Press, 1998), Schimdt ***(Data Modeling for Information Professionals,*** Prentice Hall, 1998), Bobak, ***(Data Modeling and Design for Today´s Architectures,*** Artech House, 1997), Silverston, Graziano e Inmon ***(The Data Model Resource Book,*** Wiley, 1997), Date [DAT95], Reingruber y Gregory ***(The Data Modeling Handbook: A Best-Practice Approach to Building Quality Data Models,*** Wiley, 1994), Hay ***(Data Model Patterns: Conventions of Thought,*** Dorset House, 1994) contienen una presentación detallada de la notación de modelado de datos, heurísticas, y enfoques de diseño de bases de datos. En los últimos años el diseño de almacenes de datos ha ido cobrando importancia. Los libros de Humphreys, Hawkins y Dy ***(Data Warehousing: Architecture and Implementation,*** Prentice Hall, 1999), Kimball [KIM98] e Inmon [INM95] cubren con gran detalle la materia.

Docenas de libros actuales, a menudo abordan el diseño de datos y el diseño de estructuras desde un contexto específico del lenguaje de programación. Algunos ejemplos típicos los encontramos en:

Horowitz, **E.** Y S. Sahni, ***Fundamentals of Data Structures in Pascal,*** 4.ª ed., W.H. Freeman & Co., 1999.

Kingston, J.H., ***Algorithms and Data Structures: Design, Correctness, Analysis,*** 2.ª ed., Addison-Wesley, 1997.

Main, **M.,** ***Data Structures & Other Objects Using Java,*** Addison-Wesley, 1998.

Preiss, B.R., ***Data Structures and Algorithms: With Object-Oriented Design Patterns in C++,*** Wiley, 1998.

Sedgewick, **R.,** ***Algorithms in C++: Fundamentals, Data Structures, Sorting, Searching,*** Addison-Wesley, 1999.

Standish, T.A., ***Data Structures in Java,*** Addison-Wesley, 1997.

Standish, T.A., ***Data Structures, Algorithms, and Software Principles in C,*** Addison-Wesley, 1995.

En la mayoría de **los** libros dedicados a la ingeniería de software se puede encontrar un tratamiento general del diseño del software en discusión con el diseño arquitectónico y el diseño de datos. **Los** libros de Pfleeger ***(Software Engineering: Theory and Practice,*** Prentice may, 1998) y Sommerville (*Software* ***Engineering,*** 5." ed., Addison-Wesley, 1995) son representativos de los libros que cubren en detalle el tema del diseño.

Se puede encontrar un tratamiento más riguroso de la materia en Feijs ***(Formalization of Design Methods,*** Prentice may, 1993), Witt y colaboradores ***(Software Architecture and Design Principles,*** Thomson Publishing, 1994) y Budgen ***(Software Design,*** Addison-Wesley, 1994).

Se pueden encontrar representaciones completas de diseños orientados a flujos de datos en Myers [MYE78], Yourdon y Constantine[YOU79], Buhr ***(System Design with Ada,*** Prentice may, 1984), y Page-Jones ***(The Practica1 Guide to Structured Systems Design,*** segunda edición, Prentice may, 1988). Estos libros están dedicados solamente al diseño y proporcionan unas completas tutorías en el enfoque de flujo de datos.

En Intemet están disponibles una gran variedad de fuentes de información sobre diseño de software y temas relacionados. Una lista actualizada de referencias web sobre conceptos y métodos de diseño relevantes se puede encontrar en: **http://www.pressman5.com**.