Tema 5.- Análisis estructurado

5.1.- Introducción.

Todos los métodos de análisis de requisitos se basan en la construcción de modelos del sistema que se pretende desarrollar. Utilizando alguna notación, propia de cada método, creamos modelos que reflejen el sistema, y aplicamos técnicas de descomposición y razonamiento *top-down*, de tal forma que al final establecemos la esencia del sistema que pretendemos construir.

El desarrollo de modelos presenta algunas ventajas claras[1]:

Permite centrarse en determinadas características del sistema, dejando de lado otras menos significativas. Esto nos permite centrar las discusiones con el usuario en los aspectos más importantes del sistema, sin distraernos en características del sistema que sean irrelevantes.

Permite realizar cambios y correcciones en los requisitos a bajo coste y sin correr ningún riesgo. Si nos damos cuenta que no habíamos entendido las necesidades del usuario o si el usuario ha cambiado de idea acerca de los requisitos del sistema, podemos cambiar el modelo o incluso desecharlo y empezar de nuevo. Si no hiciésemos modelos, los cambios en los requisitos sólo se efectuarían después de construir el producto software, y el coste sería muchísimo mayor.

Permite verificar que el ingeniero del software ha entendido correctamente las necesidades del usuario y que las ha documentado de tal forma que los diseñadores y programadores pueden construir el software.

Sin embargo, no todas las técnicas de análisis logran estos tres objetivos: una descripción del sistema de 500 páginas (que sería, en sentido estricto, un modelo) oculta las características del sistema, tanto las relevantes como las irrelevantes, su desarrollo puede costar más que el del propio sistema, es difícil de modificar y no permite verificar si se han establecido los requisitos.

El análisis de requisitos clásico, usado hasta finales de los 70, consistía en redactar especificaciones funcionales, en forma de documentos textuales, de este tipo:

Eran monolíticas. Para entender el sistema había que leerse la especificación de principio a fin. No había posibilidad de que ni el analista ni el usuario pudiesen centrarse en una determinada parte de la especificación, sin tener que leerse el resto.

Eran redundantes. La misma información se repartía en diferentes partes del documento. Debido a esto cualquier cambio que se hiciese en la especificación debía reflejarse en varios puntos del documento. Esta situación produce con frecuencia inconsistencia: si no se cambiaba en todos estos lugares la misma información (p.ej. el mismo código) tenía definiciones distintas.

[1] Pressman5, pag. 200

Eran ambiguas. Al estar escritas en lenguaje natural, podían ser interpretadas de forma distinta por analistas, usuarios, diseñadores o programadores. Hay estudios que muestran que el 50% de los errores encontrados en el sistema final y el 70% del coste de la corrección de los errores surgía de este tipo de malentendidos.

Eran imposibles de mantener o modificar. Por todas las razones anteriores, la especificación del sistema estaba totalmente obsoleta cuando finalizaba el desarrollo. En muchos casos, estaba incluso obsoleta cuando finalizaba la fase de análisis de requisitos. Debido a esto, la mayor parte del software de la época carece de una documentación fiable, incluso aunque se hubiese realizado análisis y redactado una especificación.

Por estos motivos fueron surgiendo nuevos métodos de análisis, cuyo objetivo era obtener especificaciones:

- **Gráficas**. Formadas por una colección de diagramas, acompañados de información textual detallada, que sirve de material de referencia, más que de cuerpo principal de la especificación.

- **Particionadas**. De forma que fuese posible leerse o trabajar sobre partes individuales de la especificación sin tener que leérsela toda.

- **Mínimamente redundantes**. De forma que los cambios en los requisitos necesitasen reflejarse en un sólo punto de la especificación.

- **Transparentes**. De forma que fuesen tan fáciles de leer y de entender que el que las utilizase no se diese cuenta de que está mirando una representación del sistema, en lugar del sistema en sí. Los sistemas son los que son complejos, las especificaciones tienen que ser claras y sencillas.

En este tema nos centraremos en los *Métodos de análisis estructurado*, que son los más utilizados en la actualidad. En los temas siguientes veremos métodos de análisis orientado a objetos y utilizando lenguajes de especificación formal.

5.2.- Técnicas de especificación y modelado

Bajo el nombre genérico de *Análisis estructurado*, se engloban una serie de aportaciones de diversos autores entre los que cabe citar, por orden cronológico, a De Marco, Yourdon, Gane & Sarson, Ward & Mellor y Hatley & Pirbhai. Cada uno de ellos ha desarrollado su propio método de análisis, mejorando, ampliando o adaptando los anteriores a algún campo de aplicación específico.

Siguiendo las técnicas del análisis estructurado podemos describir los sistemas desde tres puntos de vista:

Punto de vista de los datos. Dimensión de la información

Se centra en la información que utiliza el sistema. Representaremos el modelo de los datos que utiliza el sistema, haciendo explícitas las relaciones que se establecen entre esos datos. Para ello utilizaremos **Diagramas Entidad/Relación**.

El desarrollo del modelo de datos de un sistema evita el almacenamiento de información redundante e incoherente, garantiza la integridad y seguridad de los datos y simplifica el mantenimiento.

Punto de vista del proceso. Dimensión de la función

Se centra en qué hace el sistema. Para ello lo describiremos como un conjunto de operaciones de proceso de información. Estas operaciones reciben unos flujos de datos de entrada y los transforman en flujos de datos de salida. Para describir el sistema desde este punto de vista utilizaremos **Diagramas de Flujo de Datos** y **Especificaciones de procesos**.

Punto de vista del comportamiento. Dimensión del tiempo

Se centra en cuándo suceda algo en el sistema. Describiremos el sistema como una sucesión de estados o modos de funcionamiento. Indicaremos también cuáles son las condiciones o eventos que hacen que el sistema pase de un modo a otro. Utilizaremos **Diagramas de Flujo de Control**, **Especificaciones de Control** y **Diagramas de Estados**.

Cada sistema tiene una representación más o menos significativa en cada una de estas dimensiones y por tanto para cada sistema adquirirán más o menos importancia las técnicas que se centran en cada dimensión. Por ejemplo, un sistema de información basado en una gran Base de Datos tendrá una importantísima componente en la dimensión de la información, y por tanto, su representación se centrará en las técnicas que permiten modelar esta dimensión. Un sistema de gestión en tiempo real en cambio, como por ejemplo la centralita electrónica de encendido de un motor de combustión, tendrá el tiempo como componente más importante.

Aunque se han presentado dos casos extremos en los que una dimensión predomina sobre las demás hasta el punto en el que la representación en las otras podría ser obviada lo normal, en general, la mayoría de los sistemas requerirán utilizar modelos en varias dimensiones para representarlos. Cada modelo se centra en un número limitado de aspectos del sistema, dejando de lado otros (ésta era una de las características de los modelos). Combinando los diversos modelos podemos tener una visión detallada de todas las características del sistema. Esto es especialmente cierto hoy en día, cuando los sistemas software manejan estructuras de datos complejas, realizan funciones complejas y tienen pautas de comportamiento complejas.

Es importante dejar claro, sin embargo, que todos los modelos describen un único sistema y por tanto es razonable pensar en estrategias que nos permitan relacionar las distintas representaciones, y por tanto, la consistencia entre los modelos. Dichas

técnicas se encuentran en los planos formados por la combinación de dos dimensiones, por ejemplo, el plano Información-Función.

La tabla siguiente representa una posible clasificación de las técnicas según su dimensión. Las que tengan la misma fila y columna se centrarán en una dimensión mientras que las otras harán referencia al plano formado por las dimensiones indicadas por su fila y su columna.

	Información	Función	Tiempo
Información	Diagramas Entidad-Relación (ER).		
Función	Diagramas de Flujo de datos (DFD)	Diagramas de Flujo de datos (DFD).	
Tiempo	Diagramas de Historia y vida de entidad.	Redes de Petri Diagramas de transición de estados	Diagramas de transición de estados Diagramas de flujo de control

Fig. 5.1. Diferentes técnicas de modelado.

Dichas técnicas buscan modelar a alto nivel el sistema en cada una de sus dimensiones. Las técnicas siguientes dan el máximo nivel de detalle posible de aquel aspecto que representan y por tanto se presentan como técnicas de especificación.

	Información	Función	Tiempo
Información	Especificación de entidad		
Función		Diccionario de datos Especificación de procesos Especificación de entidades externas	
Tiempo		Definición de función	Especificación de eventos

Fig. 5.2. Diferentes técnicas de especificación.

Indicar únicamente que el **Diccionario de Datos** puede incluir, según el autor, la especificación de entidad, procesos, entidades externas, eventos y naturalmente la de los datos con lo que nos permite definir y relacionar todos los elementos presentes en las distintas representaciones.

En este punto vamos a estudiar las diferentes técnicas y notaciones de modelado de sistemas que puede utilizar un ingeniero del software. Combinando todas ellas se puede establecer un modelo completo del sistema, pero lo importante es usar aquellos diagramas que nos sirvan en una situación determinada.

Por último, hay que tener en cuenta la influencia que han tenido las herramientas de análisis en el uso y difusión de los métodos de análisis estructurado. Estas herramientas permiten dibujar los diagramas, hacen mucho más sencilla su modificación y comprueban su completitud y consistencia. Además muchas de estas herramientas sirven de soporte no sólo a la fase de análisis sino a todas las etapas del ciclo de vida. Estas son las herramientas CASE.

5.2.1.- Diagramas de flujo de datos.

Como su propio nombre indica, un sistema de procesamiento de datos incluye tanto datos como procesos, y cualquier análisis de un sistema así debe incluir ambos aspectos. Necesitamos una técnica para modelar sistemas que describa:

- Qué funciones son las que realiza el sistema.
- Qué interacción se produce entre estas funciones.
- Qué transformaciones de datos realiza el sistema.
- Qué datos de entrada se transforman en qué datos de salida.

A medida que la información se mueve a través del software, va siendo modificada mediante una serie de transformaciones. El DFD es una técnica gráfica que utiliza un diagrama en forma de red para representar el flujo de información y las transformaciones que se aplican a los datos al moverse desde la entrada a la salida.

Elementos de un DFD.

Para representar el sistema mediante DFDs utilizaremos la notación de Yourdon (1975), posiblemente la más extendida. Esta notación es la indicada en la transparencia 4.3. Los elementos que aparecen en el DFD pueden ser:

Procesos. Representan elementos software que transforman información. Son, por tanto, los componentes software que realizan cada una de las funciones del sistema, transformando datos de entrada en datos de salida. Los representaremos como cuadrados de esquinas redondeadas, que llevan asociado un número y un nombre de proceso.

- *Regla de conservación de datos*: El proceso debe ser capaz de generar las salidas a partir de los flujos de entrada más una información local
- *Pérdida de información*: Una entrada no se utiliza en un proceso para generar ningún flujo de salida

Entidades externas. Representan elementos del sistema informático o de otros sistemas adyacentes (en cualquier caso se trata de algo que está fuera de los límites del sistema software) que producen información que va a ser transformada por el software o que consumen información transformada por el software. Los flujos de datos que comuniquen el sistema con las entidades externas representan las interfaces del sistema. Los flujos entre unidades externas no son objeto de estudio y nunca deben representarse, si son necesarios hay que replantearse los límites del software. Las entidades externas sólo aparecen en el diagrama de contexto. Serán representados por un cuadrado con el nombre identificativo de la entidad externa.

Almacenes de datos. Representan información almacenada que puede ser utilizada por el software. Los almacenes de datos permiten guardar temporalmente información que luego puede ser procesada por el mismo proceso que la creó o por otro distinto. En la mayoría de los casos, utilizaremos almacenes de datos cuando dos procesos intercambian información pero no están sincronizados, esto es, el proceso destino no comienza a procesar la información en cuanto le llega. En otros casos,

utilizaremos los almacenes como copia de seguridad de los datos, para evitar pérdidas de información en caso de que el sistema falle. Los almacenes de datos pueden ir desde registros temporales para almacenar un dato hasta ficheros o bases de datos. En nuestro caso los representaremos por dos barras paralelas unidas por sendos arcos.

Flujos de datos. Representan datos o colecciones de datos que fluyen a través del sistema. La flecha indica el sentido de flujo. Posiblemente en los diagramas de nivel mayor existan flujos de datos bidireccionales (par de diálogo), que luego son refinados en sucesivos diagramas, o incluso varios flujos de datos agrupados en uno sólo (flujos múltiples). Los flujos de datos conectan los procesos con otros procesos, con entidades externas o con almacenes de datos, y pueden converger o divergir si conectan un elemento del DFD con varios otros. Mientras que los almacenes de datos representan información estática o en reposo, los flujos de datos representan información en movimiento. Puede tratarse de un elemento de datos simple o compuesto (un registro) o incluso de una colección de datos de estructura compleja, p.ej. un árbol. En algunos casos, el flujo de datos puede representar elementos que no son datos, sino p.ej. materiales, si estamos haciendo un diagrama de flujo de una cadena de producción, por ejemplo. Los flujos de datos se representan por flechas que llevan asociado un nombre.

- Por tanto, los flujos contienen información de las tres dimensiones aunque normalmente sólo se representan la de Función e Información:
  - Según su *dimensión temporal* los flujos pueden ser:
    - Discretos: Si representan movimientos de datos en un instante determinado del tiempo. ⟶
    - Continuos: Si implican una transición continua de información, por ejemplo para comprobar si un registro ha cambiado. ⟶⟶
  - Según la *dimensión de la Función* los flujos pueden ser:
    - De consulta: Salen de un almacén. Utilizan la información del almacén pero no la modifican.
    - De actualización: Entran en un almacén. Crean, modifican o borran datos del Almacen.
    - De Dialogo: Representa simultáneamente un proceso de Consulta y otro de actualización. Flujo de doble flecha y con dos nombres, uno hace referencia al iniciador y el otro es la respuesta Por ejemplo un gestor de almacén de piezas mecánicas podría comprobar si hay un tipo de pieza y si la hay indicar que queda una menos; ambas acciones quedarían reflejadas en un flujo de diálogo.
  - Finalmente en la *dimensión de la Información* los flujos pueden ser de varios tipos atendiendo a su contenido que puede ser:
    - Un elemento: Contiene un dato elemental o una pieza indivisible de información.
    - Un Grupo: Contiene varios datos elementales
    - Múltiples: En el DFD se representa como un único flujo pero en realidad está formado por un conjunto de ellos.

Cualquiera de los elementos que aparecen en un DFD tiene que estar etiquetado con un nombre, corto y significativo que debe ser único en el conjunto de DFDs que representan al sistema. Los procesos van etiquetados con la función que realizan. Lo mismo sucede con las entidades externas, que también se pueden etiquetar con el

nombre de la máquina, persona o grupo de personas que realizan esa función, en el caso de que no se trate de software. Los flujos de datos van etiquetados con un nombre identificativo de la información que transportan, y posiblemente con el estado de dicha información (p. ej. *número de teléfono, número de teléfono correcto, número de teléfono erróneo*). Los almacenes de datos van etiquetados con un nombre significativo de la información que contienen, generalmente en plural. Es importante destacar que los nombre deben ser representativos lo que quiere decir que deben hacer referencia a toda la función o información de aquello que identifican y no solamente a una parte y que deben evitarse que sean poco significativos tales como "Realizar operación" o "gestionar acción".

Hay que tener en cuenta que un DFD no representa información sobre el comportamiento del sistema o sobre el control del mismo. Representa qué funciones o que transformaciones se realizan sobre los datos pero no cuándo se realizan o en qué secuencia.

Diagrama de contexto.

Se pueden utilizar DFDs para representar el sistema a cualquier nivel de abstracción. El DFD de nivel 0 se llama **diagrama de contexto** y en él el sistema está representado por un sólo *proceso*, que identifica cuál es la función principal del sistema, mostrando además los flujos de información que lo relacionan con otros sistemas: las *entidades externas*. El diagrama de contexto tiene una gran importancia puesto que resume el requisito principal del sistema de recibir ciertas entradas, procesarlas de acuerdo con determinada función y generar ciertas salidas. A partir del diagrama de contexto podemos ir construyendo nuevos diagramas que vayan definiendo con mayor nivel de detalle los flujos de datos y procesos de transformación que ocurren en el sistema, de forma que al final obtenemos una jerarquía de diagramas.

En general, cualquier proceso que aparezca en un DFD puede ser descrito más detalladamente en un nuevo. DFD. A esto lo llamaremos **explosión** de un proceso. En este DFD el proceso que estamos describiendo aparece descompuesto en una serie de subprocesos o subsistemas, cada uno encargado de realizar un aspecto determinado del proceso original. Los flujos de datos que entraban y salían del proceso que estamos describiendo deben entrar y salir del DFD que lo desarrolla. Además de estos flujos, el DFD contendrá por lo general nuevos flujos que comunican los procesos que figuran en él y posiblemente almacenes de datos. Las entidades externas sólo aparecen en el DFD de contexto.

Dado que en el DFD de contexto nuestro sistema estará representado por un sólo proceso. Este DFD de nivel 0 sólo dará lugar a un DFD de nivel 1. A su vez, este puede dar lugar a tantos DFDs de nivel 2 como procesos contenga, y así sucesivamente hasta que hayamos alcanzado un nivel en el que los procesos sean lo suficientemente simples como para no necesitar su descripción más detallada en un DFD. De este forma, el modelo de procesos del sistema va a consistir en una jerarquía de DFDs.

Una ventaja de los DFDs es que no sólo se utilizan para representar modelos del software sino también en muchos otros campos, como la investigación operativa o la

organización empresarial. Eso hace que estén ampliamente difundidos y que sea relativamente fácil mostrarlos al usuario y que éste los entienda.

DFDs. Reglas de construcción.

- *Un DFD debe contener menos de 10 elementos.*
- *Cada elemento de un DFD debe tener un nombre corto e identificativo.*
- *Es necesario numerar los procesos.*
- *Para modelar sistemas complejos se utiliza la explosión, que da como resultado DFDs a distintos niveles de detalle.*
- *No es conveniente utilizar más de 7 u 8 niveles.*
- *Los DFDs de niveles inferiores desarrollan de forma más concreta los procesos de niveles superiores.*
- *La explosión se realiza hasta alcanzar un nivel de especificación mínimo y sencillo.*
- *Debe mantenerse la consistencia de nombres en los distintos DFDs.*
- *Debe mantenerse la consistencia entre los distintos niveles, utilizando la regla de balanceo.*
- *En cada DFD hijo deben representarse los mismos flujos de datos que en el proceso padre.*
- *No existen conexiones entre entidades externas*
- *No existen conexiones entre entidades externas y almacenes*
- *No existen conexiones entre almacenes*

Regla de balanceo.

Normalmente, el sistema que estamos modelando será lo suficientemente complejo como para necesitar varios DFDs. Construiremos entonces una jerarquía de DFDs.

Cada DFD (hijo) de un nivel *n* será resultado de la explosión de un proceso (padre) de un DFD de nivel *n-1*. Es necesario que el título del DFD sea el nombre del proceso que desarrolla, que la numeración de los procesos en el DFD hijo se derive del número del DFD padre (p.ej. 3, 3.1, 3.2) y además hay que mantener la consistencia de los flujos de datos en ambos.

La regla de balanceo dice que los flujos de datos que entran o salen del proceso padre deben aparecer en el DFD hijo, manteniendo el nombre y el sentido.

En el diagrama hijo estos flujos de datos tendrán un extremo libre, puesto que conectaban el proceso padre con algún elemento que ya no va a estar representado en el diagrama hijo. Una excepción son los flujos que conectan los procesos con almacenes de datos. Se recomienda representar los almacenes de datos en todos los DFDs en los que intervienen.

La regla de balanceo también puede tener excepciones. Antes habíamos hablado de flujos de datos compuestos, e incluso bidireccionales (que normalmente transmiten

una pregunta o petición y su respuesta). Estos flujos de datos suelen utilizarse en los DFDs de los niveles más bajos (es decir, los más altos de la jerarquía) para modelar el sistema con cierto grado de abstracción y para evitar representar muchos flujos de datos, lo que haría más confuso el diagrama. En los DFDs de niveles superiores, es necesario descomponer estos flujos en varios (p.ej. petición y respuesta). Para realizar esta descomposición podemos hacer divergir el flujo en el DFD hijo o al menos dejar reflejada esta descomposición en el diccionario de datos (que ya se verá qué es).

Ejemplos:

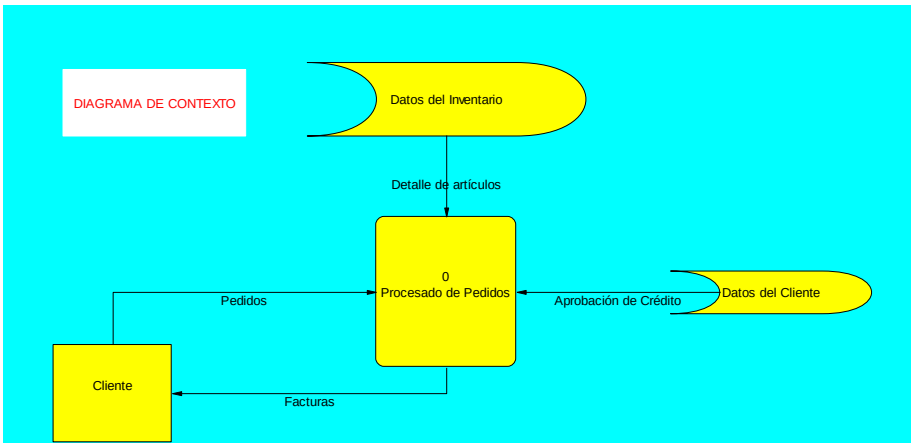

Fig. 5.3. Diagrama de contexto de un software de procesado de pedidos

Imaginémonos que tenemos que desarrollar un software para gestionar los pedidos que recibe una determinada empresa. El diagrama de contexto será el que tenemos en la figura 5.2. Simbolizamos todo el sistema como un solo proceso (cuadrado con esquinas redondeadas) que interactúa con el resto del "mundo". Este mundo o entorno estaría formado por el cliente (cuadrado con esquinas, ya que es un agente externo) que efectúa pedidos a nuestro software y recibe facturas de él (dos flujos de datos con sentidos diferentes), por la base de datos del inventario (externa a nuestro software) contra la que confirmaremos la existencia de unidades suficientes para satisfacer el pedido del cliente, y la base de datos fiscales de nuestros clientes de la que nuestro software extraerá información para generar la factura correspondiente.

5.2.2.- Especificaciones de proceso

Los DFDs permiten identificar las principales funciones o transformaciones que realiza un sistema, y las relaciones entre estas funciones pero no indican nada acerca de los detalles de cómo se realizan estas transformaciones. Para definir los detalles de qué información de entrada se transforma en qué información de salida y cómo se realiza esta información se necesita una descripción textual de los procesos. Para esto utilizaremos las especificaciones de proceso.

En los DFDs de menor nivel (los más altos en la jerarquía), los procesos se describen mediante un nuevo DFD, que define más detalladamente las funciones que realiza y los flujos que maneja. Este proceso de descomposición debe continuar hasta

que se alcance un nivel en el que un proceso puede ser descrito textualmente de forma sencilla y no ambigua. Estos procesos de los nodos hoja de la jerarquía se suelen llamar **primitivas de proceso**.

Una forma de especificar una primitiva de proceso es dando un algoritmo. Esto no quiere decir que el algoritmo propuesto en el análisis se corresponda con la implementación final del proceso, sino que es simplemente una forma de describir la operación. La elección del algoritmo definitivo se hace en la fase de diseño, considerando además de la especificación, criterios de eficiencia y las características del lenguaje de implementación.

Alternativamente, una primitiva de proceso puede ser descrita mediante una definición, sin especificar un algoritmo (p.ej. el proceso *Invertir Matriz* calcula la inversa de la matriz que recibe como entrada), matemáticamente o mediante un lenguaje de especificación formal, en lenguaje natural o mediante una tabla que indica los valores de los flujos de salida a partir de los valores de los flujos de entrada.

En cualquier caso, las especificaciones de proceso son textos breves (unas pocas líneas, en cualquier caso menos de una página), que definen la función que realiza la primitiva de proceso para transformar los flujos de entrada en flujos de salida.

5.2.3.- Diagramas de flujo de control.

Los DFDs son muy útiles a la hora de modelar sistemas de proceso de datos, describen las funciones o transformaciones que realiza el sistema y cómo fluyen los datos a través del mismo.

Sin embargo, en los DFDs no se representa explícitamente el control o flujo de sucesos del sistema, por lo que estos diagramas no aportan ninguna información sobre cómo se comporta el sistema, sobre cuándo se realizan los procesos o en qué orden. En determinados sistemas de software de gestión, este funcionamiento puede ser intuitivo, y no necesitaremos reflejarlo en el modelo, pero en otras aplicaciones (p. ej. relacionadas con sistemas de tiempo real o con automatismos) sí que es importante que el modelo incluya los aspectos de comportamiento y control del sistema.

Podemos considerar tres situaciones posibles, referidas a cómo se comporta el sistema:

- Los procesos que figuran en el DFD están activos siempre. En este caso no necesitamos especificar el control del sistema.
- Los procesos se activan cuando llegan datos a través de sus flujos de entrada, transforman estos datos y emiten los resultados a través de los flujos de salida, permaneciendo entonces inactivos hasta la llegada de nuevos datos. Este comportamiento está implícito en la notación usada para los DFDs por lo que tampoco será necesario especificar el control. Muchas de las aplicaciones de gestión pueden ser descritas de esta forma.
- Cada proceso pasa por periodos de actividad e inactividad que están gobernados por mecanismos más complejos que los descritos anteriormente. Por lo general, un proceso se activará cuando se produzca determinada situación o suceso en el sistema y permanecerá activo hasta

que se produzca otra situación. Comportamientos de este tipo no pueden ser reflejados de forma adecuada en los DFDs por lo que en estos casos necesitaremos describir el modelo de control o comportamiento del sistema, al menos para estos procesos que siguen patrones de comportamiento complejos. En aquéllas aplicaciones en las que el software controle el funcionamiento de otros dispostivos (p. ej. en un sistema empotrado) nos encontraremos con situaciones de este tipo.

Nota: *Hay que tener en cuenta que el modelo de control del sistema, establece el comportamiento de éste a alto nivel: qué procesos se encuentran activos en cada momento o en que secuencia se realizan las transformaciones de los datos. Existe un control de bajo nivel que se refiere a los saltos condicionales y a los bucles de los programas. Este control de bajo nivel estaría reflejado en las PSPECs de los procesos.*

Estas falta de expresividad de los DFDs llevaron a proponer varias extensiones de los mismos para modelar el control del sistema. Las más importantes son las de Ward & Mellor (1985) y Hatley & Pirbhai (1987). Nosotros vamos a seguir la notación de estos últimos.

Para modelar el control del sistema Hatley y Pirbhai proponen el uso de Diagramas de Flujo de Control (DFCs), similares a los DFDs ya vistos. Su método se basa en eliminar de los DFDs todo lo relativo a información de control : sucesos, señales, condiciones de datos, etc. y construir una jerarquía de DFCs paralela a la de DFDs. En estos DFCs se especifica todo el flujo de sucesos, señales y condiciones de datos del sistema, es decir, se indican los elementos de información que intervienen en el control de los procesos.

Nota: *Otros autores (W & M) proponen incluir toda la especificación del control en los DFDs, por lo que no utilizan DFCs. Hatley y Pirbhai prefieren separar ambos modelos, para dejar más claro lo que es procesamiento y lo que es control, y para evitar recargar los DFDs con más información. No obstante, en sistemas sencillos con mecanismos de control sencillos sería admisible utilizar un único diagrama combinado, siempre y cuando quede reflejado lo que son datos y lo que son eventos.*

Según lo anterior, construiremos una jerarquía de DFCs , empezando por un diagrama de contexto, de forma que a cada DFD (o al menos para aquéllos que sea necesario representar su control) le corresponda un DFC. En cada par DFD/DFC representaremos los mismos procesos y las mismas entidades externas, puesto que ambos representan modelos de la misma parte del sistema con el mismo nivel de detalle, aunque con puntos de vista distintos.

Las reglas sobre denominación numeración, relaciones padre - hijo y balanceo que se aplican a los DFCs son las mismas que se establecen para los DFDs.

Los elementos que aparecen en un DFC son prácticamente los mismos que en los DFDs.

- **Procesos, entidades externas y almacenes de datos**. Serán los mismos y tendrán el mismo significado que en el DFD al que corresponden. Se incluyen en el DFC básicamente como referencia, para mostrar a qué procesos afectan los mecanismos de control que estamos describiendo.
- **Flujos de control**. Se representan mediante trazos discontinuos y modelan el flujo de información de control en el sistema. Habrá procesos o entidades externas que generen información de control y otras que la consuman. La única diferencia es que los flujos de control transportan señales discretas (normalmente lógicas o de tipo enumerado) y son impulsos o eventos, es decir tienen una duración instantánea, mientras que los flujos de datos pueden transportar cualquier tipo de datos (posiblemente estructuras de datos complejas) y pueden transformar datos de manera continua.
- **Almacenes de control**. Se representan igual que los almacenes de datos pero con trazos discontinuos. Permiten almacenar información de control, para ser utilizada posteriormente.
- **Ventanas a especificaciones de control**. Se representan mediante barras. Estas ventanas reciben y emiten flujos de control y representan la transformación de flujos de control en el sistema.

Los procesos de un DFC no representan procesamiento ni transformación de los flujos de control (lo que se hace en las CSPECs) ni tampoco representan los estados del sistema (que se representan en los DEs). Simplemente representan a los mismos procesos de los DFDs, y lo que indica el DFC es simplemente qué flujos de control reciben o generan estos procesos.

Otro detalle importante es que un flujo de control que entra en un proceso no indica que ese proceso se active mediante ese flujo. La activación y desactivación de procesos se indica en la CSPEC. Un flujo de control que entra en un proceso puede indicar dos cosas: bien que va a ser utilizado como una dato más para que el proceso lleve a cabo su función de transformación, o bien que va a ser utilizado para controlar alguno de los procesos en los que se descompone éste. Del mismo modo, un flujo de control que sale de un proceso indica simplemente que ha sido generado por éste e intervendrá en el control del comportamiento de algún otro.

Las especificaciones de control figuran normalmente en un nivel alto de la jerarquía de diagramas. Se encargan de controlar el funcionamiento del sistema, activando o desactivando sus procesos. A bajo nivel, mucho procesamiento de control puede representarse en las primitivas de proceso sin que esto influya en los resultados del análisis.

Cómo separar datos y control.

De entre todos los flujos de información que maneja un sistema software, ¿cómo podemos clasificar unos como datos y otros como control? y ¿qué procesamiento corresponde a procesos de datos y qué procesamiento corresponde al control del

sistema? (los procesos de control no existen como tales en la notación H & P, sino que están incluidos en las CSPECs).

No hay unas normas estrictas para distinguir entre unos y otros, y en algunos casos se puede hacer de forma arbitraria. Como criterio de decisión utilizaremos el siguiente:

Modelaremos como señales de control únicamente aquéllas que intervengan en la activación y desactivación de algunos de los procesos del sistema. El resto las modelaremos como datos.

En determinadas situaciones, un elemento de información determinado se utiliza como dato en un proceso y como control en otro. En este caso, podemos modelarlo como dato (trazo continuo) en el DFD y como control (trazo discontinuo) en el DFC, pero asignaremos a ambos flujos el mismo nombre para mostrar que son el mismo.

Cuándo usar especificaciones de control.

Lo más conveniente es utilizar DFDs siempre que sea posible, puesto que son más sencillos de realizar y de entender (la notación de los DFCs es más oscura y el uso de ambos obliga a mirar los dos a la vez). Sólo para aquellos aspectos del sistema que no podamos modelar con DFDs utilizaremos DFCs. Esto significa reducir el control en la medida de lo posible.

Pese a esto, a la hora de hacer el análisis de requisitos del sistema existe la tendencia de profundizar demasiado en el modelo de control del sistema. Con frecuencia se especifican detalles de implementación como flags, contadores, llamadas a procedimientos e interrupciones, que no tienen nada que ver con los requisitos del sistema. En el análisis de requisitos nos debemos centrarnos en el **modelo abstracto o lógico** del sistema más que en su implementación. Según esto el modelo de control debe ser usado para describir la lógica que controla los procesos principales del sistema y no para describir de forma detallada interacciones entre primitivas de proceso. Esto último puede ser descrito perfectamente utilizando el modelo implícito en los DFDs, que consiste en que cuando un proceso recibe un dato lo procesa de forma inmediata e instantánea. Como este no es el funcionamiento que tendrá el sistema real, existe la tendencia de enviar, junto con el dato, una señal de control que no tiene más objetivo que activar el proceso receptor cuando llegue el dato. Esto no es necesario y no debería hacerse.

Un ejemplo muy común de especificación de control excesiva es el caso en que un proceso genera flujos de salida alternativos, de los cuales sólo se usa uno de cada vez. Podemos entonces generar señales de control para activar/desactivar los procesos receptores pero esto no es un requisito del sistema. Desde el punto de vista de los requisitos podemos dejar todos los procesos funcionando en paralelo; sólo procesarán información cuando reciban entradas. Haciendo esto, podemos expresar los requisitos de forma sencilla y dejar las manos libres a los diseñadores para decidir la implementación más adecuada, siempre y cuando se mantengan las transformaciones de entradas en salidas.

¿Para qué sirven los DFCs?

Considerado aisladamente, solo muestran la información de control que existe en el sistema, junto con los procesos y entidades externas que generan y consumen dicha información de control. Para representar el comportamiento del sistema, hay que utilizarlos en combinación con las especificaciones de control.

5.2.4.- Especificaciones de control

Las especificaciones de control (CSPECs) son similares a las PSPECs. En ambos casos, la función de estas especificaciones es definir los detalles procedimentales de cómo se realiza el procesamiento de los flujos de entrada y salida.

Sin embargo, hay una diferencia importante entre PSPECs y CSPECs. Las PSPECs se utilizan para describir las primitivas de proceso: aquéllos procesos del sistema que son lo suficientemente simples como para no necesitar crear un nuevo DFD. Las CSPECs sirven para modelar el comportamiento de un DFC, describiendo cómo se procesan los flujos de control. Habrá entonces como máximo una CSPEC por cada DFC de la jerarquía. La interfaz entre el DFC y la CSPEC se hace a través de las ventanas de control que aparecen en los DFCs.

Las CSPECs muestran cómo, a partir de las señales de control que entran en la ventana, se determina la activación o desactivación de procesos que figuran en el DFC correspondiente. Es decir, muestran cómo se calculan unos flujos de control de salida (los activadores de los procesos) a partir de otros de entrada (las señales de control que entran en la ventana).

Las CSPECs se caracterizan mediante sistemas combinacionales o, más frecuentemente, mediante sistemas secuenciales. Si son combinaciónales su representación se hace mediante Tablas de Decisión (tablas de verdad que indican cómo se calculan las señales de salida en función de las de entrada) o mediante Tablas de Activación de Procesos (iguales que las anteriores, en las que se indica para cada proceso del DFC si esta activo o inactivo ante cada combinación de las señales de entrada); si son secuenciales se representan mediante Diagramas de Estados, que no son más que autómatas finitos.

Al hablar de los DFCs habíamos dicho que los flujos de control pueden figurar como salida de alguno de los procesos del sistema. En este caso se denominan condiciones de datos. Como consecuencia del procesamiento de los flujos de datos de entrada del proceso (procesamiento que se describe en la PSPEC asociada al proceso o a alguno de sus descendientes) se genera un flujo de control, que va a intervenir en el comportamiento de otro proceso del sistema. Estos flujos son el único enlace que se establece del modelo de procesos al modelo de control.

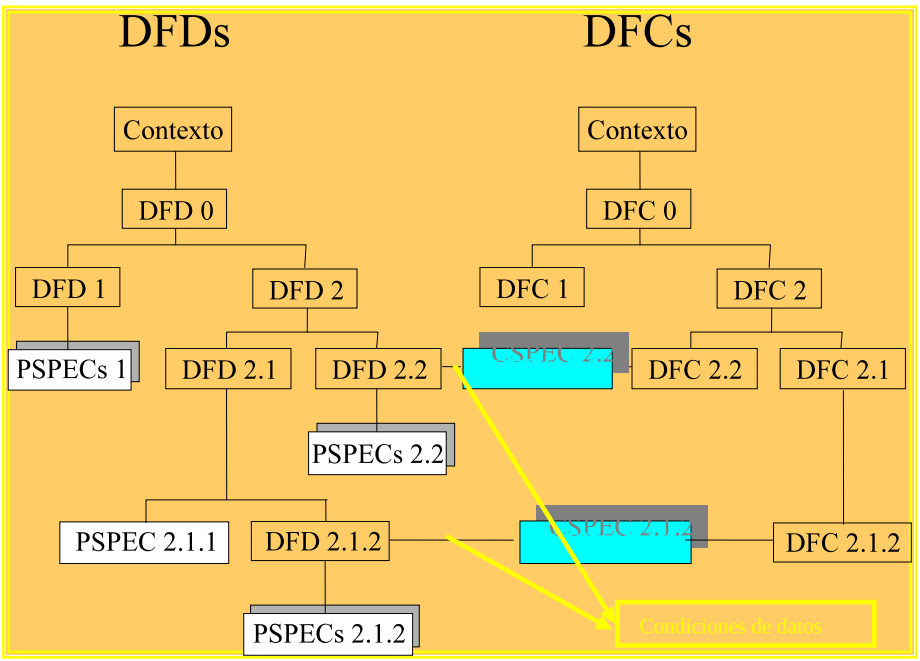

Figura 5.4.- Relación entre DFD y DFC

Condiciones de datos

Ej. El proceso *Comprobar Saldo* procesa *Número de Cuenta* e *Importe* y genera dos flujos de control: *Aceptar Operación* y *Rechazar Operación*.

Resumiendo, las condiciones de datos son flujos de control generados por un proceso del sistema. Figuran en el DFC y no en el DFD, pero es la PSPEC la que define cómo se calcula su valor a partir de los flujos de entrada del proceso.

Ventanas de control.

Estas condiciones de datos, junto con otros flujos de control provenientes del exterior del sistema (de las entidades externas) son los que deciden el comportamiento del sistema. Este comportamiento se define en las CSPECs que indican cómo se generan unos flujos de control a partir de otros o, más concretamente, cómo se activan o desactivan los procesos.

Para hacer referencia a que un flujo de control se calcula a partir de otros y a que esta transformación está definida en la CSPEC se utilizan las ventanas de control en los DFCs. Estas ventanas se representan mediante barras, donde entran y salen flujos de control. Las ventanas de control no están etiquetadas, puesto que todas las que aparecen en un determinado DFC hacen referencia a una única CSPEC que define cómo se realizan estas transformaciones.

Ej. El cajero automático utiliza dos flujos de control (*Saldo suficiente* e *Importe disponible en cajero*) para calcular los flujos de control *Activar Aceptar Operación* y *Activar Rechazar Operación*.

En un DFC podemos utilizar tantas ventanas de control como sea necesario con objeto de que el diagrama sea claro y todas ellas hacen referencia a la misma CSPEC.

Activadores de procesos.

El objetivo del modelo de control es definir el comportamiento del sistema, es decir, cuándo se activan o desactivan los procesos. Estas activaciones y desactivaciones se modelan mediante unos flujos de control especiales, denominados Activadores de procesos. Estas señales de control toman sólo dos valores: On y Off. Los flujos activadores de procesos tienen el mismo nombre que los procesos que controlan, por lo que no suelen representarse en el DFC, salvo con fines de claridad (identificar qué procesos tienen activador).

En general, los flujos de control que entran en un proceso no sólo sirven para activarlo. Se utilizan también para transformar los flujos de entrada en flujos de salida según se describe en la PSPEC correspondiente o para intervenir en el control de algún subproceso de éste.

Dado que los modelos del sistema tienen estructura jerárquica, consideraremos que un proceso esta activo **sólo si todos sus antecesores (en una jerarquía superiror) están activos**. Esto es lógico si tenemos en cuenta que los descendientes de un proceso son subprocesos que están incluidos en él. Cuando se desactiva un proceso, el sistema se comporta cómo si este proceso **y todos sus descendientes** no existiesen, y se considera que sus salidas toman valor nulo.

Un proceso que no tiene activador se considera activo siempre que sus antecesores lo estén, y responde a los datos de entrada según le van llegando (este es el modelo de control implícito).

Relaciones detalladas entre DFDs, DFCs, PSPECs y CSPECs

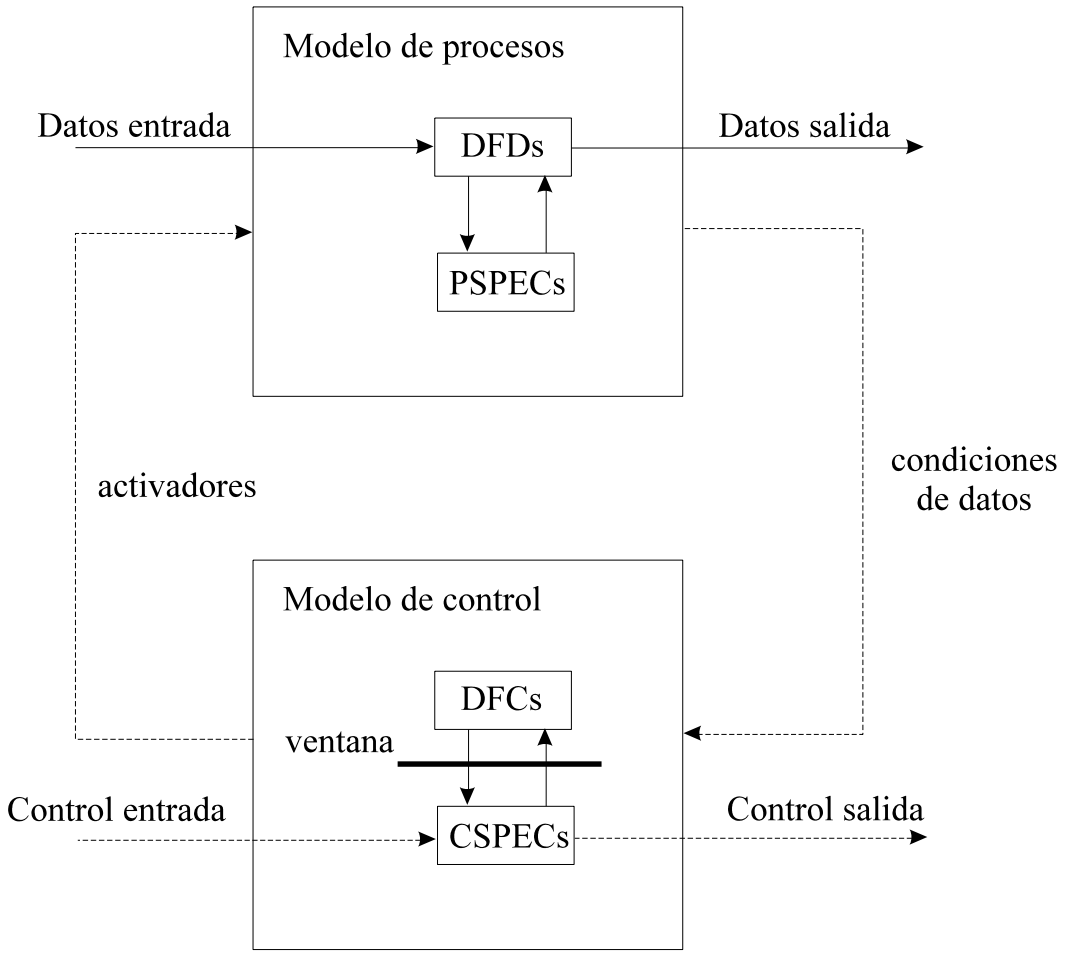

Figura 5.5.- Detalle del funcionamiento de las ventanas a especificaciones de control

En resumen, por cada DFD dibujaremos un DFC gemelo si alguno de los procesos del DFD genera o consume información de control. Además, si alguno de los procesos del DFD es activado o desactivado de forma no implícita, éste DFC llevará una ventana de control y haremos una CSPEC para indicar cuándo se realiza la activación y desactivación.

Una vez que hemos establecido la necesidad de utilizar CSPECs, debemos estudiar si podemos representarlas mediante un sistema combinacional (lo que podremos hacer si la función de control depende únicamente de los valore actuales de las señales de control disponibles) o si necesitamos un sistema secuencial (si la función de control depende de valores anteriores de las señales de control, que ya no están disponibles o si el comportamiento del sistema depende de su evolución anterior. Sólo en este caso utilizaremos CSPECs en forma de DEs.

5.2.5.- Diagramas de estados.

Los sistemas complejos suelen presentan la propiedad de que los eventos pasados influyen en su comportamiento. Estos cambios no se limitan a cambios en los valores de

los flujos de salida, sino que las computaciones pueden cambiar totalmente o incluso desaparecer, y el sistema se comporta de una forma completamente diferente a lo largo del tiempo. Este tipo de comportamiento es difícil de representar utilizando un DFD, pero podemos utilizar técnicas basadas autómatas secuenciales. La forma más habitual de representar autómatas secuenciales es utilizando DEs.

Hay dos modelos muy conocidos de autómatas secuenciales: las máquinas de Moore y las máquinas de Mealy. En las primeras, las salidas se asocian con los estados, mientras que en las segundas se asocian con las transiciones. En los DEs utilizaremos indistintamente uno u otro, incluso un modelo mixto donde sea necesario.

El DE representa el comportamiento de un sistema, mostrando los estados en los que puede estar y los sucesos que hacen que el sistema cambie de estado. Además, el DE indica qué acciones se realizan cuando un sistema cambia de estado y qué actividades se realizan mientras el sistema está en un estado.

Elementos de un DE

Los elementos que aparecen en el DE son dos, Estados y Transiciones:

Estados. Representados mediante rectángulos de esquinas redondeadas. Los estados muestran los distintos modos de comportamiento, escenarios o situaciones en que puede encontrarse el sistema. Cada estado representa un periodo de tiempo en el que el sistema muestra un cierto comportamiento. Generalmente los estados se asocian a la realización de un proceso o a un grupo de procesos del DFD. En otros casos, los estados representan al sistema *esperando* la ocurrencia de un determinado suceso (una petición de servicio o una entrada de datos, p.ej.).

Los estados tienen un nombre que los identifica y pueden ir etiquetados con una **actividad**: aquellos procesos que están activos mientras el sistema esté en dicho estado. Las actividades se corresponden con las señales de salida de una máquina de Moore.

Uno de los estados será el **estado inicial**, aquél en el que se sitúe el sistema cuando comience su funcionamiento (cuando se arranque el programa o se encienda el interruptor general). Un DE puede contener también **estados finales**, de los cuales no se salga mediante ninguna transición. Los estados iniciales y finales están marcados en el DE.

Transiciones. Muestran las evoluciones posibles entre los estados de un sistema y se representan mediante flechas. Indican cuándo *evoluciona* el sistema de un estado a otro. Las transiciones van etiquetadas con dos elementos: la **condición** que hace que se dispare la transición y la **acción** que se produce como consecuencia de la transición. Las acciones corresponden con las señales de salida de una máquina de Mealy.

Las condiciones son generalmente condiciones de datos (esto es, flujos de control) y en este caso se denominan **eventos**, aunque para evitar tener que introducir señales ficticias en el modelo del sistema podemos establecer condiciones asociadas a flujos de datos (la ocurrencia de una determinada entrada de datos, o el que una señal tome determinado valor). En este caso se denominan **guardas** y se representan entre corchetes. Según esto la condición de disparo de una transición está formada por

guardas y/o eventos. Para que una transición se dispare deben cumplirse sus guardas y producirse los eventos con los que esté etiquetada.

Las acciones consisten en generar un suceso o flujo de control o en activar un proceso del DFD (asignar valor *On* a un activador de procesos).

Tanto los eventos como las acciones son flujos de control, y deben figurar en el DFC (exceptuando los activadores de procesos, que normalmente no aparecen en los DFCs).

En el modelo de comportamiento del sistema consideraremos que las transiciones se realizan de forma instantánea, es decir, que al sistema no le lleva tiempo cambiar de estado. Esta no será la situación real del sistema una vez que lo implementemos, pero ahora estamos intentando establecer un modelo lógico o abstracto del sistema.

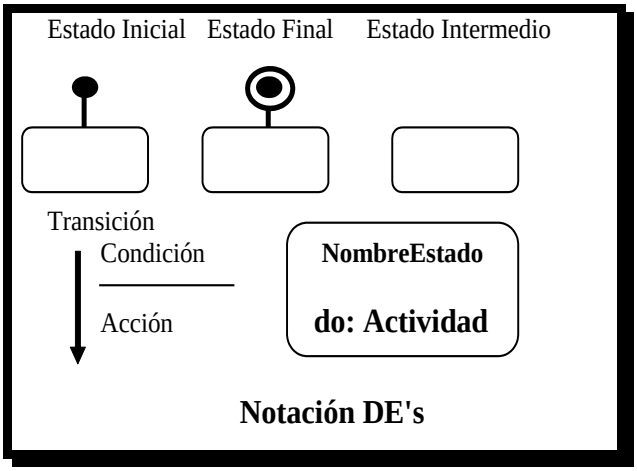

Figura 5.6.- Notación para los diagramas de estado

Como vemos, en el DE podemos representar las computaciones de dos formas distintas: como **acciones** (de duración idealmente instantánea con respecto a la escala de tiempo que utiliza el modelo) o como **actividades** (de duración prolongada en el tiempo, durante todo el intervalo de tiempo en el que el sistema permanezca en el estado correspondiente).

Con respecto a la activación de procesos, si realizamos ésta en un estado, consideraremos que dicho proceso permanece activo mientras el sistema permanece en dicho estado, desactivándose cuando lo abandone. Si, por el contrario, realizamos la activación en la transición supondremos que el proceso sigue activo hasta la siguiente activación, es decir, mientras el sistema permanezca en el estado de salida de la transición. Como se puede apreciar, ambos casos son equivalentes, y no es necesaria la desactivación explícita de procesos.

El DE muestra, por tanto, el comportamiento del sistema. Estudiando el DE podemos comprobar si hay lagunas en el comportamiento especificado: si hay estados de los que no se sale mediante ninguna transición o hemos considerado todas las transiciones posibles (esto es si en un estado hemos previsto todos los sucesos que pueden ocurrir y causar una evolución del sistema).

Esto último es muy importante: supongamos un sistema con tres estados. Del estado inicial A, pasamos al estado B pulsando la tecla F1, y del B pasamos al C pulsando F2. El DE de la figura modela este comportamiento, pero no podemos quedarnos en modelar el comportamiento del sistema basándonos en un *comportamiento correcto del usuario*. ¿Qué sucede si el usuario pulsa dos veces la misma tecla? ¿O si pulsa cualquier otra? Si no especificamos el comportamiento del sistema ante estos eventos, posiblemente los programadores no crearán código para estas situaciones y el sistema final podrá presentar un comportamiento incontrolado, ante un usuario poco experto o malicioso. No podemos hacer suposiciones sobre el buen uso del sistema y tendremos que intentar dejarlo siempre todo bien atado.

Nosotros utilizaremos los DEs para modelar el comportamiento de todo el sistema o una parte de él, incluyendo estos diagramas en las CSPECs. Por tanto, los DEs van a ir asociados a un par DFD/DFC, indicando cómo se procesan los flujos de control que entran en las ventanas de control de los DFCs y cómo se activan los procesos que figuran en los DFDs. Los flujos de control que entran en la ventana van a ser condiciones de las transiciones del DE. Los flujos que salen de la ventana serán establecidos en las acciones y actividades de dicho DE.

Composición de DEs.

En algunos sistemas, los procesos de un DFD se activan y desactivan de forma más o menos independiente. Este es el caso de un sistema con dos o más procesos concurrentes que no interactúan entre sí o lo hacen de forma limitada. En estos casos realizaremos un DE compuesto, es decir formado por un DE para cada proceso o grupo de procesos que se comportan de forma independiente.

Ej. En la máquina expendedora (ejercicio 1), el control de la aceptación de monedas y el control de la selección de productos son prácticamente independientes. La única interacción entre ellos se produce a través del almacén IMPORTE. El DE se compone entonces de dos: uno para cada subsistema.

Anidamiento de DEs.

Igual que los DFDs y DFCs, los DEs también pueden anidarse, en caso de que sea necesario. Esto será útil cuando el DE sea complejo, y sea conveniente mostrarlo en varios niveles de abstracción o cuando un grupo de estados tienen un comportamiento común (reaccionan de una forma determinada ante un mismo evento (p. ej. un evento de error o de cancelación). En estos casos utilizaremos DEs anidados.

En cualquier caso, en DE anidado va a contener estados que se descomponen en un DE completo. Cualquier transición de entrada en dicho estado nos lleva al estado inicial del DE anidado. Los estados finales del DE anidado nos llevan a la transición de salida correspondiente del estado anidado.

Acciones dentro de los estados.

La inclusión de acciones de duración instantánea dentro de los estados amplía el modelo de automátas, dándole mayor expresividad y evitando duplicaciones innecesarias:

- ***Acciones de entrada***. *Si una determinada acción ha de realizarse en todas las transiciones de entrada de un estado determinado, podemos ponerla como acción de entrada de dicho estado, evitando repetirla en cada transición.*
- ***Acciones de salida***. *Igualmente, si una determinada acción ha de realizarse en todas las transiciones de salida de un estado, la pondremos como acción de salida del mismo, evitando repetirla en cada transición.*
- ***Eventos internos***. *Si, mientras el sistema está en un estado determinado, puede producirse un evento que lleve asociada una determinada acción, pero que no provoque un cambio de estado, podemos representar esto como un evento interno del estado. En este caso no se realizarán las acciones de entrada y de salida del estado, puesto que se considera que el sistema no cambia de estado al atender dicho evento.*

5.2.6.- Redes de Petri

Las redes de Petri fueron creadas por Carl Adam Petri en 1963. Es una técnica muy apropiada para la descripción del control en sistemas de comportamiento asíncrono y Concurrente.

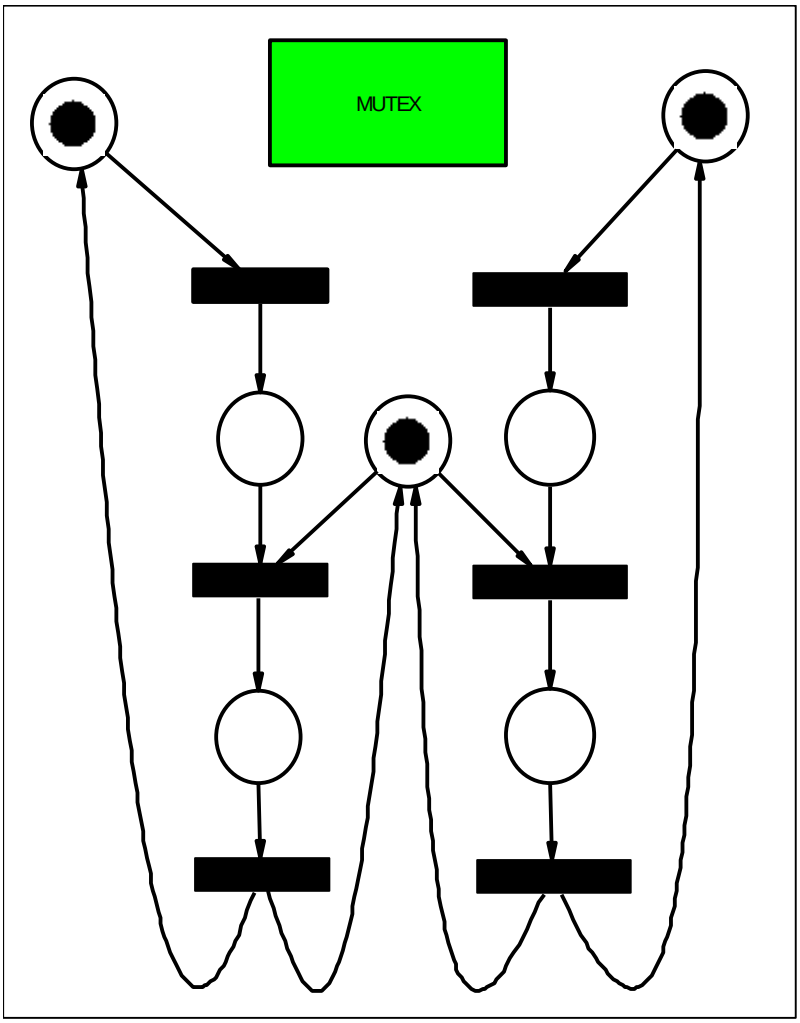

Figura 5.7. Representación gráfica de una red de Petri

Una red de Petri es un modelo compuesto por los siguientes elementos:

- Un conjunto finito de lugares representados por círculos.
- Un conjunto finito de transiciones, representados por segmentos.
- Un conjunto finito de conexiones o arcos de un lugar con una transición o viceversa, representadas por flechas.

Las redes de Petri adquieren un estado en función de su marcado, esto es, de la asignación de un número no negativo de marcas (también llamadas tokens) a cada lugar de la red, Las marcas se representan gráficamente mediante pequeños círculos negros (véase figura 5.7). Se puede considerar que cada marcado diferente representa un estado distinto. Evidentemente, toda red debe partir de un marcado o estado inicial a partir del cual evoluciona. La evolución del estado de la red se produce a través de la llamada regla de disparo de transiciones que consiste en lo siguiente:

- Cada transición consta de lugares de entrada (aquellos desde los cuales llega una flecha a la transición) y lugares de salida (a los que llega una flecha desde la transición). Una transición está habilitada cuando existe, al menos, una marca en cada uno de sus lugares de entrada.
- Una transición habilitada puede dispararse. Si se dispara, se consume (se elimina) una marca de cada lugar de entrada y se produce (se añade) una marca en cada lugar de salida.

Las redes de Petri son modelos muy genéricos que admiten muchísimas variantes. Las principales son las siguientes:

- Redes con valores en los arcos, es decir, que en cada disparo provocan el consumo de más de una marca.
- Redes que identifican las marcas, bien creando distintos tipos a los que se asignan colores (redes coloreadas) o bien asignando valores concretos a las marcas e incluyendo en condiciones relativas a dichos valores para el disparo de las transiciones (redes de predicados y funciones),
- Redes en las que se asignan tiempos a las transiciones o a los lugares. Estas redes se denominan genéricamente redes temporizadas y, a su vez, admiten muchas otras variantes en función de cómo se realice la asignación de tiempos.

De hecho, los diagramas de transición de estados podrían considerarse como un determinado tipo de redes de Petri. No obstante, en ambos modelos es importante resaltar la necesidad de una etapa de verificación de los mismos. Los modelos deberían tener las dos propiedades siguientes:

- **Limitación,** es decir, cualquiera que sea la evolución desde cualquier marcado inicial no sobrepasa un determinado límite finito de marcas en cada lugar. Esta propiedad es muy importante, porque el almacenamiento de las marcas implica un consumo de memoria, que no es un recurso infinito.
- **Vivacidad**, es decir, cualquiera que sea la evolución desde cualquier marcado inicial la red no se bloquea, siempre puede dispararse alguna transición.

La comprobación de estas propiedades para los posibles marcados iniciales suele ser bastante compleja.

5.2.7.- Diagramas Entidad/Relación.

El tercer punto de vista que podemos considerar para hacer un modelo del sistema software, es el **punto de vista de los datos**. La notación básica de los DFDs, que representan los datos mediante almacenes de datos, es suficiente cuando la información que fluyen entre los procesos es relativamente simple. Sin embargo, las aplicaciones de software de gestión y cada vez más las aplicaciones de ingeniería y de tiempo real, utilizan colecciones de datos complejas. En estos casos no basta con saber en detalle qué información contiene cada almacén de datos, sino qué relaciones existen entre estos almacenes. Para ello es necesario desarrollar un modelo de datos del sistema, lo que haremos mediante Diagramas Entidad/Relación (DER). Para seguir la metodología nos remitimos a los apuntes de la asignatura de Bases de Datos.

5.2.8.- Diccionario de datos[2].

El diccionario de datos o diccionario de requisitos es el lugar donde van a estar contenidas las definiciones de todos los elementos que aparecen en los distintos diagramas del sistema, y sirve por tanto para establecer la relación entre los distintos modelos del mismo (procesos, comportamiento y datos). El diccionario de datos permite establecer que el analista y el usuario entienden de la misma forma cada uno de los elementos de los diagramas.

El análisis del sistema software no va a estar completo porque hagamos una serie de diagramas DFD, DFC, DER y DE. En cada representación los objetos de datos y/o elementos de control juegan un papel importante. Por consiguiente. es necesario proporcionar un enfoque *organizado* para representar las características de cada objeto de datos y elemento de control. Esto se realiza con el diccionario de datos.

Se ha propuesto el *diccionario de datos* como gramática casi formal para describir el contenido de los objetos definidos durante el análisis estructurado. Esta importante notación de modelado ha sido definida de la siguiente forma:

Es un listado organizado de todos los elementos de datos que son pertinentes para el sistema, con definiciones precisas y rigurosas que permiten que el usuario y el analista del sistema tengan una misma compresión de las entradas, salidas, de las componentes de los almacenes y de los cálculos intermedios

Actualmente, casi siempre se implementa el diccionario de datos como parte de una «herramienta CASE de análisis y diseño estructurados», Aunque el formato del diccionario varía entre las distintas herramientas, la mayoría contiene la siguiente información:

[2] Pressman5, pag. 215

- ***Nombre****: Nombre principal descriptivo único del elemento de datos o de control, del almacén de datos, o de una entidad externa*
- ***Alias****: Otros nombres*
- ***Dónde se usa/Cómo se usa:*** *Un listado con*
  - *Origen, Destino del flujo de datos.*
  - *Procesos que usan el dato y como lo usan*
    - *Flujo de datos de entrada o salida.*
    - *Almacén de datos.*
    - *Entidad externa.*
- ***Descripción del contenido****: El contenido representado según alguna notación.*
- ***Comentarios adicionales*** *Tipo de datos (Archivo, Pantalla, Reporte, Interno entre procesos), valores implícitos, si es un flujo volumen por unidad de tiempo...*

Una vez que se introducen en el diccionario de datos un nombre y sus alias, se debe revisar la consistencia de las denominaciones. Es decir, si un equipo de análisis decide denominar un elemento de datos recién derivado como ***xyz,*** pero en el diccionario ya existe otro llamado ***xyz,*** la herramienta CASE que soporta el diccionario muestra un mensaje de alerta sobre la duplicidad de nombres. Esto mejora la consistencia del modelo de análisis y ayuda a reducir errores.

La información de «dónde se usa/cómo se usa» se registra automáticamente a partir de los modelos de flujo. Cuando se crea una entrada del diccionario, la herramienta CASE inspecciona los DFD y los DFC para determinar los procesos que usan el dato o la Información de control y cómo lo usan. Aunque esto pueda no parecer importante, realmente es una de las mayores ventajas del diccionario. Durante el análisis, hay una corriente casi continua de cambios. Para proyectos grandes, a menudo es bastante difícil determinar el impacto de un cambio. Algunas de las preguntas que se plantea el ingeniero del software son «¿dónde se usa este elemento de datos? ¿qué mas hay que cambiar si lo modificamos? ¿cuál será el impacto general del cambio?». Al poder tratar el diccionario de datos como una base de datos, el analista puede hacer preguntas basadas en «dónde se usa/cómo se usa» y obtener respuestas a peticiones similares a las anteriores. Para desarrollar una descripción de contenido se sigue la siguiente notación:

=	**Composición**:"está compuesto de"
+	**Inclusión**: "y"
[a | b | c]	**Selección**: situación disyuntiva
( )	**Opción**: elemento opcional
{}n	**Iteración**: repetición de un elemento n veces
texto	**Comentario**: aclarativo de una entrada del DD
@	**Identificador**: Marca los campos que identifican ocurrencia

La notación permite al ingeniero del software representar una composición de datos en una de las tres alternativas fundamentales que pueden ser construidas

1.- Como una secuencia de elementos de datos.

2.- Como una *selección* de entre un conjunto de elementos de datos.
3.- Como una *agrupación* repetitiva de elementos de datos, Cada entrada de elemento de datos que aparezca como parte de una secuencia, una selección o una repetición puede a su vez ser otro elemento de datos compuestos que necesite un mayor refinamiento en el diccionario.

Para ilustrar el uso del diccionario de datos especificamos el elemento de datos número de teléfono. ¿Qué es exactamente un número de teléfono? Puede ser un número local de siete dígitos, una extensión de 4 dígitos, o una secuencia de 25 dígitos para llamadas de larga distancia. El diccionario de datos nos proporciona una definición precisa de número de teléfono en cuestión. Además, indica dónde y cómo se usa este elemento de datos y cualquier información adicional que le sea relevante. La entrada del diccionario (le datos comienza de la siguiente forma:

Nombre:	Número de teléfono
Alias:	
Dónde se usa /	Proceso en el que es entrada
Cómo se usa:	Proceso en el que es salida
Descripción:	prefijo + número de acceso
	Prefijo = *secuencia de 4 dígitos comenzando en 0*
	Número de acceso = *secuencia numérica de cualquier tamaño*

Para grandes sistemas basados en computadora el diccionario de datos crece rápidamente en tamaño y en complejidad. De hecho, es extremadamente difícil mantener manualmente el diccionario. Por esta razón, se deben usar herramientas CASE.

5.2.9.- Comprobaciones a realizar sobre una especificación estructurada[3]

Una vez esté realizada la especificación estructurada, formada por el conjunto de DFD, e1 diccionario de datos y las especificaciones de proceso, es necesario revisarla. Como indica Yourdon [YOURDON, 1985], es conveniente hacer la revisión en base a cuatro aspectos; compleción, integridad, exactitud y calidad, por los que se comprueba:

- **Compleción**: Si los modelos de la especificación estructurada son completos.
- **Integridad**: Si no existen contradicciones ni inconsistencias entre los distintos modelos.
- **Exactitud**: Si los modelos cumplen los requisitos del usuario.
- **Calidad**: El estilo, la legibilidad y la facilidad de mantenimiento de los modelos producidos.

Las metodologías de desarrollo de software proponen la realización de revisiones en puntos concretos del desarrollo. Cuando el tipo de revisión es formal (por

[3] Piattini 2003

ejemplo, una inspección) es necesario establecer una lista de comprobación para guiar a los revisores en la reunión de revisión. Cuando se utilice una herramienta CASE, muchas de estas cuestiones se pueden resolver de forma automática. En la Tabla, se muestra un ejemplo de lista de comprobación en la que se formulan preguntas en base a los aspectos anteriormente citados y se señalan en negrita aquellas que no pueden ser resueltas por las herramientas.

Una lista de comprobación (en inglés "checklist") es una serie de preguntas sencillas con dos tipos de respuesta posibles. sí o no. Es un elemento clave para la detección de defectos en las inspecciones de software por la cual se guían los revisores, que revisan el producto centrándose en las preguntas de dicha lista.

	PREGUNTA	Sí	No
C	Todos los componentes tienen nombres		
C	Todos los procesos tienen números		
C	Todos los procesos primitivos tienen una especificación de proceso asociado		
C	Todos los flujos están definidos en el DD		
C	Todos los elementos de datos están definidos		
1	Hay elementos definidos en el DFD no incluidos en el DD		
1	Los almacenes de datos representados en los DFD están definidos en el DD		
1	**Los elementos de datos referenciados en las especificaciones de proceso están definidos en el DD**		
1	Los flujos de datos de entrada y salida de un proceso primitivo se corresponden con las entradas y salidas de la especificación de proceso		
1	Hay errores de balanceo		
1	Hay procesos que tienen sólo entradas o sólo salidas		
1	**Por cada proceso se cumple la regla de conservación de datos**		
1	**Hay flujos de entrada superfluos a un proceso**		
1	**Hay flujos de controlo flujos de datos como activadores de procesos**		
1	**Los procesos pueden generar los flujos de salida a partir de los de entrada más una información local al proceso**		
1	**Hay pérdida de información en los procesos**		
1	**Hay almacenes sólo con entradas o sólo con salidas**		
1	Hay conexiones incorrectas entre los elementos del DFD		
1	**Hay almacenes locales**		
1	**Es correcta la dirección de las flechas de los DFD**		
1	Existen redes desconectadas		
E	Cada requisito funcional del usuario tiene asociado uno o más procesos primitivos en los DFD		
CA	**El diagrama es claro (posición correcta de las etiquetas, existencia de cruces de línea, etc.)**		
CA	**Hay nombres de componentes con poca significación**		
CA	**Hay muchos flujos de entrada y salida (complejidad de interfaz alta) en procesos primitivos**		

5.3.- Consistencia entre modelos.

En los apartados anteriores hemos visto cómo desarrollar modelos de procesos, de comportamiento y de datos de un sistema, cada uno de estos modelos se centra en un determinado aspecto del sistema. También hemos visto cómo definir todos los elementos de estos modelos en un diccionario de datos y una serie de reglas para determinar la consistencia de los diferentes diagramas de se utilizan en cada modelo.

El conjunto de modelos tiene como misión dar una visión global del sistema, desde diversos puntos de vista. Los modelos pueden parecer muy distintos pero todos

ellos representan el mismo sistema por lo que deben ser consistentes. En este apartado vamos a ver como podemos asegurar la consistencia entre los distintos modelos. Los principales modelos sobre los que efectuaremos comprobaciones son los siguientes:

- **Dimensión de función**: en donde tenemos la especificación estructurada compuesta de los diagramas de flujo de datos, diccionario de datos y especificaciones de procesos.
- **Dimensión de información**: diagrama entidad/interrelación (DE/R), o diagrama de estructura de datos (DED).
- **Dimensión del tiempo**: lista de eventos, diagramas de flujo de control y especificaciones de control.

Conforme hemos descrito estos modelos hemos ido señalando algunas normas de consistencia que ahora recogeremos juntas:

Consistencia entre el modelo de procesos y el diccionario de datos.

Las reglas son las siguientes:

- **Cada elemento de los DFDs debe estar definido en el DD**. Los almacenes y flujos de datos tienen que estar definidos en el diccionario.
- **Cada dato elemental del DD tiene que estar incluido en algún flujo de datos**. Si no fuese así el diccionario contendrá datos elementales que no se usan en el sistema. Además, si el sistema debe memorizar alguno de estos datos, debe figurar también en un almacén de datos de los DFDs.

Una excepción a esta regla serían las variables locales de las primitivas de proceso, que sólo aparecen en las PSPECs, pero definir estas variables locales en el DD no es lo habitual.

Consistencia entre el modelo de control y el diccionario de datos.

Las reglas son exactamente las mismas: cada flujo y almacén de control de los DFC y cada señal utilizada en las condiciones o acciones del DE, debe estar definido en el diccionario. Por otro lado, cada señal de control definida en el diccionario tiene que estar incluida en un flujo de control de los DFCs o tiene que ser una variable local de la CSPEC.

Consistencia entre el modelo de datos y el diccionario de datos.

- Los objetos y almacenes de datos del DER tiene que estar definidos en el DD. Para cada uno de ellos hay que describir los datos elementales que contienen.

Consistencia entre el modelo de procesos y el modelo de datos.

El DER muestra el sistema desde un punto de vista muy distinto al de los DFDs, sin embargo hay una serie de reglas que deben cumplirse para que ambos sean consistentes:

- **Cada almacén de los DFD debe corresponderse con un objeto o relación del DER**. Ambos elementos representan necesidades de almacenamiento de información del sistema.
- **Los nombres de los elementos del DER y de los almacenes de los DFDs tienen que corresponderse**. El convenio que seguiremos es usar nombres en plural para los almacenes de datos (p.ej. Clientes) y en singular para los objetos (p.ej. Cliente).
- Las entradas del diccionario deben aplicarse tanto a los almacenes de datos como a los objetos y relaciones correspondientes.

Consistencia entre el modelo de procesos y el modelo de control.

- **Cada DFC y cada CSPEC están asociados a un DFD**. (El recíproco no es necesariamente cierto). Los procesos, entidades externas y almacenes del DFD deben aparecer en el DFC correspondiente.
- **Cada estado del DTE se asocia a un proceso o grupo de procesos del DFD, activos durante ese estado, o al sistema esperando que se produzca algún evento**. De acuerdo a esto, los nombres de estados y procesos deben estar relacionados: para los procesos utilizaremos infinitivos (p.ej. Comprobar Saldo Cliente) y para los estados, gerundios (p.ej. Comprobando Saldo Cliente).

A estas reglas cabe añadir:

Plano información-función

- Comprobar que todos los elementos (o datos elementales) definidos en los diagramas entidad/interrelacién están definidos como entradas en el DD, es decir, están en algún flujo de datos o almacén. No tiene por qué ocurrir lo contrario (todos los elementos del DFD en el diagrama E/R), ya que en el diagrama E/R no se representan datos secundarios.
- Realizar la misma comprobación con los diagramas de estructuras de datos. En este caso, éstos pueden contener datos secundarios a efectos de rendimiento del modelo.
- Comprobar que cada entidad e interrelación del DE/R es consultada y actualizada al menos una vez por alguna función primitiva del DFD (lo que implica que habrá que comprobarlo en las especificaciones de proceso).

Plano información-tiempo

- Comprobar que cada entidad del DED tiene asociado un diagrama de historia de la vida de la entidad (HVE). Si se hace con el DE/R además de las entidades habrá que comprobarlo con las interrelaciones M:N.
- Comprobar que por cada entidad existe un evento que la crea. Hay que preguntarse el porqué en caso de no haber eventos que la actualicen o eliminen. Para esto, se construye la matriz evento-entidad.

- Comprobar que en las HVE de las entidades «maestro» 26 se tratan las posibles repercusiones que tiene el borrado de dicha entidad sobre las entidades «detalle».

Plano tiempo-función

- Comprobar que existe un proceso primitivo dentro de los DFD que trate cada uno de los eventos identificados en la HVE.

5.3.1.- Técnicas matriciales

Las técnicas matriciales se utilizan principalmente para ayudar a verificar la consistencia entre los componentes de distintos modelos de un sistema, ya sean centrados en las funciones, en la información, o en el aspecto temporal.

	FUNCIÓN	**INFORMACIÓN**	**TIEMPO**
FUNCIÓN			
INFORMACIÓN	Matriz Entidad/Función	Matriz Entidad/Entidad	
TIEMPO		Matriz Evento/Entidad	

Hay que resaltar que todas las técnicas matriciales no tienen que encuadrarse en esta tabla. Así, la matriz de papeles de usuario/función utilizada por la metodología SSADM relaciona los usuarios que van a utilizar el sistema con determinadas funciones del mismo. Cada celda de la matriz indica un diálogo, es decir, un conjunto de pantallas mediante las cuales el usuario se comunica con una función del sistema. En SSADM es necesario realizar este estudio antes de comenzar a diseñar la interfaz del usuario mediante la técnica de diseño del diálogo.

Matriz entidad/función

Esta matriz visualiza las relaciones existentes entre las funciones que lleva cabo un sistema y la información necesaria para soportar las mismas.

Los elementos de las filas pueden ser *entidades, interrelaciones, entidades asociativas* y *subtipos,* presentes en un diagrama entidad/interrelación. Los elementos de las columnas están formados por las funciones del sistema, que pueden ser funciones de alto nivel representadas en un DFD o bien las funciones primitivas del conjunto de DFD.

En cada celda se incluyen las posibles acciones que puede realizar una función sobre las ocurrencias de las entidades, interrelaciones, etc. Estas acciones pueden ser crear o insertar (I), leer (L), modificar o actualizar (M) y borrar (B).

Funciones / Entidades	Gestionar presupuesto cliente	Gestionar Cliente	**...**
CLIENTE	**L**	**I, M, B**	
PRESUPUESTO	**I, M, B**		
...			

Matriz entidad/entidad

Es una matriz que ayuda a ver las relaciones que tiene una entidad con otras del modelo entidad/interrelación. Es especialmente útil cuando el número de entidades es grande. En el caso de que exista una interrelación entre dos entidades se incluye en la matriz una X o el nombre de la interrelación.

Entidad / Entidad	CLIENTE	PRESUPUESTO	**...**
CLIENTE		**Tiene**	
PRESUPUESTO			
...			

Matriz evento/entidad

En las filas se incluyen todos los eventos, entendiendo como evento un suceso que ocurre en la vida del sistema que supone una alteración de los datos almacenados. En cada columna se incluyen las entidades del diagrama de estructura de datos (véase apartado 75.3). En cada celda se define la alteración causada por el evento, que puede ser la inserción o creación (1), modificación o actualización (M) y horrado (B) de una ocurrencia de la entidad. Hay que comprobar que cada entidad tiene algún evento que la crea, modifica o borra. Si no es así (lo que no necesariamente indica un error), es conveniente preguntarse el porqué. Por el contrario, es necesario que exista por cada entidad un evento que la crea. Además, es necesario asegurarse de que cada evento afecta a la vida de al menos una entidad. Si alguna fila o columna no tiene marca, se habrá cometido un error grave.

Entidades / Eventos	CLIENTE	PRESUPUESTO	**...**
Datos del Cliente	**I, M, B**		
Datos del presupuesto	**I**	**I, M, B**	
...			

5.4.- Metodología del análisis estructurado.

En el apartado anterior hemos visto diversas notaciones para el análisis estructurado de sistemas. Estas notaciones nos permiten modelar el software desde diversos puntos de vista. Vamos a ver ahora como podemos coordinar estos modelos y cuál es el método de trabajo apropiado.

5.4.1.- Fases.

Creación del modelo de procesos.

El punto de partida será el modelo de procesos del sistema. Mediante el uso de DFDs y PSPECs podemos modelar el ámbito de información y ámbito funcional del sistema.

La tarea de análisis consiste básicamente en eso: en analizar o pensar, no en ponerse a dibujar diagramas a lo loco. Por esto, lo primero que tenemos que hacer es estudiar toda la documentación inicial que tengamos del sistema, bien sea una especificación preliminar, o el resultado de las reuniones o entrevistas preliminares con el usuario.

Podemos hacer un análisis gramatical de la información de entrada. Identificando los nombres y los verbos de la especificación preliminar podremos identificar muchos de los componentes del sistema. Podemos clasificar estos elementos en entidades externas (nombres), flujos y almacenes de datos (nombres) y procesos o funciones (verbos). La especificación relaciona estos nombres y verbos entre sí. Esto nos permitirá establecer la relación entre procesos y el resto de los componentes. Podemos hacer listas separadas de cada uno de estos componentes. No serán correctas ni completas, pero es un buen punto de partida.

A continuación, comenzaremos por hacer el DFD de contexto, mostrando el sistema software en relación con las entidades externas con las que interactúa. En este nivel representamos el sistema mediante un sólo símbolo de proceso, no se suelen mostrar almacenes de datos, y se identifican las entradas y salidas principales del sistema (no se detallan todos los flujos de datos, sino que se los agrupa).

A continuación, vamos refinando el DFD de contexto en mayores niveles de detalle. Este es un proceso de descomposición funcional, en el que tenemos que ir descomponiendo el sistema en las distintas funciones que realiza. Hay que aplicar los principios de **acoplamiento mínimo** (máxima independencia) entre procesos distintos y **máxima cohesión** (fuerte conexión funcional entre todo lo que está dentro de un proceso) en cada proceso. Si todos los flujos de datos que aparecen en un DFD son utilizados en todos los procesos del mismo, posiblemente algo irá mal. Según esto, un proceso debe realizar una función clara y coherente.

El número de procesos que aparecen en un DFD debe estar (idealmente) entre 5 y 10. Poner menos llevará a tener que hacer demasiados DFDs y a que alguna de las burbujas no sea más que un saco de procesos (es decir, habrá poca cohesión). Poner más dificultará el conseguir una visión general del diagrama.

Según vamos descomponiendo, tenemos que tener cuidado en mantener la consistencia: consistencia de nombres, de numeración y de flujos de datos, aplicando la regla de balanceo.

No hay que caer en detalles de implementación, el DFD pretende dar una visión de cómo se mueve idealmente la información entre los diferentes procesos, y no en ser un lenguaje de programación gráfico.

Según vamos incluyendo elementos en los DFDs hay que ir definiéndolos en el diccionario de datos del proyecto.

Seguiremos el proceso de descomposición hasta encontrar las primitivas de proceso: estas serán procesos sencillos, con máxima cohesión (realizan una única función). Los flujos de datos que entran y salen de las primitivas constituyen la interfaz de estos procesos. Describiremos estas primitivas mediante PSPECs, a ser posible en pseudocódigo, manteniendo la consistencia de flujos de entrada y salida con respecto a los DFDs.

Las PSPECs no contienen definiciones de datos, por tanto, no podemos ocultar en ellas almacenes de datos. Estos tienen que aparecer en los DFDs.

Creación del modelo de control.

No todos los sistemas necesitan de un modelo de control. En muchos casos, el comportamiento implícito en los DFDs servirá para mostrar cómo se comporta el sistema. Debido a esto, lo primero es determinar si es necesario o no desarrollar un modelo de control.

Aquí hay que tener especial cuidado en no caer en detalles de implementación, y empezar a pensar en qué programas llaman a qué programas o como se sincronizan los procesos. Esta es una tarea más propia del diseño.

Si hemos decidido desarrollar un modelo de comportamiento del sistema, debemos empezar por establecer una jerarquía de DFCs simétrica a la de DFDs. Cada par DFD/DFC contiene los mismos procesos, almacenes de datos y entidades externas, y en la misma posición, para facilitar la identificación.

A continuación, eliminaremos del DFD todos los flujos que transporten información de control: aquélla que sirve para determinar el comportamiento del sistema, activando o desactivando procesos. Estos flujos de control los representaremos en los DFCs.

La jerarquía de DFCs será normalmente más corta que la de DFDs. Continuaremos desarrollando DFCs mientras los procesos que aparecen en ellos generen y sean controlados por los flujos de control que vayamos especificando. En aquellos casos en los que el comportamiento del sistema quede reflejado implícitamente en los DFDs no haremos DFCs.

A cada par DFD/DFC le corresponde una CSPEC. Las condiciones de datos generadas por los procesos y las señales de control provenientes del exterior del DFC entrarán en las ventanas de control. De estas ventanas saldrán los activadores de procesos (se pueden representar o no, opcionalmente) y las señales de control que salgan del DFC.

Hay que mantener la consistencia de flujos de control entre las ventanas de control y las CSPECs.

La CSPEC pude ser combinacional, secuencial o compuesta. Elegiremos siempre el modelo más sencillo posible. Las CSPECs combinacionales pueden representarse mediante tablas de decisión y tablas de activación de procesos. Las secuenciales mediante DEs.

Respecto a los DEs, hay que tener cuidado con las posibles omisiones de transiciones. ¿Existe alguna otra forma de llegar a un estado o salir de él?, ¿Están previstas todas las contingencias que pueden presentarse en cada estado?

Creación del modelo de datos.

Si los datos que maneja el sistema tienen una estructura compleja, no bastará con definir en el diccionario el contenido de cada uno de los almacenes de datos. Desarrollaremos un modelo de datos, utilizando DERs, donde se muestren las relaciones que existen entre los datos que maneja el sistema. Como punto de partida de este modelo utilizaremos los almacenes de datos y entidades externas que aparecen en el modelo de procesos.

Consistencia entre modelos.

Una vez se han realizado los distintos modelos del sistema se deberían llevar a cabo las técnicas de consistencia entre modelos discutidas en el apartado anterior.

5.5.- Modelos del sistema: esencial y de implementación.

El modelo esencial del sistema (algunas veces llamado modelo lógico del sistema) es un modelo de lo que el sistema debe hacer con objeto de satisfacer los requisitos del usuario. En el modelo esencial no figura (al menos idealmente) nada acerca de cómo se va a implementar el sistema. Es por tanto, un modelo abstracto del sistema, que supone que disponemos de una tecnología perfecta sin coste alguno.

Este modelo esencial es el resultado de la fase de análisis de requisitos del sistema, y de este tipo son todos los modelos que hemos estado viendo hasta ahora.

El modelo esencial tiene que estar completamente libre de detalles de implementación. Alguno de los errores típicos en los que se cae al hacerlo son:

- **Secuenciar de forma arbitraria los procesos del DFD**. Los procesos de los DFDs deben ser lo más concurrentes posible, no hay que pensar en que el sistema realiza una tarea, y luego otra, y luego otra. El único secuenciamiento de funciones que debe aparecer en los DFDs es el impuesto por la dependencia de datos. Si un proceso P2 recibe como entrada un flujo de datos generado por un proceso P1, no podrá realizarse hasta que P2 acabe. Cualquier otra secuencia es puramente arbitraria, y dependerá más de necesidades de implementación (p.ej. capacidad del ordenador) que de requisitos del sistema.
- **Utilizar ficheros temporales o de backup**. Los ficheros temporales se usan para conectar procesos que no pueden ejecutarse simultáneamente por problemas de capacidad o multiproceso del ordenador, no porque los procesos deban ejecutarse de forma secuencial. Los ficheros de backup se utilizan para evitar perder información en caso de que falle el sistema (sea el fallo hardware, software o humano). Supuesta una tecnología perfecta,

no necesitamos de ninguno de ellos. No son requisitos esenciales del sistema sino de implementación.

- **Utilizar información redundante o derivada**. El incorporar en las bases de datos o en los procesos información redundante tiene más que ver con la eficiencia de la implementación que con los requisitos del modelo. (P.ej. los totales de factura se pueden calcular a partir de las líneas, sin embargo, es normal almacenarlos para evitar tener que calcularlos cada vez que se accede a la factura). En el modelo esencial no usaremos nunca información redundante o derivada.

A partir del modelo esencial, los diseñadores podrían decidir cómo implementar el sistema, usando la tecnología disponible. Podrían decidir cuál va a ser el hardware, la base de datos, el lenguaje de programación, etc. y cómo implementar con estas herramientas los elementos del modelo esencial.

Sin embargo, lo normal es que el cliente proporcione más información que los simples requisitos del sistema, y que decida también sobre detalles de implementación: qué funciones van a ser manuales y cuáles automáticas, el formato de los informes y de las pantallas, requisitos operacionales de velocidad de cálculo o capacidad, etc. Toda esta información que el cliente proporciona al analista, que no se refiere a los requisitos esenciales del sistema, sino a los requisitos de implementación, formarán parte del modelo de implementación del sistema.

El desarrollo de un modelo de implementación es una tarea que está a caballo entre el análisis y el diseño. No puede ser desarrollado sólo por el analista y el cliente pues se necesita el consejo de diseñadores e implementadores sobre las elecciones de hardware y software y sobre la posibilidad de cumplir las restricciones de tiempos de respuesta y capacidades del sistema usando la tecnología elegida. Por otra parte, no puede ser realizado únicamente por diseñadores e implementadores porque el usuario debe definir una gran cantidad de requisitos de implementación, que irán surgiendo en la fase de análisis, y es al analista al que se los describe. (El analista es el principal vínculo entre el cliente y el equipo de desarrollo. El cliente tiene muy poco trato con diseñadores e implementadores).

Este modelo de implementación será una versión revisada y anotada del modelo esencial, donde se especifiquen todos estos detalles adicionales proporcionados por el usuario. Entre estas anotaciones podemos citar:

- **La elección de dispositivos de entrada y salida**. Desde el punto de vista del modelo esencial, el sistema recibe y emite flujos de datos a las entidades externas. Un modelo de implementación debe indicar qué dispositivos se utilizan para estas entradas o salidas: interruptores y señales luminosas o estaciones de trabajo, p.ej.
- **La elección de los dispositivos de almacenamiento**. Discos duros, discos ópticos, cintas magnéticas, etc.
- **El formato de las entradas y salidas**. Incluyendo aquí el tamaño de los datos (p.ej. longitud de los nombres, formato de fechas y números de teléfono, número de líneas máximo en un pedido, etc.). Todas estas restricciones deben ser incorporadas al diccionario de datos.

- **La secuencia de operaciones de entrada y salida.** Incluyendo la definición de cómo se van a hacer los diálogos del sistema con el usuario (orden de las pantallas de captura de datos y disposición de los datos en la pantalla, valores por defecto de los datos, mensajes de error y de ayuda, cómo se pasa de una pantalla a otra, etc.) Muchos de estos aspectos pueden definirse mediante DEs.
- **Volumen de datos**. El usuario debe indicar el volumen de datos que manejará el sistema de forma que se puedan prever las necesidades de almacenamiento y procesamiento.
- **Tiempo de respuesta**. El tiempo de respuesta de las aplicaciones interactivas y el tiempo máximo de procesamiento que puede consumir in proceso batch, etc.
- **Copias de seguridad y descarga de datos del sistema**. El usuario puede indicar cuándo y de qué datos es necesario hacer copia de seguridad o qué datos necesita mantener on-line (información del último año p.ej.) y qué datos se pueden descargar del sistema a cinta magnética. Habrá que prever mecanismos de recuperación o recarga de esa información cuando sea necesaria.
- **Seguridad**. Mecanismos de seguridad para evitar accesos no autorizados a todos o parte de los datos, a determinados procesos, etc.

5.6.- Ejemplos.

5.6.1.- Traductor y Distribuidor de Comunicaciones

Trataremos de modelizar un Traductor y Distribuidor de Comunicaciones (TDC) que sirva de interfaz entre un visualizador estándar y una serie de "suministradores de datos" a través de la red. La idea fundamental que subyace en este gestor es que los visualizadores de datos sean independientes del suministrador, de forma que si cambia el agente suministrador el desarrollo del visualizador siga siendo válido. Para ello suponemos que hay un protocolo perfectamente definido que describe los datos entre el visualizador y el gestor de comunicaciones, representación R1, y que el gestor de comunicaciones conoce también el protocolo de comunicación con el suministrador, representación R2,. La conversión de protocolos se encuentra almacenada en una base de datos, que denominaremos BD_POE. Para concretar más estas ideas pensemos en un sistema de visualización de datos de una sala de calderas de calefacción procedentes de un edificio inteligente (Figura 5.7).

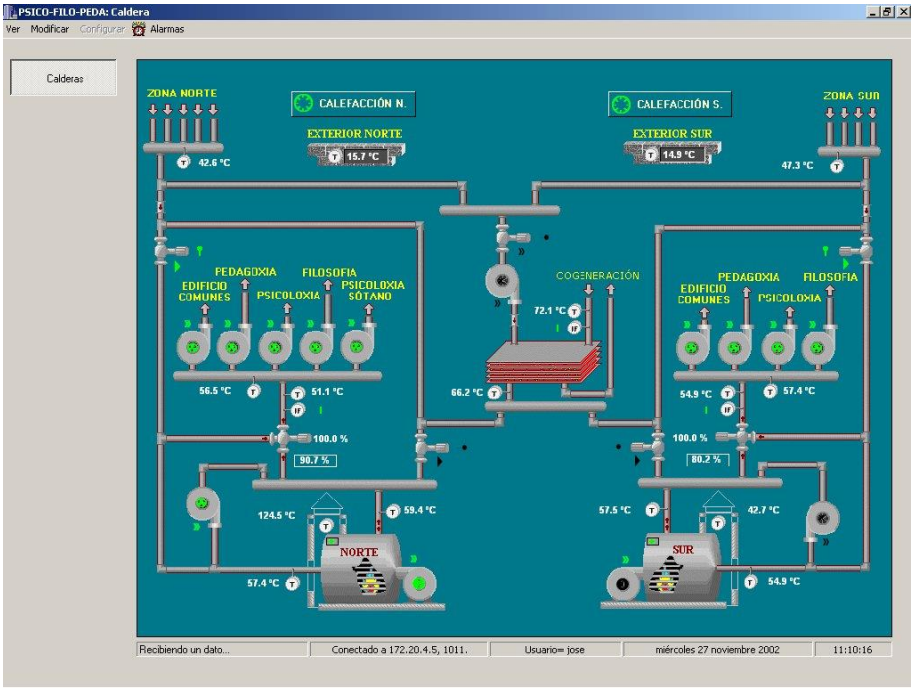

Figura 5.7.- Detalle del gráfico que representa a la sala de calderas

El suministrador es un ordenador que controla, supervisa y almacena todos los datos del edificio (datos de calefacción, agua y electricidad por ejemplo). Llamémosle NCRS, y su apariencia real es la podemos ver en la figura 5.8.

Figura 5.8.- Detalle del NCRS

Además el TDC está dotado de una ventana de monitorización que sirve, en un principio para que el desarrollador pueda monitorizar todo el tráfico de comunicaciones, tanto entre el TDC y los NCRS, como entre el TDC y los visualizadores. Esta ventana tendría el aspecto de la figura 5.9.

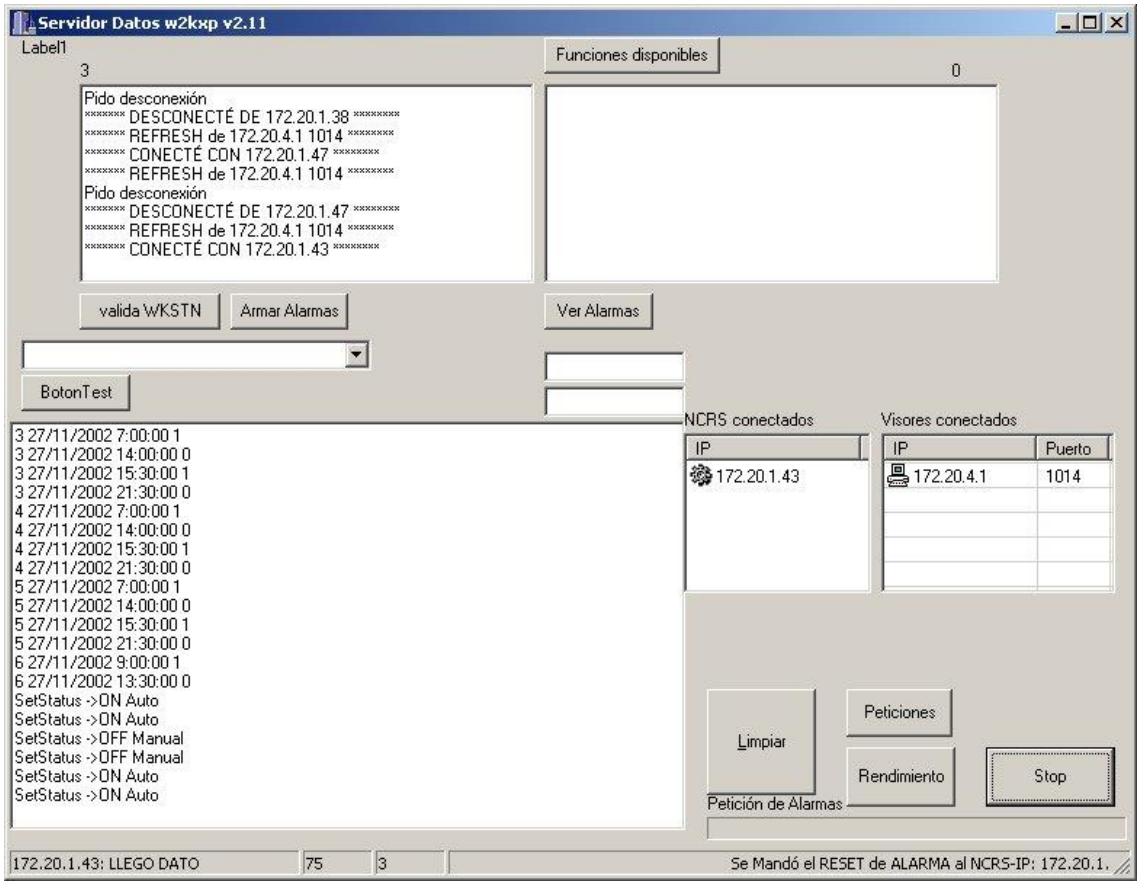

Figura 5.9.- Detalle del *Display* de monitorización

Estos tres elementos que acabamos de describir son para nuestro sistema agentes externos que consumen o producen datos. En el diagrama de contexto de nuestro gestor de comunicaciones, figura 5.10, situaremos al NCRS a la izda. y a la derecha indicando que es tanto productor como consumidor de datos.

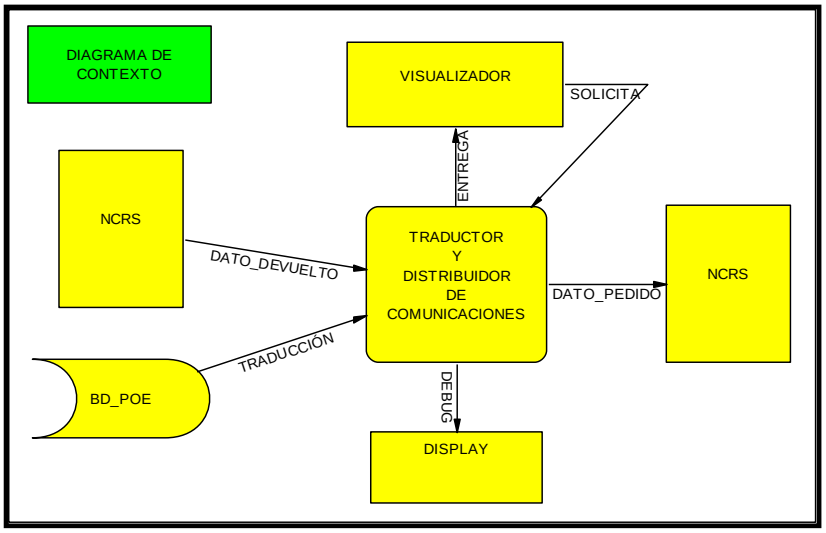

Fig. 5.10.- Diagrama de contexto del Traductor y Distribuidor de Comunicaciones

La BD_POE es un productor de datos, por lo que estará situada a la izda. El visualizador constituye una interfaz de usuario, mientras que el display, lo podemos entender como un interfaz de mantenimiento.

El DFD de nivel 1 describe los módulos principales del TDC, y su interacción con los agentes externos propuestos. Debemos tener presente que la comunicación con los visualizadores se realiza en la representación de nuestro sistema R1, mientras que la comunicación con los controladores de los edificios se hace en el protocolo de estos, esto es R2. En caso de tener varios suministradores de control en los edificios sería preciso replicar el módulo 3. Aunque como vimos no es habitual, en este primer nivel descomponemos el flujo de *debug*, que en el diagrama de contexto vimos como una interface con la pantalla en el debug de las comunicaciones en R1, DEBUG_CR1, y el de las comunicaciones en R2, DEBUG_CR2.

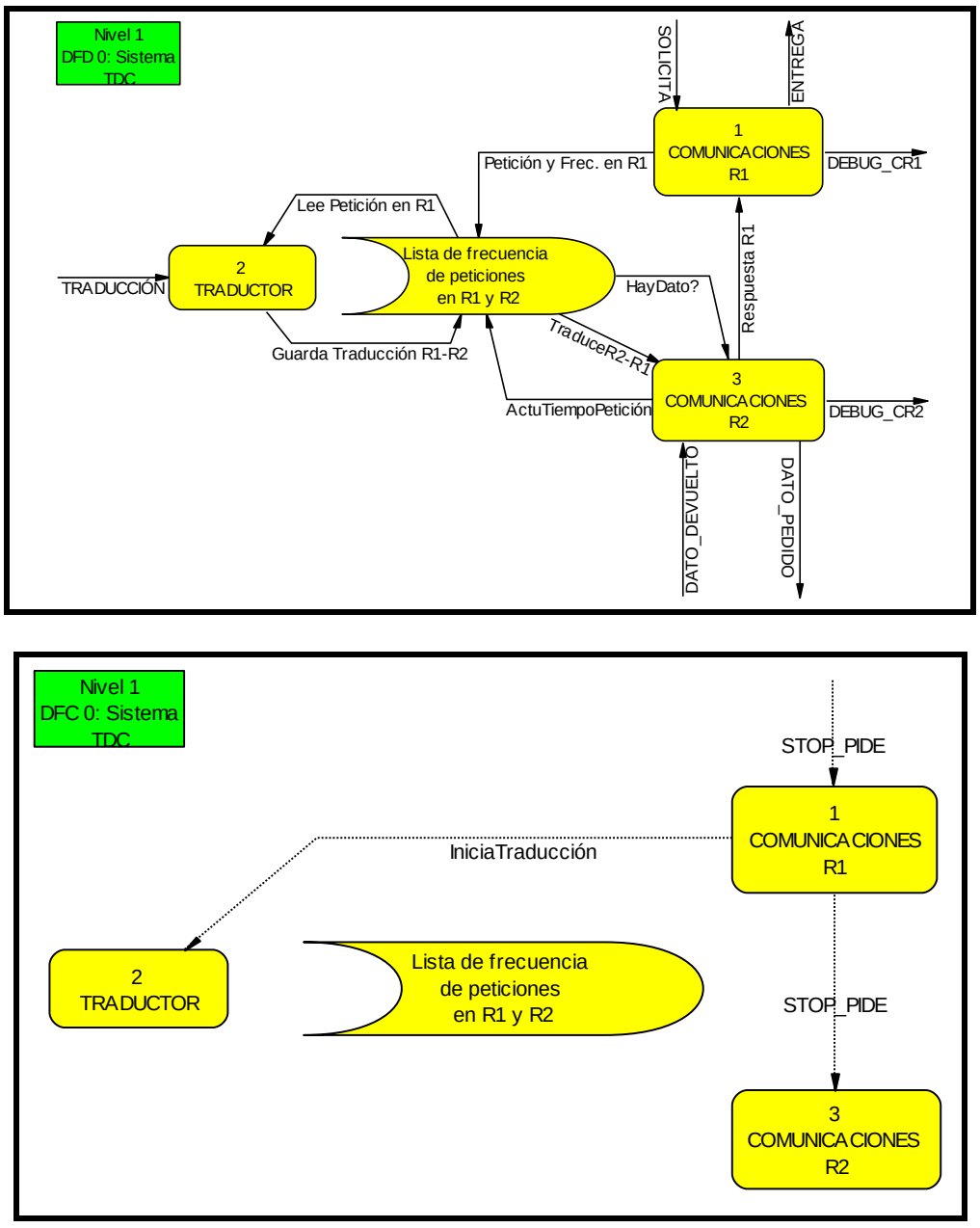

Figura 5.11.- DFD y DFC de nivel 1 del TDC

La figura 5.12 nos muestra el DFD que resulta de explotar el proceso 3. El diagrama de flujo de control (figura 5.13) nos muestra cuales son las señales y condiciones de datos que regulan el funcionamiento del TDC.

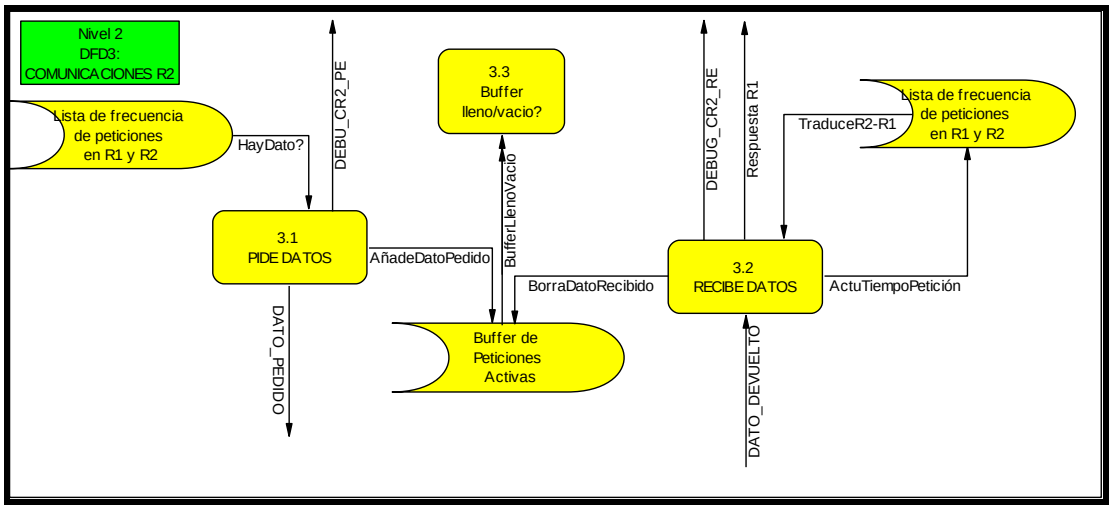

Figura 5.12.- DFD del proceso 3

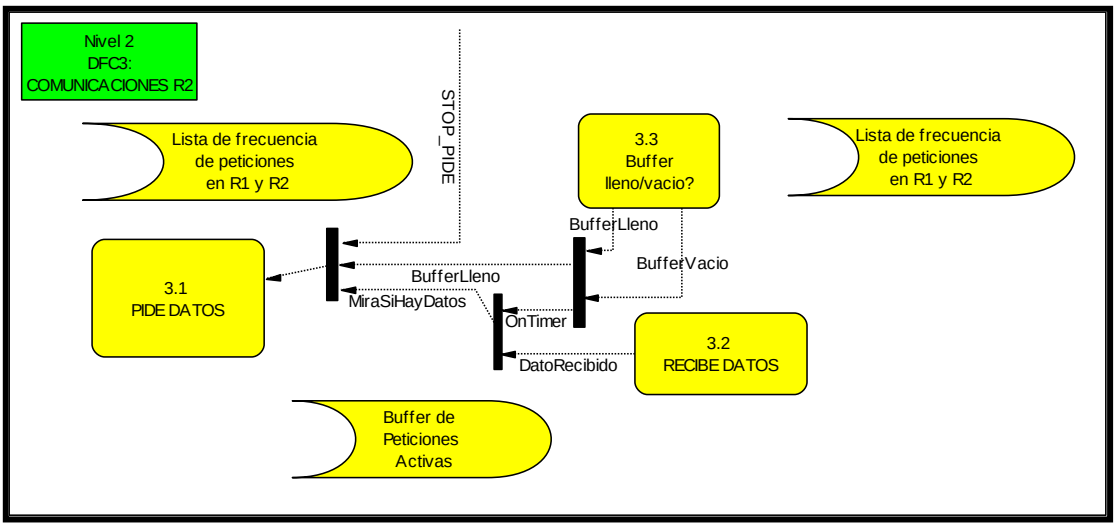

Figura 5.13.- DFC de nivel 1 del TDC

Las siguientes tablas de activación y desactivación forman parte de las Especificaciones de Control del DFC de nivel 2, junto con el Diagrama de Transición de Estados de la figura 5.14 y con la descripción natural de la figura 5.15, y nos muestran qué procesos se activan (o desactiva) bajo qué señales de control

	MiraSiHayDatos
3.1	1
3.2	0
3.3	0

Tabla de Activación

	STOP_PIDE	BufferLleno
3.1	1	1
3.2	0	0
3.3	0	0

Tabla de Desactivación

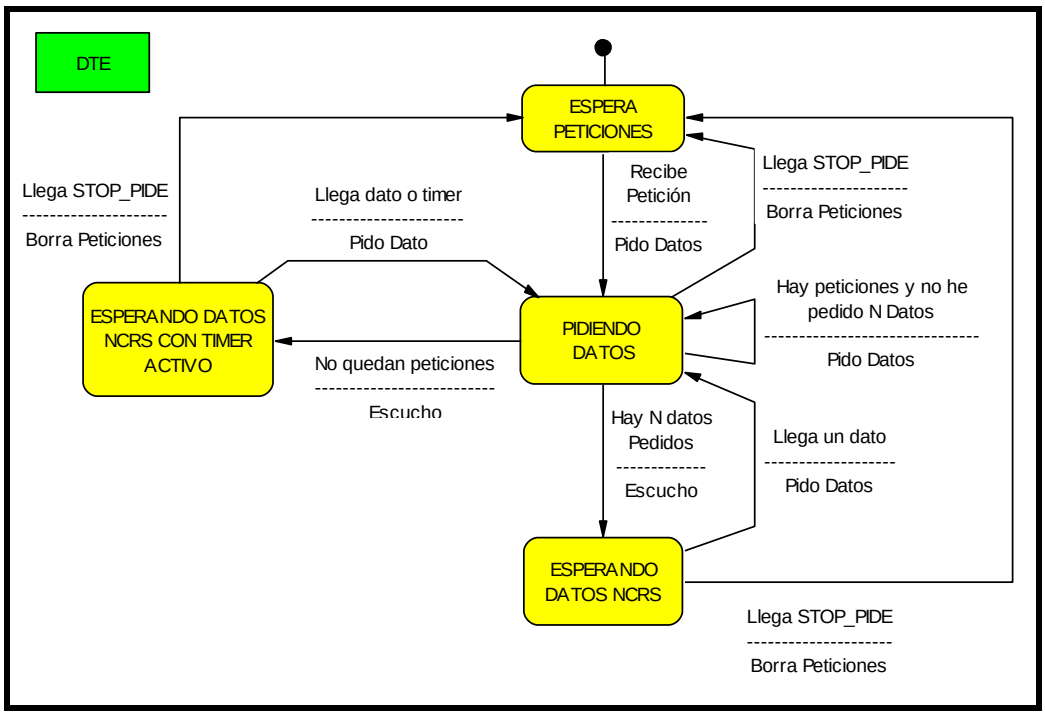

Figura 5.14. Diagrama de Transición de Estados del TDC

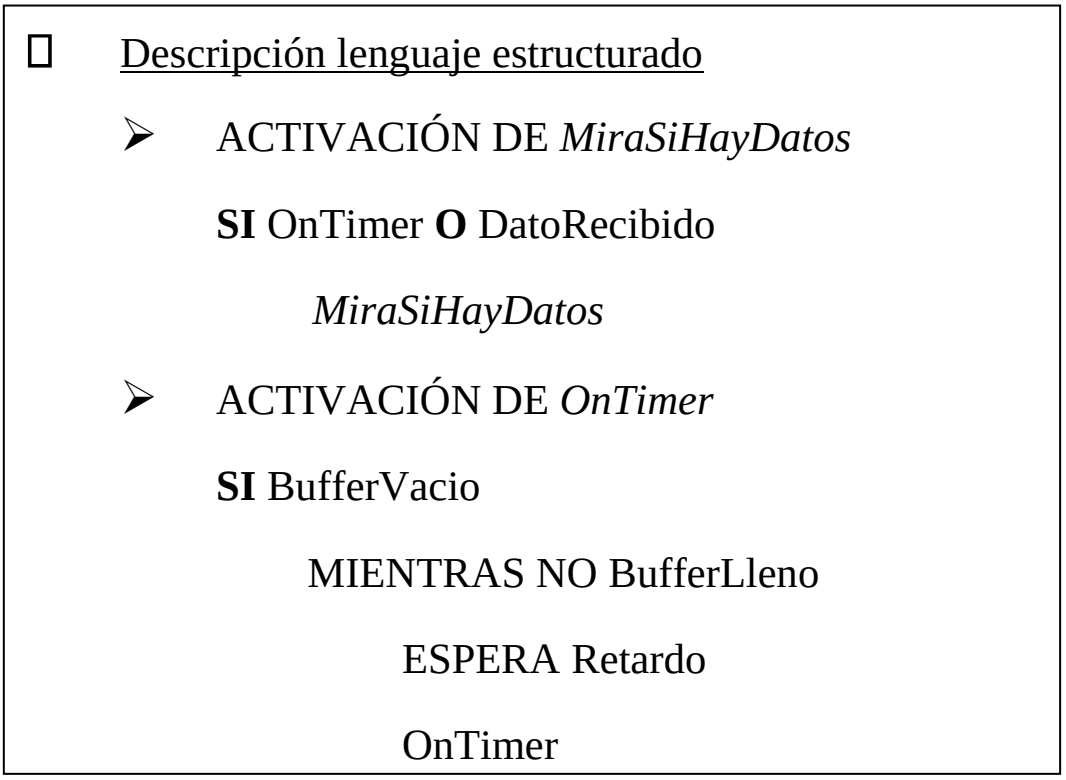

Figura 5.15.- Descripción natural de la activación de los eventos MiraSiHayDatos y OnTimer

5.6.2.- Hogar Seguro

Para describir la construcción del modelado de este ejemplo vamos a suponer que somos un equipo interdisciplinar y que vamos a seguir una técnica de obtención de requisitos de *brainstorming*, denominada TFEA (Técnicas para Facilitar las Especificaciones de una Aplicación), tal y como vimos en el tema anterior. Para obtener una lista inicial de objetos vamos a utilizar el *Análisis Gramatical*, de la narrativa de procesamiento[4] de forma que identificaremos los verbos con procesos y todos los nombres o expresiones sustantivadas, bien son entidades externas, bien flujos de datos o bien almacenes de datos.

Del análisis gramatical podemos hacer las siguientes listas:

Lista de objetos (Sustantivos):

–Panel de Control (Teclado, Display)
–Sensores (Humos, Aperturas, Movimiento)
–Alarma visual/sonora

[4] La narrativa de procesamiento del software *Hogar Seguro* la podemos encontrar en Pressman5, pag. 212.

–Línea telefónica
–Usuario
–Contraseña
–Lista de Teléfonos
–Suceso
–Retardo
–Servicio de Monitorización
–Información relevante a un suceso

Lista de Servicios/Operaciones (verbos del AG):

–Permitir (Configurar/interactuar/programar)
–Supervisar (Sensores)
–Activar/Desactivar (Sistema)
–Contactar (Servicio de monitorización)
–Detectar (Suceso)
–Invocar (Alarma)
–Especificar (Retardo)
–Proporcionar (Información al Servicio Monitorización
–Establecer (Llamada)
-Remarcar

Además, tras la supuesta reunión del equipo TFEA, obtendríamos las siguientes listas de rendimiento y restricciones:

Lista de Rendimiento:

–Respuesta inferior al segundo

Lista de Restricciones:

-Coste de fabricación < 200 €
-Interfaz usable
-Conexión a línea telefónica habitual
-Conexión a línea telefónica inalámbrica
-Funcionamiento mínimo independiente de electricidad

El diagrama de contexto que generaríamos sería el que podemos ver en la figura 5.16.

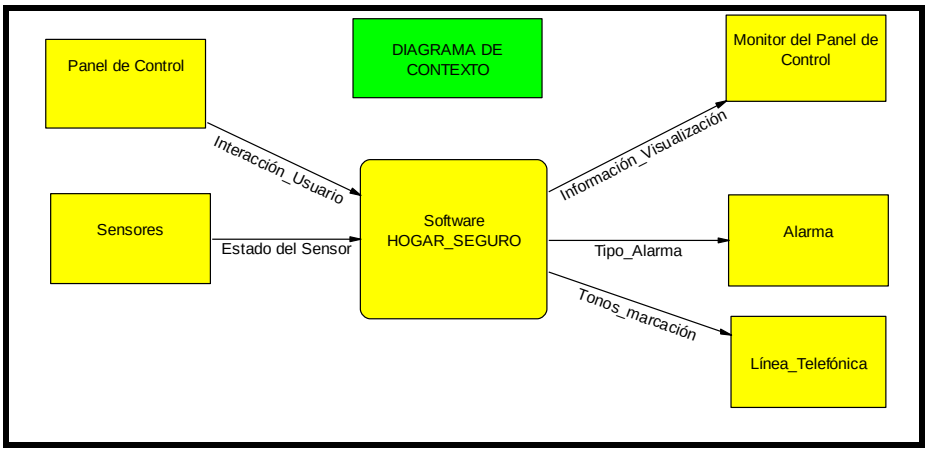

Figura 5.16.- Diagrama de contexto del software Hogar_Seguro

El diagrama de flujo de nivel 1, resultado de explotar el proceso número 0 que identifica al sistema como un todo en el diagrama de contexto, lo tenemos en la figura 5.17. De explotar el proceso 5, que hace referencia a la monitorización de los sensores y a la ejecución del procedimiento en caso de alarma, obtenemos la figura 5.18. Las siguientes (figs. 5.19 y 5.20) nos muestran el DFC y el DTE

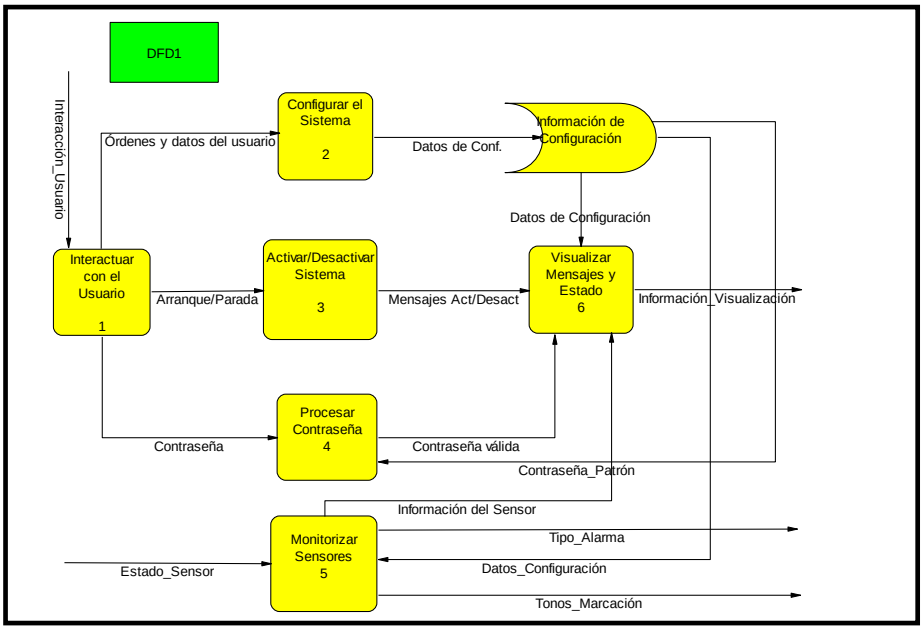

Figura 5.17.- DFD de nivel 1 del software Hogar_Seguro

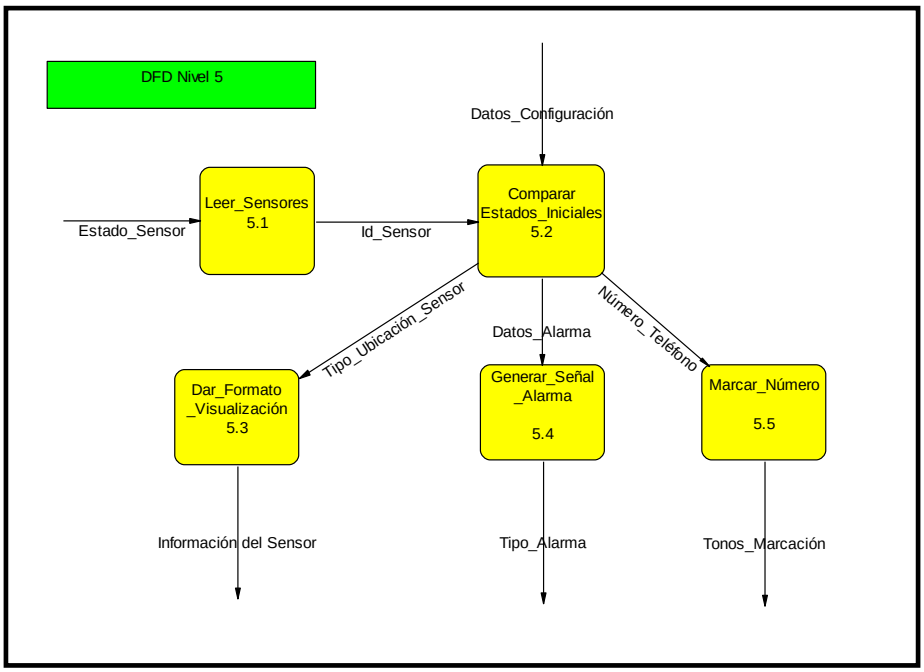

Figura 5.18.- DFD de nivel 5 del software Hogar_Seguro

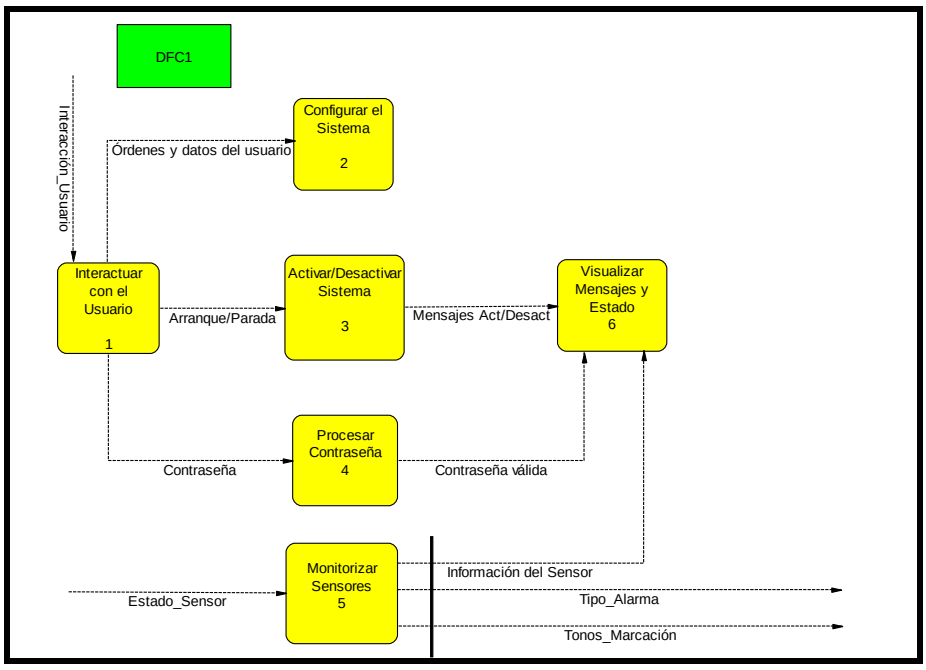

Figura 5.19.- Diagrama de flujo de control de nivel 1 del software Hogar_Seguro

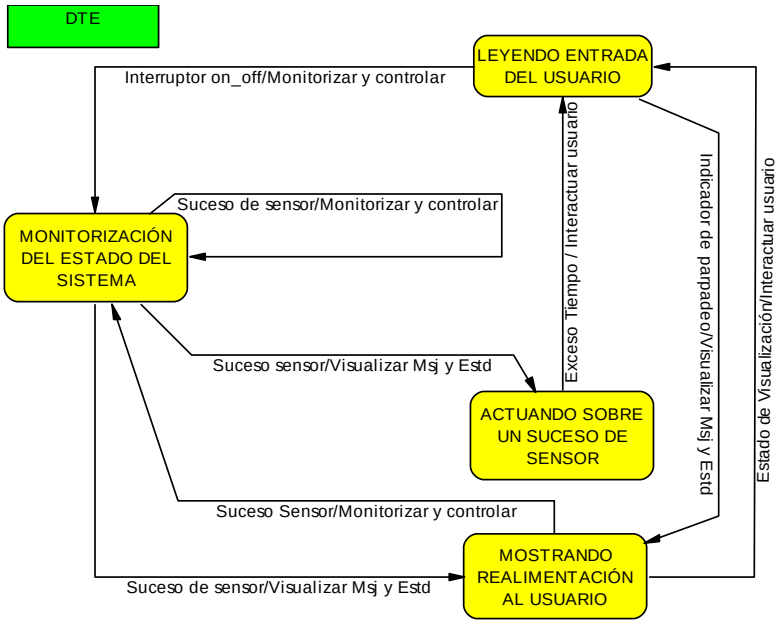

Figura 5.20.- Diagrama de estados del software Hogar_Seguro