UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

PROPUESTA DE UNA PLATAFORMA TRANSACCIONAL DE MEDIOS ALTERNOS BANCARIOS

T E S I S

QUE PARA OBTENER EL TITULO DE:

M A T E M A T I C O

PRESENTA:

JOSE ALBINO PALMA ALMENDRA

MEXICO, D. F. **1994**

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

VNIVERSIDAD NACIONAL
AVFNºMA DE
MEXICO

M. EN C. VIRGINIA ABRIN BATULE
Jefe de la División de Estudios Profesionales
Facultad de Ciencias
P r e s e n t e

Los abajo firmantes, comunicamos a Usted, que habiendo revisado el trabajo de Tesis que realiz(ó)ron EL pasante(s) JOSE ALBINO PALMA ALMENDRA

con número de cuenta 6859370-0 con el Título:

"PROPUESTA DE UNA PLATAFORMA TRANSACCIONAL DE MEDIO ALTERNOS BANCARIOS"

Otorgamos nuestro **Voto Aprobatorio** y consideramos que a la brevedad deberá presentar su Examen Profesional para obtener el título de MATEMATICO

GRADO	NOMBRE(S)	APELLIDOS COMPLETOS		FIRMA
M.en C.	JOSE	GUERRERO	GRAJEDA	
Director de Tesis				
M. en C.	ELISA	VISO	GUROVICH	
MAT.	AGUSTIN	CANO	GARCES	
DR.	FERNANDO	BRAMBILA	PAZ	
Suplente				
M. en C.	VIRGINIA	ABRIN	BATULE	
Suplente				

Al Dr. Alberto Jones
por su apoyo

Al Mat. José Guerrero
por su asesoría

PREFACIO

En una institución bancaria se van desarrollando los sistemas de información de acuerdo a las nuevas necesidades de los usuarios, a los avances en las tecnologías de hardware y software y a la competencia que existe entre las diferentes instituciones por ofrecerle un mejor servicio a los clientes. Esto ha traído como consecuencia que sistemas que tienen muchas cosas en común, se hayan desarrollado sin aprovechar esas características compartidas, trayendo como consecuencia un mayor tiempo de desarrollo, repeticiones innecesarias, equipos de personal diferentes para darles soporte y mantenimiento. De ahí que surja la necesidad de rediseñar esos sistemas para aprovechar todo lo que tienen en común y en consecuencia, mejorar el servicio.

En este trabajo se plantea una propuesta para la solución de tal problema, aunque dada su complejidad, aquí solo presentamos un posible camino de solución. Se desarrolla a partir de los sistemas con los que existe una experiencia directa, y con la asesoría de los maestros participantes en el convenio que existe entre la institución bancaria y la Facultad de Ciencias.

El trabajo se compone de tres partes. En la primera se hace un planteamiento general del problema, ubicándolo en el marco general de la empresa-sistema en estudio, siguiendo las líneas generales del Análisis de Sistemas. En la segunda parte se presentan los elementos teóricos con base en los cuales sustentar una crítica y elaborar una propuesta respecto del problema planteado; cuestiones que constituyen el núcleo principal de la tercera parte, en la que también se incluye un apéndice que contiene la descripción de un producto de software, cuya comprensión es básica para entender los sistemas que se describen en el trabajo.

INDICE GENERAL

Capítulo 1

EL PROBLEMA Y SU ENTORNO

El problema que abordaremos en este trabajo se ubica dentro de una institución de servicios bancarios. Por lo tanto, si queremos describirlo, resulta útil primero describir el entorno en el cual vive; esto es, la organización bancaria. Para estudiar a la organización nos basamos en los conceptos del enfoque de sistemas o Análisis de Sistemas. A continuación pasamos a describir a la organización bancaria en la cual vive nuestro problema, y mostramos de manera superficial los subsistemas de que se compone, para posteriormente centrarnos en el subsistema de nuestro interés.

Toda esta discusión la llevamos a cabo porque sabemos que en un sistema todas sus partes se influyen entre sí, y un cambio en una parte tiene influjo en las demás.

Después de describir brevemente el subsistema de nuestro interés, ya tenemos una base para entender el problema que queremos atacar, así que pasamos a describirlo.

1.1. El Enfoque de Sistemas en las organizaciones

El Enfoque de Sistemas nos proporciona un marco conceptual formal para el estudio de las Organizaciones.

Una Organización puede ser: una fábrica, una institución bancaria, un comercio, un hospital, un aeropuerto. En fin, un grupo de personas que organizadas, persigan un fin común.

Un miembro de una Organización debe basar sus decisiones en una imagen del resto de la misma y en la influencia que de éste recibe, así como en su propia influencia sobre el conjunto de la organización. En ausencia de un modelo adecuado, deberá recurrir a descripciones muy burdas, a menudo inadecuadas.

Las organizaciones son sistemas deliberados, cuyos elementos están esencialmente implicados en una actividad de solución de problemas. Ahora bien, por la naturaleza misma del proceso de toma de decisiones debe existir un modelo o una imagen del objeto de la decisión antes de que ésta se tome.

La "intuición y la experiencia" del tomador de decisiones será necesaria todavía, pero el "poder lógico" de los tomadores de decisiones se "ampliará".

Un modelo conceptual formal de una organización permitirá un uso más extenso de los conceptos formales y de las técnicas analíticas o cuantitativas en el estudio del comportamiento de una organización.

Los modelos generales de sistemas deben extender la capacidad humana para razonar acerca de las organizaciones, así como, por analogía, las computadoras digitales ampliaron la capacidad de cálculo y organización de la información.

En la Teoría General de los Sistemas se define una Organización como un sistema buscador de metas que tiene subsistemas interrelacionados buscadores de metas distintas ordenadas de modo jerárquico. Se conoce y acepta generalmente la existencia de una multiplicidad de metas como característica necesaria para que un sistema se identifique como organización. La teoría general de los sistemas afirma también que la existencia de una multiplicidad de subsistemas buscadores de metas ordenadas de modo jerárquico es característica suficiente para que un sistema represente una organización; es decir, la existencia de una estructura de múltiples niveles y múltiples metas es la característica primordial de una organización.

Esta manera especial de analizar a las organizaciones, es llamada "Enfoque de Sistemas".

Definiciones:

Sistema. Un sistema es un conjunto de objetos con relaciones entre los objetos y entre sus atributos.

Ambiente. Para un sistema dado, el ambiente es el conjunto de todos los objetos que quedan fuera del sistema:
1) un cambio de cuyos atributos afecta al sistema y
2) cuyos atributos cambian por el comportamiento del sistema.

Centralización. Un sistema centralizado es aquel en que un subsistema desempeña un papel principal o dominante en la operación del sistema. Podemos llamar a este subsistema la parte principal, o decir que el sistema está centrado alrededor de esta parte. Un cambio pequeño en la parte principal se reflejará entonces en todo el sistema, provocando un cambio considerable.

La centralización es progresiva si a medida que evoluciona el sistema surge una parte como entidad central y controladora.

El concepto del sistema centralizado produce el principio importante de que cuanto más se centralice un sistema más deberá protegerse a la parte principal del daño causado por factores ambientales inestables.

Sistemas abiertos y sistemas cerrados. Un sistema es abierto si hay intercambio de componentes con sus ambientes. La mayor parte de los sistemas orgánicos son abiertos.

Un sistema es cerrado si no existen estos intercambios. Un ejemplo es una reacción química que ocurre en un recipiente sellado y aislado.

El que un sistema dado sea abierto o cerrado depende de la porción del universo que se incluya en el sistema y la porción que se incluya en el ambiente. Adjuntando al sistema la parte del ambiente con la que ocurren intercambios el sistema se vuelve cerrado.

Al estudiar un sistema como abierto el interés se centra en los insumos o productos del sistema, porque es el transporte de energía a través del sistema lo que nos permite obtener trabajo de él.

Es un auxilio poderoso el análisis de los sistemas abiertos mediante el reconocimiento de sus aspectos de red. Disponemos de una considerable teoría matemática de las redes.

Sistemas adaptables. Muchos sistemas naturales, sobre todos los vivientes, muestran una cualidad que suele llamarse *adaptación.* Es decir, poseen la capacidad para reaccionar ante sus ambientes en una forma favorable, en algún sentido, para la continuación de la operación de los sistemas. Es como si los sistemas de este tipo tuviesen algún "fin" previamente asignado, y el comportamiento del sistema es tal que lo lleva a este fin a pesar de condiciones ambientales desfavorables. El "fin" podría ser la mera supervivencia; la teoría de la evolución se basa en gran medida en la noción de la adaptación al ambiente. La noción de estabilidad se relaciona estrechamente con los conceptos de adaptación, aprendizaje y evolución.

1.2 Descripción general de la organización en estudio

El problema que queremos abordar en este trabajo se ubica en el contexto de una institución de servicios bancarios, que tiene como su objetivo principal: "proporcionar servicios financieros al público en general".

Para cumplir con este objetivo, mostramos en la figura 1.1 la estructura de esta organización.

1.2.1 Algunas características de la organización en estudio

1. *Sistema abierto.* Podemos entender a la organización como un "sistema adaptable" de procesamiento de la información que interacciona con un entorno al que trata permanentemente de comprender y dominar. Es, por lo tanto, un sistema abierto a su entorno en varios sentidos. El primero y más obvio es el de base física: intercambia bienes y servicios con clientes y proveedores para obtener un excedente económico.

Para sobrevivir y tener éxito de forma duradera frente al entorno, la organización debe planificar y coordinar conscientemente sus propios movimientos. Esto implica la formulación de una estrategia, entendida como el "conjunto de decisiones en una empresa que: 1) determina,configura y revela sus metas,propósitos y objetivos; 2) genera las principales políticas y planes para alcanzar tales objetivos; y 3) define el negocio en el que la empresa quiere operar, así como la naturaleza de las contribuciones económicas y no económicas que pretende hacer a sus accionistas, empleados, clientes y comunidades sociales".

La formulación de una estrategia requiere un conocimiento profundo del entorno competitivo, así como de las capacidades que la propia organización tiene de responder y sacar ventaja de los cambios que puedan producirse en el mismo.

En la figura 1.2 el entorno de la empresa aparece representado por el anillo exterior que rodea a la organización. Dentro de él se han singularizado aquellos elementos que consideramos más importantes. La empresa está en continua interacción con ellos, lo que se indica con las flechas en doble sentido que aparecen en el gráfico.

Los competidores son las otras empresas pertenecientes al mismo sector económico. Un sector económico es el conjunto de empresas cuyos productos o servicios son altamente sustitutivos entre sí. Ejemplo de ellos serían otros bancos, casas de bolsa y los bancos extranjeros; sobre todo, este último elemento del entorno tendrá un efecto muy importante en la organización ahora con la entrada del Tratado de Libre Comercio con Canadá y Estados Unidos.

Los inversores externos son los accionistas extranjeros y nacionales. Los proveedores podrían ser entre otros, las empresas fabricantes del hardware y el software, además de las empresas que se dedican a dar servicios de desarrollo de software. Las políticas gubernamentales tienen también un gran efecto en la organización, lo cual vimos recientemente al quitarle 3 ceros al peso, en donde se tuvieron que mantener todos los sistemas para que contemplaran esto.

2. *Sistema centralizado.* También podemos considerar a esta organización como un sistema centralizado, conformado por varios subsistemas, en donde el subsistema "Informática y Nuevos Productos" ocupa un papel central en muchos aspectos, ya que es bien conocida la importancia

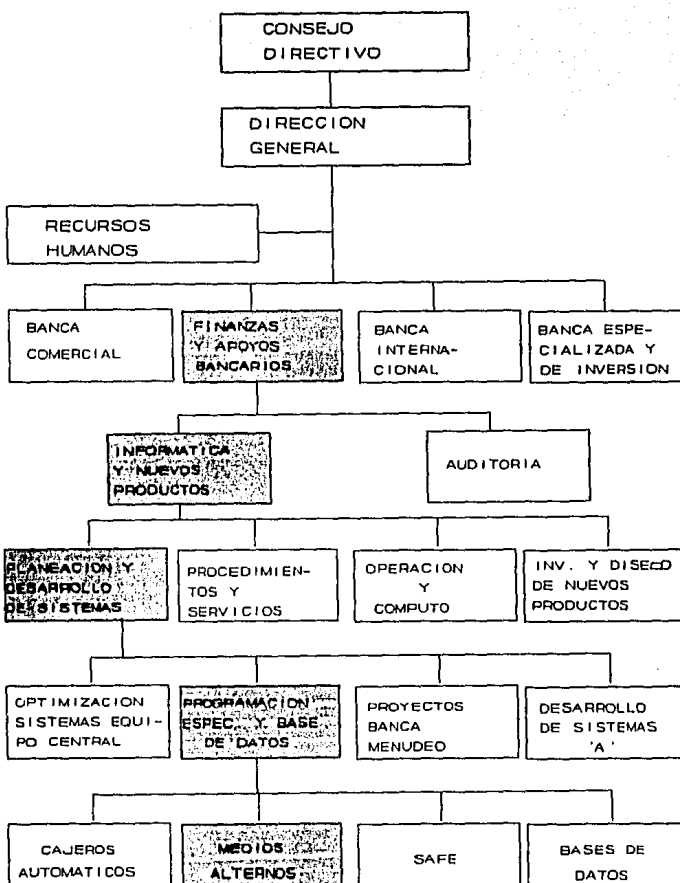


Fig. 1.1 ESTRUCTURA DE LA ORGANIZACION

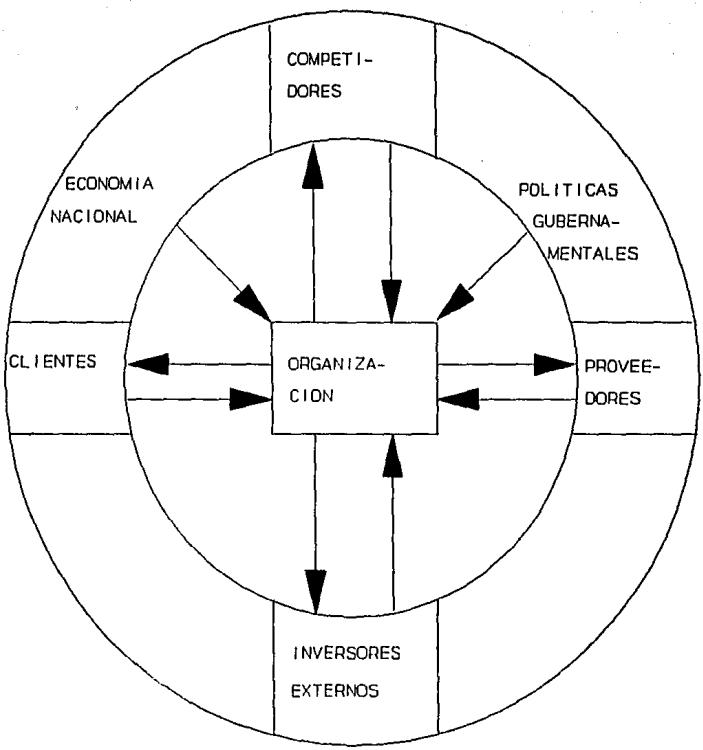


Fig. 1.2 LA ORGANIZACION Y SU ENTORNO

que tiene la computadora y los sistemas de información en una institución bancaria, y cómo su comportamiento y sus metas afectan a su vez el comportamiento y las metas de otros subsistemas dentro de la misma organización.

3. *Sistema adaptable*. Dado que una de las perspectivas de la empresa en estudio a largo plazo

es la de permanecer en el mercado, irremediablemente está obligada a adoptar cambios, que le permitan subsistir y de ser posible mejorar su posición competitiva en su ámbito de acción.

A continuación describiremos los objetivos de algunos subsistemas relevantes:

Servicios de Informática y Nuevos Productos. - Proporciona la infraestructura técnica de hardware y software que requiera la institución para cumplir con sus objetivos.

Planeación y Desarrollo de Sistemas.- Desarrolla, mantiene y da soporte a la producción de los sistemas de información de las distintas aplicaciones con que cuenta la institución para cumplir con sus objetivos.

Procedimientos y Diseño de Servicios.- tiene el mismo objetivo que el anterior pero orientado a servicios a sucursales.

Servicios de Operación y Cómputo.- tiene un objetivo similar, pero en lo que respecta a la parte técnica del hardware y el software.

Investigación y Diseño de Nuevos Productos.- Realiza investigaciones de mercado, para diseñar y desarrollar nuevos productos para introducir tecnología de punta con el fin de hacer competitiva a la institución en sus servicios de informática.

Todas las áreas que pertenecen al subsistema "Planeación y Desarrollo de Sistemas", se puede decir que tienen el mismo objetivo general que el del subsistema, esto es: *El desarrollo, mantenimiento y el soporte a la producción de los sistemas de información de las aplicaciones con que cuenta la Institución.* El atributo que básicamente diferencia a cada una de ellas es la aplicación o aplicaciones que contienen.

1.3 Descripción del subsistema de interés

Nuestro problema se ubica en el subsistema "Medios Alternos", que podemos localizar en la figura 1.1. En el capítulo 3 lo describiremos con más detalle; aquí sólo lo introducimos para poder plantear el problema que pretendemos abordar en este trabajo.

Las aplicaciones de Medios Alternos se pueden caracterizar porque están orientadas a ofrecer servicios bancarios fuera de una sucursal, en línea, y en casi todos los casos operadas por el propio cliente, a través de una terminal conectada en línea al equipo central.

En la figura 1.3 presentamos el modelo conceptual actual de los Medios Alternos Bancarios.

El cliente envía un requerimiento (E), que podría ser la transacción: traspaso de una cuenta de cheques a una cuenta de tarjeta de crédito. Este requerimiento llega a un módulo de control, que de acuerdo a un código que lleva el requerimiento, lo envía al módulo asignado para su proceso. Este módulo procesa el requerimiento, y envía una salida (S) al cliente, que es una respuesta a su requerimiento.

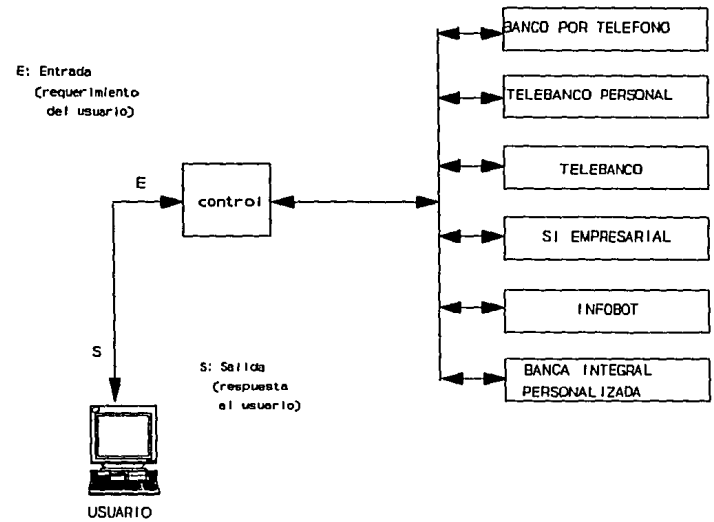


Fig. 1.3 MODELO CONCEPTUAL DEL FUNCIONAMIENTO DE MEDIOS ALTERNOS ACTUAL

En SI-EMPRESARIAL el usuario final o cliente, es una empresa.

En BANCA INTEGRAL PERSONALIZADA el usuario (o cliente) es una persona física a quien por la magnitud de sus transacciones, se le ofrece un servicio especial en un área de la sucursal.

En INFOBOT el usuario (o cliente) utiliza los dígitos del aparato telefónico para realizar sus transacciones, orientado por una voz grabada que escucha a través de la línea telefónica.

En BANCO POR TELEFONO el usuario es una operadora de la Institución que realiza las transacciones, a través de los requerimientos que le hace el cliente por teléfono.

En TELEBANCO el usuario (o cliente) realiza las transacciones a través de una terminal conectada a la línea telefónica.

A pesar de que varios de los requerimientos son comunes, los procesos para cada uno de ellos están separados, lo que ocasiona desperdicio de recursos e ineficiencias. El objetivo de este trabajo es elaborar una propuesta para intentar resolver este problema.

1.4 Planteamiento del problema a tratar y ubicación de su contexto teórico

En este trabajo presentamos una "Propuesta de una Plataforma Transaccional de Medios Alternos Bancarios", que será el software que atenderá en forma estándar todas las transacciones provenientes de cualquier medio alterno. Esto es, convertir a Medios Alternos en un subsistema centralizado de software, en donde la Plataforma sea el elemento central.

El contexto teórico con el cual se pretende abordar este problema es la Ingeniería de Software, y el desarrollo se plantea con base en el enfoque de Calidad Total.

Capítulo 2

ELEMENTOS DE INGENIERIA DE SOFTWARE

Nuestro interés se basa fundamentalmente en el rediseño de un sistema de software ya existente. Para llevarlo a cabo nos basamos en la teoría de la ingeniería de software, y aquí presentamos los elementos de esta materia que están relacionados con nuestro trabajo. La sección 2.4 sobre el diseño del software es la más importante; las demás las incluimos aquí porque son básicas para hacer un buen diseño, que es nuestro propósito.

2.1 INTRODUCCION

La Ingeniería de Software es una actividad relacionada con el desarrollo y mantenimiento de grandes sistemas de software.

Su base principal son las ciencias de la computación (algoritmos, arquitectura de computadores, estructuras de datos).

Los grandes sistemas de software son tan complejos que resulta imposible para cualquier individuo recordar los detalles de cada aspecto del proyecto. Se necesitan técnicas más formales de especificación y diseño; debe documentarse apropiadamente cada etapa del proyecto, y es esencial una cuidadosa administración.

La ingeniería de software surgió porque hubo necesidad de técnicas y metodologías que permitieran controlar la complejidad inherente a los grandes sistemas de software.

La ingeniería de software:

a) trata de la construcción de sistemas que son más grandes de lo que normalmente podría manejar un individuo.
b) utiliza principios de ingeniería para el desarrollo de estos sistemas.
c) tiene aspectos técnicos y no técnicos.

Junto con los programas, el software incluye toda la documentación necesaria para instalar, usar, desarrollar y mantener esos programas.

2.1.1 El ciclo de vida del software

Los grandes sistemas de software requieren un tiempo considerable para su desarrollo y permanecen en uso durante un tiempo aún mayor. En este período de desarrollo y uso pueden identificarse varias etapas, que juntas constituyen lo que se llama el ciclo de vida del software, a saber:

1. *Análisis y definición de necesidades.* Los servicios, restricciones y objetivos del sistema, se establecen consultando con los usuarios. Una vez acordados, deben definirse de una manera comprensible, tanto para los usuarios como para el personal de desarrollo.

2. *Diseño del sistema y del software.* Partiendo de su definición, las necesidades se dividen en sistemas de hardware y sistemas de software. A este proceso se le llama diseño de sistemas. El diseño del software es el proceso de representar las funciones de cada sistema de software

a fin de poderlo transformar con facilidad en uno o más programas de computación.

3. *Aplicación y prueba de unidades.* Durante esta etapa, el diseño del software se realiza como un conjunto de programas o unidades de programa escritos en algún lenguaje de programación ejecutable. Las pruebas de unidades implican la comprobación de que cada unidad cumple con su especificación.

4. *Pruebas del sistema.* Las unidades de programa individuales o los programas se integran y prueban como un sistema completo para asegurar que se cubren las necesidades del software. Después de las pruebas, el sistema de software se envía al cliente.

5. *Operación y mantenimiento.* Esta fase suele ser la más larga del ciclo de vida. Se instala el sistema y se pone en uso práctico. La actividad de mantenimiento implica corregir errores que no se descubrieron en las primeras etapas del ciclo de vida, mejorar la aplicación de las unidades del sistema y aumentar los servicios de éste a medida que se perciben nuevas necesidades.

2.1.1.1 Evolución del software

Los grandes sistemas de software no son objetos estáticos. Existen en un ambiente sujeto a cambios constantes. El sistema de software deberá adaptarse a esos cambios o ir perdiendo utilidad, hasta que acaba siendo desechado.

El *mantenimiento del software* es el proceso de corregir errores en el sistema o de modificarlo para que refleje los cambios del ambiente. En los sistemas grandes, este mantenimiento se realiza mediante una serie de versiones del sistema.

Lehman propone que la evolución de un sistema de software está sujeta a varias "leyes":

1. *Cambio continuo.* Cuando un sistema de software se introduce en un ambiente, los usuarios cambian su comportamiento a medida que se familiarizan con el sistema y redefinen lo que esperan de él. El sistema debe entonces ser modificado y reintroducido en el ambiente, después de lo cual se reinicia el proceso.

2. *Complejidad creciente.* La estructura del programa original se estableció para aplicarse a un conjunto de necesidades iniciales. A medida que se produce el cambio evolutivo de esas necesidades la estructura original se degrada. Para reducir la complejidad estructural, se requiere, entonces, la reestructuración parcial o total del sistema, a fin de reflejar las necesidades en un solo punto en el tiempo.

Es claro que los costos de mantenimiento del software nunca pueden eliminarse.

2.1.2 Confiabilidad del software

La confiabilidad de un sistema de software es una medida de lo bien que proporciona los

servicios que los usuarios esperan de él.

La eficiencia de un sistema de software se mide por la velocidad de ejecución de los programas y la cantidad de memoria requerida. Mientras un programa se ejecute con mayor velocidad y requiera menos memoria se considera más eficiente.

La consecución de una alta confiabilidad implica invariablemente una buena cantidad de codificación adicional, a menudo redundante, incorporada al sistema para realizar las verificaciones necesarias. Sin embargo, a medida que los usuarios del computador adquieren experiencia, su principal criterio para la calidad del sistema es la confiabilidad más que la eficiencia. Hay varias razones para esto:

1. Un software no confiable está sujeto a que los usuarios lo eviten y, con independencia de su eficiencia, pronto carecerá de valor.

2. A medida que los equipos se hacen más baratos y veloces, hay menos necesidad de maximizar su utilización a la vez que adquiere mayor importancia la comprensión que el usuario tenga de él.

3. Un sistema eficiente se puede afinar con mucho éxito debido a que la mayor parte del tiempo de ejecución se dedica a secciones bastante pequeñas de un programa. Un sistema no confiable es mucho más difícil de mejorar, pues la no confiabilidad tiende a estar distribuida en todo el sistema de software.

4. La ineficiencia es predecible; los programas requieren mucho tiempo para su ejecución. El caso de la no confiabilidad es mucho peor. Sus errores pueden ser detectados hasta mucho tiempo después.

Si el software no es fácil de mantener, su utilidad e importancia para los usuarios disminuirá.

La confiabilidad y la facilidad de mantenimiento son considerados como los atributos más importantes de un sistema de software bien diseñado.

2.2 DEFINICION DE REQUISITOS

El proceso de establecer los servicios que debe proporcionar el sistema y las restricciones con las cuales debe operar se denomina análisis y definición de requisitos. Esta suele ser la primera fase importante del ciclo de vida del software.

También es importante distinguir entre los objetivos y los requisitos del sistema. En esencia, un requisito es algo que puede probarse, mientras que un objetivo es una característica más general que debe exhibir el sistema.

El desglose de los requisitos del software no siempre se considera como una tarea del ingeniero de software. En ocasiones, es el analista de sistemas el encargado de esta tarea, sobre todo cuando se van a automatizar sistemas manuales ya existentes.

2.2.1 El documento de los requisitos del software

La descripción precisa de los requisitos de un sistema de software se denomina documento de los requisitos del software. Se define como:

> Un conjunto de propiedades o restricciones precisamente establecidas que debe satisfacer un sistema de software.

El documento de los requisitos del software no es un documento de diseño. Debe establecer lo que ha de hacer el sistema sin especificar cómo ha de hacerlo.

Una organización de una definición de requisitos podría ser:

1. *Introducción.* Debe presentar un fundamento del sistema de software y describir con brevedad sus funciones. También debe explicar la estructura del resto del documento.

2. *Hardware.* Debe describirse el hardware en el que se va a aplicar el sistema.

3. *El modelo conceptual.* Esta sección debe describir el modelo conceptual en que se basan los requisitos. El modelo conceptual del sistema es una visión de muy alto nivel del sistema que muestra los principales servicios proporcionados por el software y las relaciones de unos con otros. Esto se expresa mejor en una notación gráfica.

4. *Requisitos funcionales.* Debe describir los servicios proporcionados al usuario.

5. *Requisitos de la base de datos.* Aquí debe describirse la organización lógica de los datos utilizada por el sistema y su interrelación.

6. *Requisitos no funcionales.* Son las restricciones bajo las cuales operará el software.

7. *Información para mantenimiento.* Debe describir los cambios anticipados del software.

8. *Glosario.* Debe definir los términos técnicos empleados en el documento.

9. *Indice.*

2.2.2 El modelo del sistema

Se usa este modelo para comprender el sistema.

Salter considera que el modelo general de un sistema es una función de tres elementos: control,función y datos. De manera intuitiva, las funciones son los transformadores de la información del sistema, los datos son las entradas y salidas de las funciones y el control es el mecanismo que activa las funciones en la secuencia deseada.

El siguiente paso en el proceso de modelado conceptual es tomar cada servicio principal del

usuario y establecer un modelo conceptual para ese servicio.
El creador del modelo debe cuidarse de no incluir un diseño del sistema.

2.2.3 Definición de requisitos funcionales

En general, la mayoría de los documentos de requisitos del software comienzan con una definición de los requisitos funcionales del sistema.
Los requisitos funcionales del sistema son aquellos servicios que el usuario espera del sistema. En general, al usuario no le interesa cómo se aplican esos servicios, así que el ingeniero de software debe evitar la inclusión de conceptos de aplicación en esta sección del documento de los requisitos.
En principio, los requisitos funcionales de un sistema deben ser completos y consistentes. Por completos se entiende que todos los servicios requeridos por el usuario deben especificarse, y la consistencia significa que ninguna definición de requisitos debe contradecir a otra. En la práctica, y para sistemas grandes y complejos, es casi imposible lograr que los requisitos sean consistentes y completos en la versión inicial del documento. A medida que se descubren los problemas durante las revisiones o en las etapas posteriores del ciclo de vida, el documento debe modificarse en consecuencia.
Hay tres maneras de expresar los requisitos funcionales de un sistema:

1. en lenguaje natural.

2. en un lenguaje estructurado o en un formato que tenga algunas reglas, pero no una especificación sintáctica o semántica rigurosa.

3. en un lenguaje formal de especificación con una sintaxis y semántica rigurosamente definidas.

2.2.3.1 Definición de requisitos en lenguaje natural

El método más utilizado para la definición de requisitos es el establecimiento de los requisitos del sistema como párrafos numerados de texto en lenguaje natural. Por ejemplo, el requisito 4.C.1 tomado del documento de Stoneman es como sigue:

4.C.1 Debe proporcionarse una interfase virtual que sea independiente de cualquier máquina huésped para la comunicación de APSE.

Obsérvese que este requisito es impreciso, descriptivo y de muy alto nivel.
Hay dos problemas principales que surgen de vez en cuando al usar el lenguaje natural para la definición de requisitos:

1. No se distinguen con claridad los requisitos funcionales, los no funcionales y los objetivos

del sistema.

2. Cada párrafo puede incluir varios requisitos individuales en una sola proposición, lo que hace muy difícil la revisión de la consistencia y de la plenitud.

El párrafo 4.C.1 citado antes define un requisito funcional de alto nivel. Sin embargo, el 4.C.2 establece lo siguiente:

> La interfase virtual debe basarse en conceptos generales simples que sean fáciles de entender y usar, y que sean pocos.

Este es un objetivo admirable del sistema, no un requisito, y es muy difícil valorar la simplicidad, facilidad, utilidad y generalidad de los conceptos del sistema.

El requisito 4.C.8 establece:

> Debe ser posible expresar todas las comunicaciones necesarias entre el APSE y el usuario en el conjunto de caracteres estándar de Ada.

En lugar de describir una función que se debe proporcionar, este requisito es una limitación práctica al diseño de un ambiente de programación en Ada, es decir, es un requisito no funcional mezclado con proposiciones de requisitos funcionales y objetivos del sistema.

Es muy difícil evitar la mezcla de los requisitos funcionales y no funcionales y de los objetivos del sistema cuando se utiliza el lenguaje natural para la definición de requisitos. La razón es que no hay una distinción notacional entre ellos, y la separación sólo se puede lograr con ayuda de cuidadosas revisiones de los requisitos.

El segundo problema esbozado antes puede detectarse de nuevo durante las revisiones de requisitos. El definidor original debe hacer un esfuerzo consciente para evitar la inclusión de múltiples conceptos en un solo párrafo. Sin embargo, esto también es difícil de lograr, porque tales conceptos tienden a estar íntimamente ligados en la mente del definidor.

Sin embargo, es necesario usar el lenguaje natural cuando se formulan requisitos de alto nivel, porque la generalidad de éstos no puede expresarse en una notación más restringida.

Además, una proposición de requisitos en lenguaje natural se puede emplear como suplemento y para documentar una definición más detallada de requisitos. De hecho, quizá sea esencial dar algunas definiciones en lenguaje natural, de modo que los usuarios sin experiencia en notaciones formales o estructuradas puedan comprender la definición de requisitos. Parnas (1979) también encontró que la expresión de requisitos en lenguaje natural era más comprensible y, sorprendentemente, menos malinterpretado por los desarrolladores del software que las definiciones más formales de requisitos.

Por lo tanto, es improbable que el lenguaje natural llegue a ser totalmente suplantado por los lenguajes formales o semiformales para la definición de requisitos. Más bien, se producirá una definición estructurada de requisitos que expresará en lenguaje natural los requisitos de alto nivel y utilizará lenguajes cada vez más formales a medida que estos requisitos se desarrollan con

mayor detalle.

2.2.4 Requisitos de las bases de datos

Muchos grandes sistemas de software necesitan una gran base de datos de información. Conforme se ejecuta, el sistema toma información de esta base de datos y se la proporciona. En algunos casos, la base de datos es independiente del sistema de software; en otros, se crea para el sistema en desarrollo. De cualquier manera, se necesita una definición de la forma lógica de esta base de datos.

2.2.5 Definición de requisitos no funcionales

Un requisito no funcional de un sistema es una restricción u obligación impuesta al servicio de éste. Ejemplos de requisitos no funcionales son las obligaciones impuestas a los tiempos de respuesta del sistema, las limitaciones en la cantidad de memoria que ocupará el software y las restricciones en la representación de los datos del sistema.

Aunque tanto los requisitos funcionales como los no funcionales están sujetos a cambios, los requisitos no funcionales se ven especialmente afectados por los cambios en la tecnología de hardware. Puesto que el tiempo de desarrollo de un gran sistema puede ser de varios años, es probable que el hardware disponible al concluir el proyecto sea más potente que el disponible cuando se concibió el proyecto. Además, el hardware evolucionará a través del tiempo de vida del software desarrollado y los requisitos no funcionales se modificarán mientras el software esté en uso.

Por ejemplo, un requisito no funcional tomado del documento de Stoneman es:

4.C.8 Todas las comunicaciones necesarias entre la APSE y el usuario podrán expresarse en el conjunto de caracteres estándar de Ada.

Este requisito restringe la libertad de un diseñador de APSE para elegir los símbolos que pueden utilizarse para activar los servicios de APSE.

A causa de la variedad y complejidad de los requisitos no funcionales, es improbable que el lenguaje natural llegue a ser desplazado por notaciones formales para definir los requisitos no funcionales.

2.2.6 Confirmación de requisitos

Una vez establecidos, los requisitos del sistema han de confirmarse. Si no se realiza ninguna confirmación, los errores cometidos en la definición de requisitos se propagarán al diseño y a la aplicación del sistema, y puede ser necesario realizar costosas modificaciones en el sistema para corregir esos errores.

El costo de los errores en el establecimiento de los requisitos puede ser muy alto, sobre todo

si no se descubren esos errores hasta la aplicación del sistema. Boehm (1974) informa que en algunos grandes sistemas hay que escribir de nuevo hasta el 95% del código para satisfacer los requisitos modificados del usuario, y también que el 12% de los errores descubiertos en un sistema de software durante un período de tres años se debieron a errores en los requisitos originales del sistema. La mayor parte del llamado mantenimiento de programas no es en realidad una corrección de código erróneo, sino una modificación de código para apoyar los cambios o errores en los requisitos originales del sistema.

En la confirmación de los requisitos hay cuatro pasos independientes:

1. Debe demostrarse la consistencia de los requisitos. Los requisitos no deben estar en conflicto entre sí.

2. Debe mostrarse que los requisitos son completos. La definición debe incluir todas las funciones y restricciones proyectadas por el usuario del sistema.

3. Debe demostrarse que los requisitos son realistas. No tiene caso especificar requisitos irrealizables mediante el empleo de la tecnología existente de hardware y software.

4. Debe demostrarse que las necesidades del usuario son válidas. Un usuario puede pensar que se necesita un sistema para realizar ciertas funciones. pero un estudio y análisis posteriores pueden identificar funciones adicionales o distintas de las requeridas.

Los otros pasos implicados en la confirmación de requisitos, la comprobación de la plenitud de los requisitos y la demostración de que el sistema cumple con las necesidades reales del usuario, sólo se puede lograr con la cooperación del usuario del sistema, que debe examinar y comprender los requisitos y comprobar si especifican la clase de sistema que en realidad se desea.

Las revisiones de requisitos regulares que comprenden tanto a los usuarios como a los ingenieros de software son esenciales mientras se formula la definición de requisitos. Durante esas revisiones, el grupo de desarrollo debe "pasear" al usuario por los requisitos del sistema y explicarle las implicaciones de cada uno. Se deben señalar los conflictos y las contradicciones, y se deja entonces que el usuario modifique sus requisitos para que resuelva esos conflictos y contradicciones.

2.2.6.1 Construcción de prototipos

El problema fundamental con que se enfrenta el usuario que trata de definir un nuevo sistema de software es que resulta muy difícil valorar cómo afectará a su trabajo la existencia de tal sistema. Para sistemas nuevos, en especial si son grandes y complejos, quizá sea imposible obtener una definición de requisitos consistente, completa y válida antes de construir y poner en uso el sistema.

Esto ha conducido a las sugerencias, analizadas en la sección 2.1, de que se debe adoptar un enfoque evolutivo en el desarrollo de sistemas. Ello implica presentar al usuario un sistema que

se sabe incompleto y después modificar y ampliar ese sistema a medida que se van evidenciando los requisitos reales del usuario. Una manera de lograr esto es mediante la construcción de prototipos, haciendo que uno de los prototipos del sistema se transforme mediante una serie de pasos en el sistema final entregado.

Hay también la alternativa de tomar una decisión deliberada para construir un prototipo "desechable". Este es un enfoque muy utilizado en el desarrollo de hardware, donde se construye un prototipo para identificar los problemas iniciales y, después de experimentar, se formula una especificación mejorada. A continuación, se desecha el prototipo y se construye un sistema de calidad de producción. Este enfoque también es aplicable al desarrollo del software.

Las ventajas de utilizar un sistema prototipo durante el análisis de requisitos y la fase de definición del ciclo de vida del software pueden resumirse como sigue:

1. Los malentendidos entre los desarrolladores del software y los usuarios pueden identificarse a medida que se demuestran las funciones del sistema.

2. Pueden detectarse los servicios que le faltan al usuario.

3. Se pueden identificar y redefinir los servicios del usuario difíciles de utilizar o confusos.

4. El personal de desarrollo del software puede encontrar los requisitos incompletos e inconsistentes durante el desarrollo del prototipo.

5. Se dispone con rapidez de un sistema de trabajo, aunque limitado, para demostrar la viabilidad y utilidad de la aplicación a la administración.

6. El prototipo puede servir como especificación para el desarrollo de un sistema de calidad de producción.

2.3 ESPECIFICACION DEL SOFTWARE

La primera etapa del ciclo de vida del software es el establecimiento de la definición de requisitos; esto es, exponer los servicios proporcionados al usuario y las restricciones a las cuales está sujeta la operación del software. Una vez aceptada esta definición, la siguiente etapa significativa del ciclo de vida es el diseño del software. Durante esta etapa, se analiza la definición de requisitos y se diseñan los componentes del software para proporcionar los servicios al usuario. Este diseño se expresa de manera que a continuación se puedan realizar estos componentes en un lenguaje de programación. La especificación del software es parte de este proceso, en el cual el diseño se expresa en una forma abstracta de alto nivel.

En sistemas pequeños, es posible pasar directamente de la definición de requisitos a un diseño detallado de componentes, pero en cuanto a grandes sistemas de software, la actividad de diseño puede dividirse en tres etapas:

1. Asociar componentes abstractos de software con los servicios establecidos en la definición

de requisitos y construir especificaciones precisas para esos componentes.

2. Construir un diseño de alto nivel que muestre la relación recíproca de los componentes abstractos del software.

3. Formular un diseño detallado para cada componente abstracto. Este diseño se expresa en términos de abstracciones más simples fácilmente traducibles a código ejecutable.

Como es natural, no hay una división clara en estas etapas y el diseñador debe repetirlas una y otra vez a medida que formula el diseño.

Se afirma a menudo que la especificación del software y la definición de requisitos son la misma cosa. En realidad, hay una clara distinción entre la especificación del software y la definición de requisitos.

La función principal de la definición de requisitos es establecer aquellos servicios que el software debe proporcionar al usuario. Dicha definición debe ser un documento orientado al usuario y ha de expresarse en términos que éste pueda comprender. Por otra parte, la especificación del software está destinada al diseñador más que al usuario. Está compuesta de definiciones abstractas de los componentes del software, no de servicios al usuario. Como al usuario no suele importarle la forma en que se proporcionan los servicios, no necesita comprender este documento.

Por supuesto, hay casos en los que existe una relación uno a uno entre componentes del software y servicios al usuario. En estos casos, la especificación del software y la definición de requisitos muy bien pueden ser equivalentes. En otros casos, sin embargo, el servicio al usuario puede ser proporcionado por la interacción de varios componentes de software, de modo que la definición de requisitos y la especificación de esos componentes son bastante distintas.

2.4 DISEÑO DEL SOFTWARE

El diseño del software es un proceso creativo que requiere del diseñador ciertas cualidades y el diseño final suele ser una repetición de varios diseños preliminares. El diseño no se puede aprender en un libro; debe practicarse y aprenderse mediante la experiencia y el estudio de sistemas ya existentes. Un buen diseño es la clave de una ingeniería de software efectiva. Un sistema de software bien diseñado es fácil de aplicar y mantener, además de ser comprensible y confiable. Los sistemas mal diseñados, aunque puedan funcionar, pueden ser caros de mantener, difíciles de probar y poco confiables. La etapa de diseño es, por tanto, la parte más importante del proceso de desarrollo del software.

Hasta no hace mucho, el diseño del software era un proceso destinado a un fin determinado. Dado un conjunto de requisitos, por lo general en lenguaje natural, se preparaba un diseño informal, a menudo en forma de organigrama. Entonces se empezaba la codificación y se modificaba el diseño a medida que se aplicaba el sistema. Al acabar la etapa de aplicación, el diseño solía diferir tanto de las especificaciones iniciales que el documento del diseño original resultaba una descripción totalmente inadecuada del sistema.

Este enfoque del diseño del software fue la causa de muchas fallas graves y caras de

proyectos. Ahora se ha comprendido que las notaciones completamente informales, como los organigramas, semejantes al lenguaje de programación, son vehículos inadecuados para formular y expresar el diseño de sistemas. Se ha reconocido que una especificación precisa (aunque no necesariamente formal) es una parte esencial del proceso de diseño y que el diseño del software es una actividad repetitiva de múltiples etapas que no puede representarse en una sola notación. En consecuencia, se han desarrollado varias notaciones de diseño como diagramas de flujo de datos, diagramas HIPO, diagramas de estructura y lenguajes para la descripción del diseño, que son más apropiados que los organigramas para expresar los diseños del software.

Dada una definición de requisitos, el ingeniero de software debe utilizarla para desarrollar el diseño de un sistema de programación que satisfaga esos requisitos. Esto se realiza en varias etapas:

1. Deben establecerse los subsistemas que componen el sistema de programación.

2. Cada subsistema debe dividirse en componentes individuales y ha de establecerse la especificación de los subsistemas definiendo la operación de esos componentes.

3. Cada programa se puede diseñar a base de subcomponentes que actúen recíprocamente.

4. Después, hay que refinar cada componente. Esto suele implicar la especificación de cada componente como una jerarquía de subcomponentes.

5. En algún momento de este proceso de refinamiento hay que especificar con detalle los algoritmos utilizados en cada componente.

Además de estas etapas del diseño de sistemas de programación, el ingeniero de software también puede tener que diseñar los mecanismos de comunicación entre los procesos. Puede tener que diseñar estructuras de archivos y con toda seguridad deberá diseñar las estructuras de datos que se utilicen en sus programas. También tendrá que diseñar los casos de prueba para comprobar sus programas.

No hay una manera definida de establecer lo que se entiende por un "buen diseño". Dependiendo de la aplicación y los requisitos del proyecto particular, el buen diseño puede ser uno que permita producir una codificación muy eficiente, puede ser un diseño mínimo donde la aplicación sea lo más compacta posible o puede ser el diseño de más fácil mantenimiento. Este último es el criterio de "bondad" adoptado aquí. Un diseño mantenible implica minimizar el costo de los cambios del sistema, y eso significa que el diseño tiene que ser comprensible y que las modificaciones deben tener un efecto local. Ambas cosas se logran si el diseño del software es muy coherente y débilmente acoplado.

Se dice que una unidad de programa es muy coherente si los elementos de esa unidad muestran un alto grado de relación funcional. Esto significa que cada elemento de la unidad de programa debe ser esencial para que esa unidad alcance su propósito, como ordenar un archivo, buscar en un diccionario, etc. Los elementos que se agrupan en una unidad de programa por alguna otra razón, como efectuar acciones que tienen lugar al mismo tiempo o que realizan varias funciones, tienen un bajo grado de coherencia.

El acoplamiento se relaciona con la coherencia. Es un indicador de la fuerza de las conexiones entre las unidades de programa. Los sistemas muy acoplados tienen conexiones fuertes en las que las unidades de programa dependen una de la otra, mientras que los sistemas débilmente acoplados se componen de unidades independientes o casi independientes.

Las ventajas obvias de los sistemas con mucha coherencia y poco acoplamiento es que cualquier unidad de programa se puede reemplazar por una unidad equivalente con poco o ningún cambio en las otras unidades del sistema. Esto es importante a la hora de refinar los diseños. Tener unidades poco acopladas significa que el diseñador posee la opción de cambiar de opinión sobre el diseño de una unidad sin que haya efectos negativos en el resto del sistema.

El diseño efectivo del software se logra mejor utilizando una metodología consistente de diseño. Hay una gran cantidad de metodologías de diseño desarrolladas y que se utilizan en diferentes aplicaciones. En esencia, la mayoría de estas metodologías se pueden clasificar en una de las tres áreas siguientes:

1. *Diseño funcional descendente.* El sistema se diseña desde un punto de vista funcional, empezando con una visión de alto nivel y refinándola de manera progresiva hasta llegar a un diseño más detallado. Dicha metodología está ejemplificada por el Diseño Estructurado y el refinamiento por pasos.

2. *Diseño orientado a objetos.* El sistema se ve más como una colección de objetos que como funciones que pasan mensajes de un objeto a otro. Cada objeto tiene su propio conjunto de operaciones asociadas. El diseño orientado a objetos se basa en la idea del ocultamiento de información y busca fortalecer la coherencia y el acoplamiento débil.

3. *Diseño controlado por los datos.* Esta metodología plantea que la estructura de un sistema de software debe reflejar la estructura de los datos que éste procesa. Por tanto,el diseño del software se obtiene de un análisis de los datos de entrada y salida del sistema.

Nos centraremos en la descomposición funcional descendente por ser la técnica más empleada, y porque, como ya dijimos, nuestro problema se puede ubicar en el marco del mantenimiento de un sistema ya existente.

2.4.1 Notaciones de diseño

Unas notaciones consistentes y completas son muy apreciadas en la creación de objetos abstractos como los sistemas de software. Sin tales notaciones, los diseños no se pueden evaluar, comparar, probar o comunicar. Aunque el programa de computación es en sí mismo la especificación absoluta del diseño, el grado de detalle que se presenta en el programa es tal que resulta inadecuado para transmitir el diseño a los lectores humanos. Esto resulta especialmente cierto en los niveles más altos del diseño, donde un sistema grande se descompone en unidades funcionales como subsistemas o subprogramas. Es muy difícil expresar esto con claridad mediante un lenguaje de programación.

No existe ninguna notación sencilla que sea idealmente adecuada para expresar un diseño de

software. Elegimos aquí tres notaciones complementarias que pueden utilizarse juntas para describir un diseño de software.

Las notaciones descritas son:

1. *Diagramas de flujo de datos.* Son diagramas que se utilizan para describir un diseño de sistemas de alto nivel; muestran cómo se transforman los datos al pasar de un componente del sistema a otro.

2. *Diagramas de estructura.* Son gráficas de jerarquía que muestran la relación estructural de los componentes de un sistema de software. De nuevo, esta notación es más útil para describir el diseño de sistemas de alto nivel.

3. *Un lenguaje para la descripción del diseño.* Es una notación con algunos atributos de los lenguajes de programación, adecuada para describir operaciones de control y de diseño detallado. Se puede usar en cualquier nivel del diseño de sistemas.

Hay que resaltar que estas notaciones son complementarias y que no hay una notación única que sea adecuada para describir todos los niveles de un diseño. Por tanto, es normal, sobre todo en grandes sistemas, utilizarlas todas para documentar el diseño del sistema. Las notaciones gráficas, en especial, son muy importantes y útiles para ilustrar la compleja estructura inherente a los grandes sistemas de software.

2.4.1.1 Diagramas de flujo de datos

Estos diagramas documentan cómo los datos de entrada se transforman en datos de salida, donde cada etapa del diagrama representa una transformación diferente.

Los diagramas de flujo de datos constan de tres componentes:

1. flechas con anotaciones,

2. burbujas con anotaciones,

3. los operadores * y +.

Las burbujas con anotaciones representan centros de transformación en los que la anotación especifica la transformación. Las flechas representan el flujo de datos hacia adentro y afuera de los centros de transformación, donde las anotaciones dan nombre al flujo de datos. Los diagramas de flujo de datos describen cómo una entrada se transforma en una salida. No deben incluir información de control o sucesión de la información. Cada burbuja se puede considerar una caja negra independiente que transforma sus entradas en salidas. Los operadores * y + se utilizan para unir flechas, * significa AND ("y" lógico) y + significa EXCLUSIVE OR ("o" exclusivo lógico).

La figura 2.1 muestra la transformación de los datos de entrada D1 y D2 en D3.1 o en D3.2.

Por convención, las entradas se hacen por la izquierda, y las salidas, por la derecha. En este ejemplo, D1 se transforma mediante T1 en D1.1, que a su vez se transforma, por medio de T2, en D1.2. D1.2 se combina con D2 en el centro de transformación T3 para producir D3. D3 se transforma por medio de T4 para producir D3.1 ó D3.2.

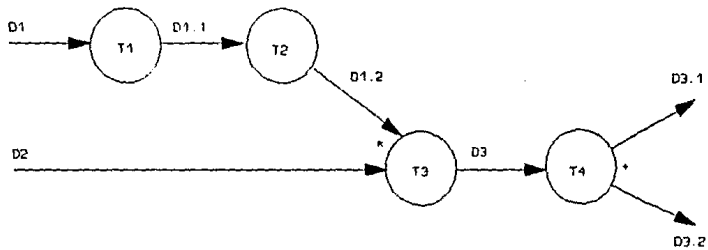


Fig. 2.1 Diagrama de flujo de datos

Como ejemplo práctico, considérese un programa de comprobación ortográfica que busca en un diccionario las palabras empleadas en un documento. Si una palabra aparece en el diccionario, se considera correcta; de otro modo, se muestra en la terminal del usuario. Este puede entonces decidir si la palabra tiene un error de ortografía o si está correctamente escrita. Si tiene error, la palabra se guarda en un archivo de palabras mal escritas; si está bien escrita, se añade al diccionario.

Hay varias maneras de realizar este sistema. En la figura 2.2 se muestra el diagrama de flujo de datos de una de esas posibilidades.

Una de las principales ventajas de los diagramas de flujo de datos es que muestran las transformaciones sin hacer ninguna suposición sobre su aplicación. En la figura 2.2, el usuario de una terminal se representa como una transformación; las otras transformaciones pueden realizarse en formas distintas. Por ejemplo, el sistema podría aplicarse como un solo programa que utilice unidades de programa para aplicar cada transformación. De manera alternativa, también puede aplicarse como varios programas separados relacionados entre sí, utilizando archivos para comunicarse unos con otros o, también, la aplicación puede ser una amalgama de estos métodos.

La preparación de los diagramas de flujo de datos se enfoca mejor si se tienen en cuenta las entradas al sistema y se trabaja hacia las salidas. Cada burbuja debe representar una transformación distinta: su salida debe, de alguna manera, ser diferente de su entrada. No hay reglas para determinar la estructura total del diagrama, y construir un diagrama de flujo de datos es uno de los aspectos creativos del diseño de sistemas. Como en todo diseño, es un proceso iterativo en el cual los primeros intentos se refinan en etapas para producir el diagrama final.

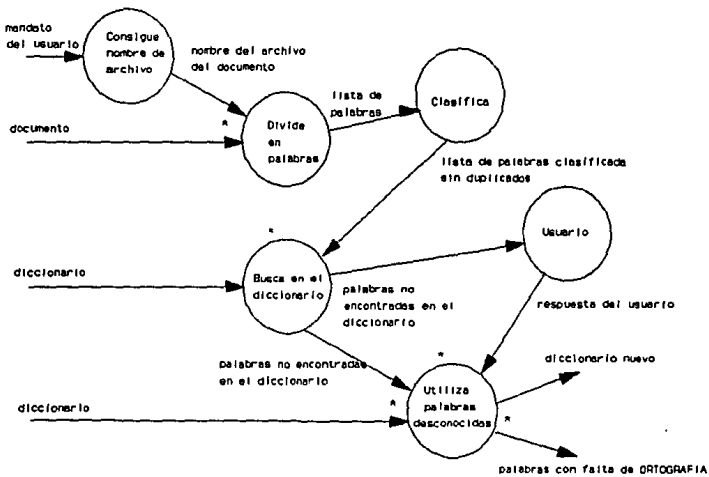


Fig. 2.2 DIAGRAMA DE FLUJO DE DATOS PARA UN COMPROBADOR DE ORTOGRAFIA

Hoy en día, utilizando las técnicas de Warnier-Orr, se trabaja desde las salidas hacia las entradas, determinando de esta manera cuál es la información mínima requerida para producir las salidas deseadas. Finalmente, lo que interesa es producir las salidas, y no procesar las entradas.

2.4.1.2 Diagramas de estructura

Los diagramas de estructura describen el sistema de programación como una jerarquía de partes y lo muestran gráficamente como un árbol. Estos diagramas documentan cómo se pueden aplicar los elementos de un diagrama de flujo de datos como una jerarquía de unidades de programa.

Un diagrama de estructura muestra las relaciones entre las unidades de programa sin incluir ninguna información acerca del orden de activación de esas unidades. Se traza mediante tres símbolos:

1. un rectángulo con el nombre de la unidad,

2. una flecha que conecta los rectángulos,

3. una flecha punteada, con el nombre de los datos que se pasan entre los elementos del diagrama de estructura. Las flechas punteadas suelen dibujarse paralelas a las flechas que conectan los rectángulos del diagrama.

En la figura 2.3 se muestra un ejemplo de diagrama de estructura.

La unidad A llama a las unidades B, C y D. La unidad B llama a las unidades E y F, y la unidad C llama a la unidad F. Obsérvese que los nodos del nivel *n* del árbol pueden ser compartidos por dos o más nodos del nivel *n-1*, pero que los nodos del nivel *n* no pueden utilizar otros nodos del mismo nivel. El ordenamiento de izquierda a derecha de B, C y D no implica que las unidades se llamen en esa secuencia.

En el diagrama de la fig. 2.3 los datos Y se originan en la unidad E, son transformados por B en Y' y pasan a A. La unidad B también pasa los datos T a la unidad F. La unidad A pasa Y' a C y C pasa T a F. C devuelve Z a A, que a su vez pasa Z a D. Las flechas de los datos originados en un nodo inferior se toman como entradas.

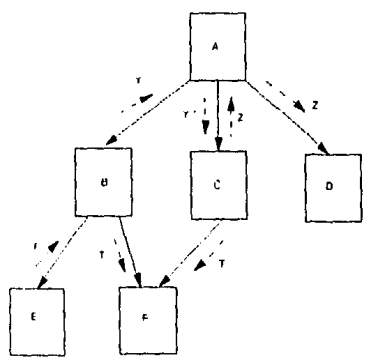


Fig. 2.3 Diagrama de estructura

Los nodos que no devuelven datos a un nodo de nivel superior se considera que dan salida a esos datos.

A partir de cualquier diagrama de flujo de cierta importancia, se pueden obtener varios diagramas de estructura distintos. Un problema importante para el ingeniero de software es cómo obtener el diagrama de estructura más apropiado a partir del diagrama de flujo de datos. Esto se analizará posteriormente.

2.4.1.3 Lenguajes para la descripción del diseño

El nivel más bajo de un diseño de software se describe mejor por medio de un lenguaje formal. Se ha dicho que la notación más apropiada para esto es un lenguaje de programación de alto nivel, como Ada o Pascal. Si bien este enfoque tiene la ventaja evidente de que el diseño es ejecutable si se dispone de un compilador adecuado del lenguaje, también tiene desventajas.

En vez de utilizar un lenguaje de programación ya existente como vehículo para expresar el diseño, una alternativa mejor es emplear un lenguaje para la descripción del diseño especialmente destinado a comunicar y documentar diseños de software. Se han inventado varios de esos lenguajes.

En esencia, los lenguajes de descripción del diseño utilizan las familiares estructuras de control de los lenguajes de programación de alto nivel para especificar el flujo del control, pero permiten al diseñador una considerable flexibilidad en la descripción de las operaciones.

En la notación utilizada aquí se emplean descripciones en español de las operaciones. Sin embargo, en donde sea apropiado, se utilizarán construcciones de control y operadores en la descripción.

En general, el diseño se refina a través de varios niveles de abstracción, a partir de una proposición de alto nivel de la operación y hasta llegar a un diseño detallado cuya representación se asemeje al lenguaje de programación. Por ejemplo, considérese el diseño de un programa para detectar palabras con faltas de ortografía en un documento:

```
procedure REVISA_ORTOG is
begin
   divide el documento en palabras
   busca las palabras en el diccionario
   muestra las palabras que no están en el diccionario
   crea un diccionario nuevo
end REVISA_ORTOG
```

Esta descripción del diseño de muy alto nivel se puede refinar con más detalle como se muestra más adelante. En ocasiones, puede ser útil llevar una descripción de diseño de alto nivel a un refinamiento en forma de comentarios. De esa forma, el símbolo de comentario __* significa que el comentario asociado representa una descripción operacional de nivel más alto. El nivel siguiente de refinamiento de REVISA_ORTOG podría ser:

```
procedure REVISA_ORTOG is
begin
   __ * divide el documento en palabras
   loop
      obtén la siguiente palabra
      añade la palabra a la lista de palabras ordenadas
      exit when todas las palabras procesadas
   end loop
   __ * busca las palabras en el diccionario
   loop
      obtén una palabra de la lista de palabras
      if palabra no está en el diccionario then
         __ * muestra palabras que no están en el diccionario
         muestra palabra, pregunta en el terminal del usuario
         if respuesta del usuario dice palabra correcta then
```

```
            añade palabra a lista de palabras correctas
          else
            añade palabra a lista de palabras erróneas
          end if
        end if
        exit when todas las palabras procesadas
      end loop
      __ * crea un diccionario nuevo
      DICCIONARIO:=intercala diccionario y lista de palabras correctas
    end REVISA_ORTOG
```

En este ejemplo se utilizaron algunas construcciones de control para describir las decisiones hechas en el programa, pero todas las operaciones reales se describieron en lenguaje natural. Los lectores familiarizados con un lenguaje de programación de alto nivel no deben tener dificultad en comprender el fragmento de programa anterior, pues usa sólo construcciones familiares de lenguajes de programación.

2.4.2 Diseño descendente

El diseño del software es un proceso creativo que no se puede formular como un conjunto de reglas. Sin embargo, el uso de una metodología sistemática simplifica el proceso de diseño y da como resultado un software comprensible, comprobable y confiable. Una de esas metodologías se llama diseño descendente o refinamiento por pasos. El diseño descendente se basa en la noción de que la estructura del problema debe determinar la estructura de la solución del software. Esta metodología utiliza la característica humana más fundamental para la solución de problemas: la abstracción.

Según el diccionario, la abstracción es el "proceso de eliminar de una idea sus acompañamientos concretos". La idea se considera como una entidad abstracta, sin especificar cómo se realiza esa entidad. El ejemplo anterior del comprobador de ortografía ilustra el proceso: revisar la ortografía implicó dividir el documento en palabras, ordenar esas palabras y después buscarlas en un diccionario. Estas operaciones se identificaron como fundamentales y, en un principio, se ignoraba cómo funcionaban en realidad. A medida que el diseño progresa, cada componente se refina en sus propias operaciones fundamentales y el proceso continúa hasta que se formula un diseño de bajo nivel.

La formulación y descripción de un diseño de software incluye varias etapas:

1. Estudiar y comprender el problema. Sin esta comprensión, es imposible el diseño efectivo del software.

2. Identificar las características generales de, al menos, una posible solución. En esta etapa suele ser útil identificar varias soluciones y evaluar cada una de ellas. Se debe elegir la solución más simple posible. Es de particular importancia no permitir detalles de aplicación de bajo nivel,de los cuales tenga conocimiento el diseñador, que interfieran en la elección

de una solución.

3. Construir un diagrama de flujo de datos que muestre las transformaciones generales de los datos del sistema. Si esto parece imposible, quizá no se haya comprendido de manera adecuada el problema.

4. Mediante el diagrama de flujo de datos, construir un diagrama de estructura que muestre las unidades de programa relacionadas con la solución.

5. Describir cada abstracción utilizada en la solución mediante un lenguaje de descripción. Es probable que en las primeras etapas del diseño esto consista casi exclusivamente en una descripción en lenguaje natural.

Después de haber formulado y descrito una solución inicial de alto nivel, el proceso de solución del problema debe repetirse para cada abstracción utilizada. Este proceso de refinamiento continúa hasta que se ha preparado una especificación de bajo nivel para cada abstracción.

Es muy importante que la representación de cada etapa del diseño sea clara y concisa. Una regla empírica útil que puede adoptarse es expresar el diseño de modo que cada parte de la especificación pueda describirse sin problemas en una hoja de papel de tamaño normal.

2.4.2.1 Obtención de los diagramas de estructura

Una etapa importante en el proceso de diseño es la transformación de un diagrama de flujo de los datos en un diagrama de estructura. Esta etapa convierte las transformaciones abstractas en una jerarquía de unidades de programa, lo que representa un paso importante en la transición de una solución abstracta de un problema en una realización concreta de esa solución.

La meta debe ser obtener un diseño donde las unidades de programa muestren un alto grado de cohesión y un bajo grado de acoplamiento. La identificación de unidades poco acopladas y con alta cohesión se simplifica si se considera que las unidades son las principales encargadas de tratar con uno de cuatro tipos de flujos de datos.

1. Entrada. La unidad de programa es la encargada de aceptar los datos de una unidad de un nivel inferior en el diagrama de estructura y de pasar esos datos a una unidad de un nivel superior en alguna forma modificada.

2. Salida. La unidad de programa es la encargada de aceptar los datos de una unidad de mayor nivel y de pasarlos a otra de menor nivel.

3. Flujo de transformación. Una unidad de programa acepta datos de una unidad de nivel superior, los transforma y los devuelve a esa unidad.

4. Flujo coordinado. Una unidad es la encargada de controlar y administrar otras unidades.

Las representaciones típicas para cada tipo de unidad en un diagrama de estructura se muestran en la figura 2.4.

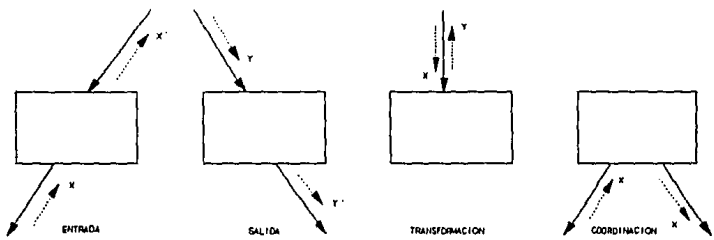


FIG. 2.4 REPRESENTACION DE UN DIAGRAMA DE ESTRUCTURA.

El primer paso en la conversión de un diagrama de flujo de datos en uno de estructura, es identificar las unidades de entrada y salida de más alto nivel. Estas unidades son aquellas que aún están implicadas en el paso de datos hacia arriba y hacia abajo de la jerarquía, aunque lo más distante de la entrada y salida físicas. Este paso no suele incluir todas las burbujas; las transformaciones restantes se denominan transformaciones centrales.

La identificación de las burbujas de entrada y salida de más alto nivel depende de la habilidad y experiencia del diseñador del sistema. Una manera posible de enfocar esta tarea es rastrear las entradas hasta encontrar una burbuja cuya salida sea tal que su entrada no se pueda deducir del examen de la salida. La burbuja anterior representa entonces la unidad de entrada de más alto nivel. Se usa un criterio similar para establecer la burbuja de salida de más alto nivel. El primer nivel del diagrama de estructura se produce mediante la representación de la unidad de entrada y de cada transformación central como una sola caja. La caja en la raíz del diagrama de estructura se designa como unidad de control. Este proceso de factorización puede entonces repetirse para las unidades de primer nivel en el diagrama de estructura hasta que estén representadas todas las burbujas del diagrama de flujo de datos. Considérese de nuevo el diagrama de flujo de datos para el programa comprobador de ortografía que se mostró en la figura 2.2.

La aplicación del criterio anterior al diagrama de flujo de datos del comprobador de ortografía sugiere que la burbuja "busca" representa una transformación central, mientras que "ordena" representa la unidad de entrada de mayor nivel, y "utiliza palabras desconocidas", representa la unidad de salida de más alto nivel. Hasta la burbuja "busca", las transformaciones son fundamentalmente rearreglos de la entrada del sistema, después de la burbuja "busca", son transformaciones "maquilladas" de la salida del sistema. La operación de búsqueda, por otra parte, es una transformación donde la entrada desaparece y la salida emerge, por lo que se puede identificar como la transformación central.

Esto hace que la siguiente estructura de primer nivel se obtenga como en la figura 2.5. Si se aplica el mismo proceso a la unidad "ordena" para obtener la estructura de segundo nivel, y si el proceso de obtención se aplica una tercera vez para lograr el diagrama de estructura final

como el que aparece en la figura 2.6. Obsérvese que "ordena" ha tomado ahora el papel de una transformación central.

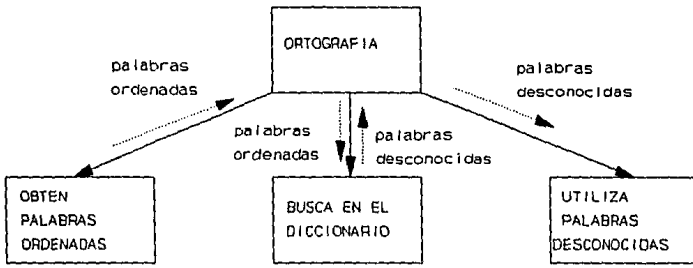


Fig. 2.5 ESTRUCTURA DE PRIMER NIVEL PARA EL COMPROBADOR DE ORTOGRAFIA.

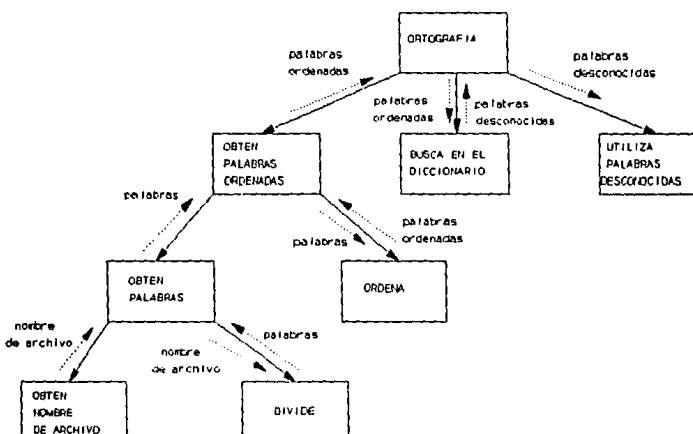


Fig. 2.6 DIAGRAMA DE ESTRUCTURA FINAL PARA EL COMPROBADOR DE ORTOGRAFIA

En general (aunque no por fuerza), es cierto que cada nodo del diagrama de estructura de un diseño bien estructurado tendrá entre dos y siete subordinados. Si un nodo tiene uno solo, significa que la unidad representada por ese nodo puede tener un grado de coherencia bajo, la

unidad lleva a cabo más de una función simple, y la existencia de un solo subordinado quiere decir que una de las funciones puede haber sido factorizada. Si un nodo tiene muchos subordinados, esto implica que el diseño se ha desarrollado a un nivel demasiado bajo en esa etapa.

2.4.3 Confirmación del diseño

La confirmación de un diseño de software es de gran importancia. Los errores y omisiones no detectados que se arrastran hasta la fase de aplicación del proyecto y que no se detectan hasta la prueba del sistema, pueden resultar muy caros de corregir. Pueden necesitar el rediseño y la reaplicación de partes completas del sistema.

El objeto de la confirmación de un diseño de software es conseguir dos cosas:

1. Mostrar que el diseño del software es "correcto"; esto es, el diseño debe aplicar de manera correcta las intenciones del diseñador. En ocasiones, este proceso se llama verificación, para distinguirlo del proceso más general de confirmación.

2. Mostrar que el software es válido; es decir, debe demostrarse que el diseño cumple con los requisitos en su totalidad. Cada requisito debe disponer de un fragmento de diseño que lo cumpla.

La tarea 2 se simplifica considerablemente si se establecen los requisitos de manera formal. Si el establecimiento es informal, las ambigüedades de los requisitos pueden originar en el diseño resultados inciertos difíciles de resolver.

La tarea principal de una revisión informal del diseño es verificar la correspondencia entre la especificación del software y su diseño.

Capítulo 3

ANALISIS CRITICO DEL SUBSISTEMA EN ESTUDIO Y PLANTEAMIENTO DE PROPUESTA

En este capítulo pretendemos analizar cómo funcionan actualmente los sistemas de Medios Alternos. En la primera sección creamos un modelo general de este funcionamiento, y a continuación pasamos a describir por medio de diagramas de flujo de datos el sistema Banco por Teléfono. Tomamos como ejemplo a Banco por Teléfono, aunque lo que digamos de él lo podemos generalizar a todos los demás sistemas que conforman a Medios Alternos.

Posteriormente, creamos un nuevo modelo de Medios Alternos introduciendo el nuevo concepto de plataforma, y a la vez mostramos por medio de diagramas de flujo de datos cómo quedaría Banco por Teléfono si funcionara bajo este nuevo concepto. Esto también se puede generalizar a todos los demás sistemas.

Los mandatos del usuario los llamamos *transacciones*, las cuales son las entradas a los sistemas de Medios Alternos. Pensando en forma abstracta, podemos considerar a un sistema como un conjunto de transacciones; y al conjunto de todos los sistemas, es decir, Medios Alternos, como el conjunto universal de transacciones. Así que pasamos a hacer una lista de las transacciones de este universo,o sea todas las posibles entradas del sistema, señalando aquéllas que pertenecen a cada sistema. Podemos incluirlas a todas en la plataforma, y sería algo válido, pero señalamos que muchas de ellas se repiten en los diferentes sistemas. Con esto mostramos la falla que tienen los sistemas de Medios Alternos en su funcionamiento actual, y la forma en que lo pretendemos remediar. Finalmente, se plantea una propuesta en la que se superan las fallas actuales.

3.1 Modelo conceptual actual de los Medios Alternos bancarios

A continuación presentamos el Modelo Conceptual Actual de los Medios Alternos Bancarios.

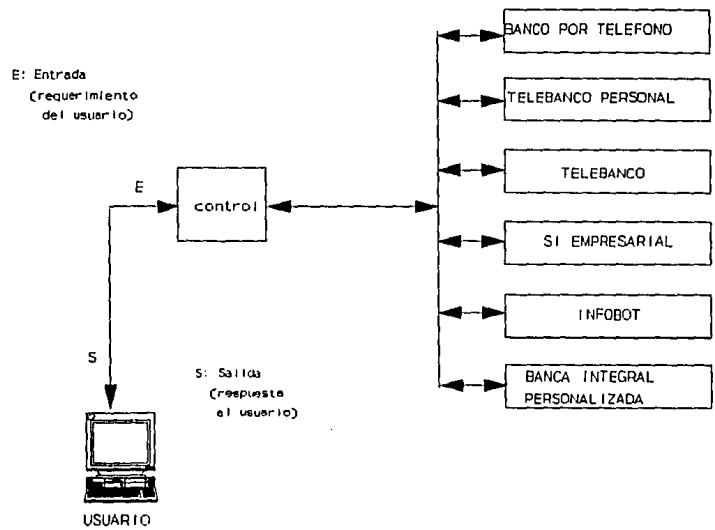


Fig. 3.1 MODELO CONCEPTUAL DEL FUNCIONAMIENTO DE MEDIOS ALTERNOS ACTUAL

Este modelo ya lo presentamos en la sección 1.3, para ejemplificar el objetivo de este trabajo. Lo volvemos a presentar, porque aquí lo pretendemos estudiar con más detalle, para adentrarnos más a fondo en el funcionamiento de estos sistemas. Este modelo describe esquemáticamente el objeto del trabajo, o sea, aquello que queremos mejorar en su funcionamiento.

La descripción de este modelo es la siguiente:

Un usuario, a través de un terminal, envía un requerimiento (E) al sistema. Este requerimiento lo recibe primero un módulo de control, que interpreta el requerimiento y lo envía a su vez al sistema que le corresponde para su proceso. Después de ser procesado, se regresa una respuesta a la terminal del usuario (S).

El módulo de control que aparece en el modelo es un producto de IBM, especialmente creado

para manejar aplicaciones en línea. Este módulo de control no lleva a cabo ninguna función aplicativa, lo único que hace es coordinar los recursos que necesitan los requerimientos del usuario para llegar a buen fin. Incluimos este módulo, porque es esencial para el funcionamiento de las aplicaciones en línea en un ambiente bancario.

3.1.1 Diagrama de flujo de datos de Banco por Teléfono

Para poder entender como funcionan los sistemas que están incluidos en el Modelo Actual de la figura 3.1, presentamos a continuación en la figura 3.2 el diagrama de flujo de datos de Banco por Teléfono (BPT). Lo que digamos acerca de él se puede extender a todos los demás.

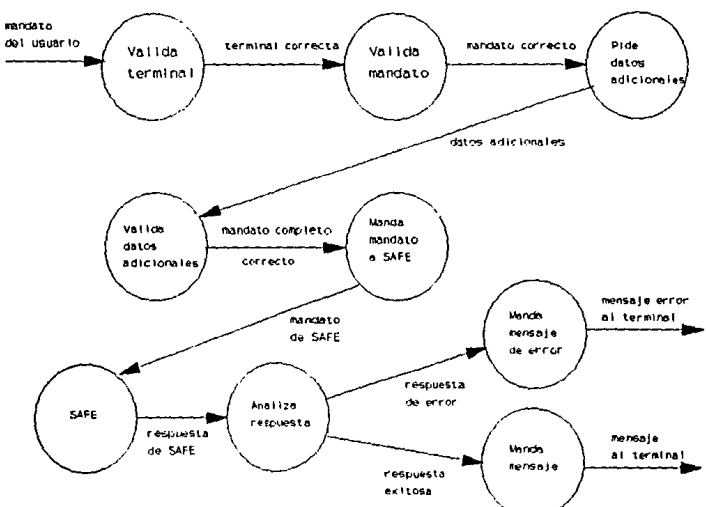


Fig 3.2 DIAGRAMA DE FLUJO DE BANCO POR TELEFONO

Este diseño fue obtenido a partir del funcionamiento actual del sistema.

El sistema se activa por medio de una solicitud desde la terminal del usuario, y siempre devuelve un mensaje que indica el éxito o fracaso de la solicitud. En el diagrama la solicitud es un mandato del usuario, que son las diferentes transacciones que puede realizar un usuario de

BPT. En una sección posterior daremos una lista completa de todas estas transacciones. Por lo pronto daremos algunos ejemplos de mandatos típicos:

COCH Consulta de saldos de las cuentas de cheques
CHTC Traspaso de una cuenta de cheques a una de tarjeta de crédito

Todas las burbujas del diagrama, que describiremos a continuación, están incluidas en el módulo "Banco por Teléfono" de la figura 3.1.

Lo primero que hace el sistema es validar que la terminal del usuario es una terminal válida de BPT, y posteriormente valida que el mandato que introdujo el usuario en su terminal es un mandato válido de BPT -los mandatos válidos de BPT los reseñamos en la sección 3.3-. Si no lo es se produce una condición de error, y el sistema le regresa un mensaje de error a la terminal del usuario. Estas condiciones de error no están especificadas en el diagrama, para hacerlo más comprensible, pero es necesario entender que a lo largo del flujo se pueden producir, y el sistema siempre le regresa un mensaje de error al usuario.

A continuación, una vez identificado el comando el sistema pide los parámetros del mandato, que es la información adicional. Ejemplo, en el mandato CHTC los parámetros son:

CHTC cuenta de cargo, cuenta de abono, importe.

A continuación valida esta información adicional. Esta validación puede consistir en checar que los datos sean numéricos, que las cuentas existan en los archivos del sistema, etc. Después, se formatea el mandato para enviarlo a SAFE. SAFE es un sistema independiente a BPT, que es quien hace realmente el cargo y el abono a las cuentas, pero aquí debido a que no forma parte de nuestro análisis adentrarnos en el funcionamiento de este sistema lo consideraremos como una caja negra que recibe una entrada y regresa una salida. SAFE responde con un código, que nos dice si la transacción fue errónea o exitosa, lo cual se convierte en una salida para el usuario. Por el análisis de este diagrama, vemos que Banco por Teléfono está formado solamente de burbujas de entrada y de salida y que la única burbuja de transformación central es SAFE, que no forma parte de BPT. Esta es una característica de todos los sistemas de Medios Alternos, y es debido a las similitudes que tienen que se adecuan para rediseñarlos con una plataforma común, que explicaremos más adelante.

3.2 Modelo conceptual propuesto de los medios alternos bancarios

Como explicaremos más adelante con más detalle, muchos requerimientos de los usuarios de los diferentes sistemas son comunes. Ejemplo: un cliente de Banco por Teléfono puede mandar el requerimiento de los saldos de sus cuentas, pero el mismo requerimiento puede solicitar un cliente de Telebanco Personal. Debido a que estos sistemas se fueron creando independientemente, sin tomar en cuenta las ligas que había entre ellos, los procesos para resolver estas transacciones comunes son independientes, teniendo con ello duplicaciones innecesarias, lo que produce una elevación de costos de operación. Podríamos entonces pensar en un modelo como el siguiente, en que tratamos de evitar esta duplicación.

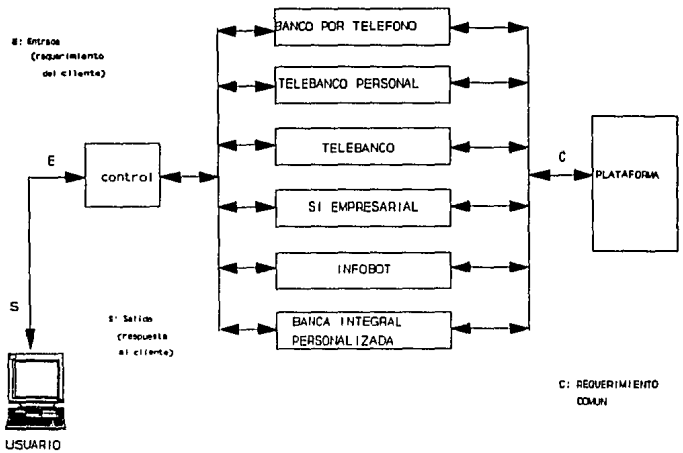


Fig. 3.3 MODELO CONCEPTUAL DEL FUNCIONAMIENTO DE MEDIOS ALTERNOS PROPUESTO

Este modelo es una variación del modelo actual, en donde sólo le quitamos a los sistemas de Medios Alternos las subrutinas correspondientes a transacciones comunes, y las mandamos a un módulo separado, al cual llamamos plataforma de Medios Alternos. Los sistemas de medios alternos, ejemplo Banco por Teléfono, sólo se quedarían con las subrutinas que son únicas para este sistema, todas las demás subrutinas irían a parar a la plataforma. En secciones posteriores iremos explicando cómo podemos llevar a cabo esto.

3.2.1 Diagrama de flujo de datos propuesto de Banco por Teléfono

A continuación, en la figura 3.4 presentamos el diagrama de flujo de datos propuesto de Banco por Teléfono, para que funcione con el concepto de plataforma.

En este diagrama, todas las burbujas que aparecen en él, excepto la burbuja "Plataforma" están incluidas en el módulo "Banco por Teléfono" de la figura 3.3. La burbuja "plataforma" pertenece al módulo "plataforma" de la figura 3.3, y la describiremos con más detalle en la sección 3.2.2. Debemos hacer notar aquí que bajo este nuevo diseño la burbuja "plataforma" ya no forma parte de BPT, sino que es un elemento separado que va a ser usado por todos los sistemas.

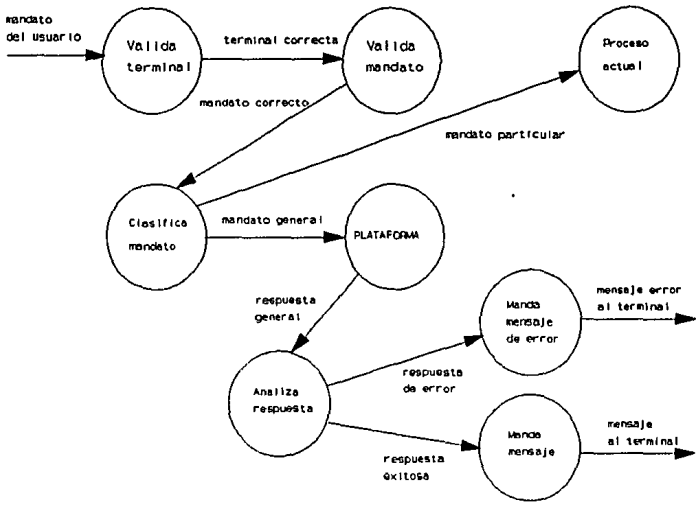


Fig 3.4 DIAGRAMA DE FLUJO NUEVO DE BANCO POR TELEFONO

Como observamos en la figura 3.4, las nuevas funciones del módulo "Banco por Teléfono" se reducen bastante. Al recibir el mandato del usuario, valida la terminal como ya explicamos en el diagrama anterior, aunque esta rutina podría quedar también dentro de la plataforma ya que todos los sistemas hacen una validación de terminal. Después valida que sea un mandato de BPT, y a continuación va a checar si este mandato es un mandato común a los sistemas, en cuyo caso lo transfiere a la burbuja "plataforma" (la cual describiremos en la siguiente sección); si no es ese el caso, se trata de un mandato propio de BPT, en cuyo caso sigue el proceso que se sigue actualmente. Al procesar el mandato la plataforma nos regresa una respuesta común a todos los sistemas. A continuación interpretamos esta respuesta para ver si fue exitoso el requerimiento o se presentó un error, y en ambos casos le regresamos un mensaje a la terminal del usuario, para darle a conocer el resultado de su solicitud.

3.2.2 Diagrama de flujo de datos de la plataforma

En la figura 3.5 presentamos el diagrama de flujo de datos de la plataforma.
Al recibir el mandato común, valida los parámetros que lleva el mandato. La plataforma sabe

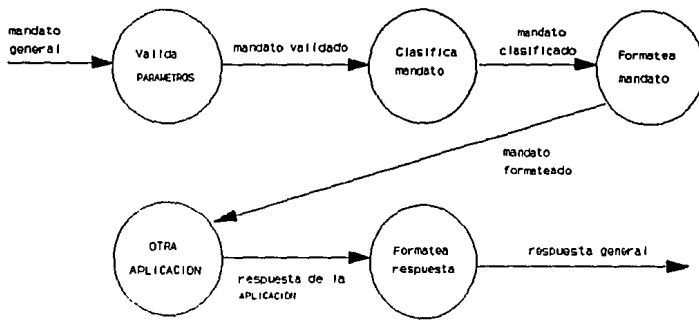


Fig. 3.5 Diagrama de flujo de datos de la Plataforma

qué parámetros debe validar, por la clave del mandato. A continuación ve que tipo de mandato es, o sea a qué aplicación está dirigido, si a SAFE, a Tarjeta de Crédito, a Bursátil, etc., y después formatea el mandato de acuerdo al formato predeterminado por la aplicación correspondiente, y lo envía a la aplicación. Como ya apuntamos en la sección 3.1.1 con respecto a SAFE, estas aplicaciones no pertenecen a los sistemas de Medios Alternos, por eso no necesitamos adentrarnos en su explicación, sólo necesitamos saber cómo le debemos enviar las entradas y cómo recibiremos las salidas de ellas. Y como observamos con respecto a SAFE, es en estas aplicaciones en donde realmente se ejecutan los mandatos. Al recibir la respuesta de la aplicación, la formateamos para regresársela al sistema que hizo el requerimiento, que en este ejemplo es BPT.

Este modelo de la plataforma está simplificado; ya en un diseño más detallado tomaría una complejidad mucho mayor.

3.3 Las transacciones de los medios alternos bancarios

En la sección anterior dijimos que los mandatos del usuario a los sistemas las llamábamos transacciones. Debemos saber cuáles de esos mandatos los va a procesar la plataforma o si los va a procesar a todos. Quizá lo más conveniente en una primera fase del desarrollo de la plataforma sea incluir en ella sólo el software asociado a los mandatos que más se repiten en los diferentes sistemas.

Para llevar a cabo ello, damos a continuación una lista de todas las transacciones de Medios Alternos. A la derecha de cada transacción aparecen las siglas de los sistemas a que pertenece cada una. El significado de las siglas es el siguiente:

S : Si Empresarial
B : Banca Integral Personalizada
L : Telebanco
T : Banco por Teléfono
P : Telebanco Personal
I : Infobot

En la tabla podemos ver cuáles son las transacciones que más se repiten, y de ahí nos podemos formar una idea de cuáles pasarán a formar parte de la plataforma.

UNIVERSO DE TRANSACCIONES

N	TRANSACCIONES	S	B	L	T	P	I
	CONSULTAS						
1	TASAS DE INTERES NACIONAL	X	X	X		X	
2	TASAS DE INTERES INTERNACIONAL	X		X			
3	TIPOS DE CAMBIO	X	X	X			
4	COTIZACIONES DE CETES Y ACEPTACIONES	X		X			
5	COTIZACIONES DE PETROBONOS	X	X	X			
6	COTIZACIONES DE BIBS	X		X			
7	SALDO DE SUS CUENTAS REGISTRADAS	X	X	X	X	X	X
8	ESTADO DE CUENTA DE UNA CUENTA ESPECIFICA (CHQ,TDC)	X	X			X	

N	TRANSACCIONES	S	B	L	T	P	I
9	REVISION DE ACTIVIDADES DEL DIA	X	X				
10	CONCENTRACION DE FONDOS	X					
11	NOTICIAS IMPORTANTES PARA UD.	X	X				
12	MOVIMIENTOS DEL DIA (de un Cliente =Contrato)	X		X		X	
13	MENSAJES INTERCLIENTES RECIBIDOS	X					
14	MERCADO CAMBIARIO	X					
15	ESTADO DE CUENTA DE UNA CUENTA EN EL EXTRANJERO	X					
16	TRANSMISION DE ESTADO DE CUENTA	X					
17	REVISION DE ACTIVIDADES DEL DIA HABIL ANTERIOR	X					
18	TRANSMISION DE CONCENTRACION DE FONDOS	X					
19	BOLETIN HISTORICO FINANCIERO		X				
20	MOVIMIENTOS (en el día) DE UNA CUENTA DE CHEQUES			X	X		
21	NOTICIERO ECONOMICO			X			
	TRASPASOS						
1	TRASPASOS (CHQ,TDC)			X	X	X	
2	TRASPASOS (CHQ,DBD)	X					
3	DE CHEQUES A TDC EMPRESARIAL	X					
4	DE CTA. NACIONAL A CTA. DEL EXTRANJERO	X					
5	DE CTA. DEL EXTRANJERO A CTA. NACIONAL	X					
6	DE CTA. DEL EXTRANJERO A CTA. DEL EXTRANJERO	X					

N	TRANSACCIONES	S	B	L	T	P	I
7	REVERSOS DE TRASPASOS (CHQ-TDC)				X		
	INVERSIONES						
1	INVERSION/LIQUIDACION DE VALORES A PLAZO FIJO	X			X		
2	COMPRA/VENTA DE CETES O ACEPTACIONES	X					
3	COMPRA/VENTA DE PETROBONOS	X					
4	FONDOS DE INVERSION	X					
5	COMPRA/VENTA DE BURSATIL CON CUENTA ASOCIADA			X		X	X
6	COMPRA/VENTA DE BURSATIL CON CUALQUIER CUENTA PROPIA(CH,TC,BUR)			X		X	
	SOLICITUDES						
1	ACLARACION Y/O COPIA DE ESTADO DE CUENTA	X	X	X			
2	USO DE SU LINEA DE CREDITO	X					
3	CHEQUERA	X	X	X			
4	LIQUIDACION DE OPERACION DE CARTERA	X					
5	ENVIO DE MENSAJES INTERCLIENTES	X					
6	EFECTIVO	X					
7	SERVICIOS GENERALES	X	X				
8	RETIRO DE CUENTA DE EXPORTADORES	X					
9	PAGO DE SERVICIOS	X					
10	CARTA DE CREDITO	X					

N	TRANSACCIONES	S	B	L	T	P	I
11	FINANCIAMIENTO FOMEX	X					
12	SUSPENSION PAGO DE CHEQUES	X					
13	CAMBIO NUMERO TELEFONICO DE ENLACE	X					
14	CAMBIO DE DOMICILIO			X			
15	BUZON DE SUGERENCIAS			X			
	DISPERSION Y TRANSFERENCIA						
1	ORDEN DE PAGO	X					
2	ORDEN DE PAGO INTERNACIONAL	X					
3	PAGO A TERCEROS	X		X		X	
4	TRANSFERENCIA DE FONDOS A OTRO BANCO	X					
5	ORDEN DE PAGO CON RECURSOS DEL EXTRANJERO	X					
	AUTORIZACIONES						
1	DE PAGO A TERCEROS	X					
2	DE ORDENES DE PAGO	X					
	OTRAS						
1	PAGO DE SERVICIOS			X		X	
2	BUZON					X	
3	CAMBIO DE NIP					X	

De acuerdo a la tabla anterior, tenemos el total de transacciones por Sistema:

SI-EMPRESARIAL	47
BANCA INTEGRAL PERSONALIZADA	11
TELEBANCO	18
BANCO POR TELEFONO	5
TELEBANCO PERSONAL	10
INFOBOT	2
TOTAL DE TRANSACCIONES	59

Las razones de que haya tanta diferencia entre el número de transacciones de algunos sistemas son variadas; por ejemplo, SI-empresarial tiene muchas transacciones porque sus usuarios son empresarios, que por la magnitud de sus operaciones, el banco trata de ofrecerles una amplia gama de servicios. Infobot al contrario, tiene pocas transacciones, porque es un servicio que apenas se está introduciendo en el mercado. Banco por Teléfono también tiene pocas, pero es porque se le ha tratado de dotar de las transacciones más comunes que cualquier cliente puede requerir del banco. Otro parámetro a considerar en la selección de las transacciones de la plataforma, es incluir en ella aquellas transacciones que tienen mayor frecuencia de uso, ya que un mal funcionamiento de ellas tiene un mayor impacto desfavorable entre los clientes del banco, cosa que no sucede con aquellas que casi no son usadas.

También, de acuerdo a la Tabla anterior, podemos calcular el número de transacciones que se repiten n veces, donde n=1,2,3,4,5 o 6. En la siguiente tabla, en la columna de la izquierda tenemos las frecuencias con que se repiten las transacciones, y en la columna de la derecha tenemos el número de transacciones que se repiten con esa frecuencia.

FRECUENCIAS	TRANSACCIONES
1	38
2	10
3	9
4	1
5	0
6	1
TOTAL	59

Podemos elegir las transacciones que se repiten 2 o más veces, que de acuerdo a la tabla anterior son 21, y eliminar aquellas que son poco usadas, y con ello ya tenemos una propuesta aceptable de las transacciones que formarán la plataforma. Esto mismo lo podemos describir más claramente por medio de diagramas de Venn, como se muestra en la figura 3.6.

En el diagrama, vemos más claro lo que se asentó en la tabla anterior. La parte rayada representa las intersecciones de 2 o más sistemas, o sea las transacciones que se repiten 2 o más

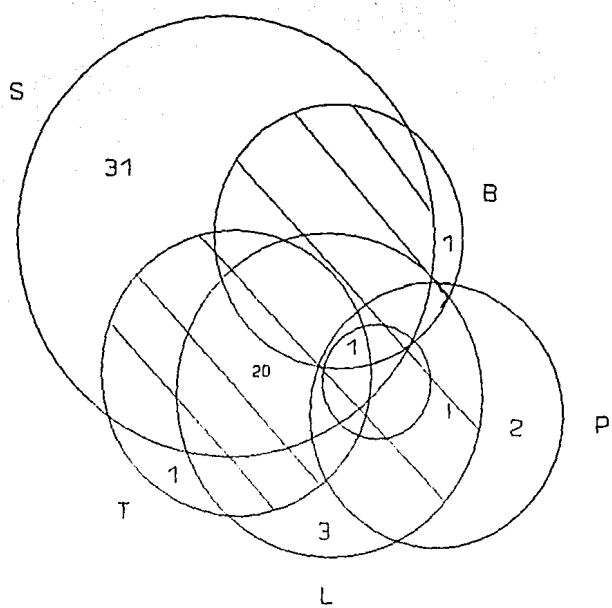


Fig. 3.6 Diagrama de las intersecciones de los sistemas de Medios Alternos

veces y que habíamos dicho que eran 21. Vemos que sólo hay 1 transacción que es común a todos los sistemas, y a los lados están representadas las transacciones que son únicas.

3.4 Propuesta

Los sistemas de software no son objetos estáticos. Existen en un ambiente sujeto a cambios constantes. A medida que el ambiente cambia o se comprende mejor, el sistema de software

deberá adaptarse a esos cambios o ir perdiendo utilidad, hasta que acaba siendo desechado.

El producto que intentamos desarrollar obedece a estos requerimientos, y además debe ser desarrollado enmarcado en uno de los objetivos fundamentales de la empresa: prestar un servicio con calidad total. Esto significa que el producto elaborado debe ser orientado a la satisfacción de las necesidades del consumidor/cliente final.

Este servicio de calidad, en la situación actual no se está prestando, debido fundamentalmente a las siguientes razones:

1. La integración de nuevos servicios es tardada y costosa, ya que cada medio alterno está sobrecargado de funciones que ya existen en los demás medios alternos, o sea, no se está aprovechando algo que ya existe. Además cada cambio que haya que hacer a los sistemas tiene que repetirse en cada medio alterno. Hay una dificultad de adaptación a cambios del medio ambiente.

2. La duplicación de funciones en los sistemas también se refleja en los grupos que atienden cada aplicación.

3. Los desarrollos no están debidamente articulados, y esto hace que la interfase con el usuario no de una imagen única, sino múltiple.

4. La evolución natural de los sistemas de medios alternos ha llevado a que su estructura general se haga más compleja, por lo tanto debemos hacer un esfuerzo para revertir este fenómeno, o sea reestructurar parcial o totalmente los sistemas y con ello conseguir que su comprensión por las personas encargadas de darles mantenimiento se haga más fácil. Hay sistemas que han evolucionado más que otros, que se han quedado rezagados, llevando con ello a que los usuarios soliciten continuos requerimientos, difíciles de atender con los recursos con que se cuenta. Esto nos lleva a la necesidad de una reestructuración que permita la fácil incorporación de los cambios y que retarden el deterioro de la estructura del sistema.

En función de la mejora de la calidad del servicio, se plantea desarrollar una PLATAFORMA DE MEDIOS ALTERNOS en los términos que se establecen a continuación:

3.4.1 Definición de requisitos

3.4.1.1 Introducción

Se trata de elaborar un producto en el que las fallas mencionadas anteriormente sean eliminadas, con el objeto de mejorar la calidad del servicio y por ende la imagen de la empresa, logrando con ello una mejora en la competitividad y la permanencia en el negocio a futuro.

OBJETIVO:

Desarrollar una plataforma transaccional para medios alternos, integral y modular.

BENEFICIOS:

1. Integración de los medios alternos, en una sola filosofía de control y servicio.

2. Permitir la integración de nuevos medios alternos en forma ágil y oportuna, descargando de muchas funciones al propio medio.

3. Atención inmediata a problemas.

4. Satisfacción del cliente o consumidor final.

5. Reducción de esfuerzos en el desarrollo y mantenimiento.

6. Presentación de una sola imagen al usuario.

7. Adaptación rápida a cambios del medio ambiente.

3.4.1.2 Requisitos funcionales

Los servicios al cliente que deberán incluirse en la plataforma son aquellos que se repiten en dos o más medios alternos; estos son:

1 CONSULTAS

1.1 Saldo de cuentas registradas

1.2 Movimientos del día (de un cliente=contrato)

1.3 Movimientos del día (de una cuenta de cheques)

1.4 Estado de cuenta (chq,tdc)

1.5 Revisión de actividades del día

1.6 Tasas de interés nacional

1.7 Tasas de interés internacional

1.8 Tipos de cambio

1.9 Cotizaciones de Cetes y Aceptaciones

1.10 Cotizaciones de Petrobonos

1.11 Cotizaciones de BIBS

1.12 Noticias importantes para Ud.

2 TRASPASOS

2.1 Traspasos (chq,tdc)

3 INVERSIONES

3.1 Inversión/liquidación de valores a plazo fijo

3.2 Compra/venta de bursátil con cuenta asociada

3.3 Compra/venta de bursátil con cualquier cuenta propia(ch,tc,bur)

4 TRANSFERENCIA

4.1 Pago a terceros

5 SOLICITUDES

5.1 Aclaración de estado de cuenta

5.2 Copia de estado de cuenta

5.3 Chequera

5.4 Servicios generales

6 OTROS

6.1 Pago de servicios

3.4.1.3 Requisitos no funcionales

El software a desarrollar va a residir en un sistema de cómputo IBM 370, y se va a desarrollar en el lenguaje COBOL con comandos CICS/MVS.

3.4.2 Modelo conceptual

En el modelo conceptual de la fig. 3.7, en los módulos de la izquierda tenemos los medios alternos, los cuales están conectados a una plataforma, que es en donde están concentradas las

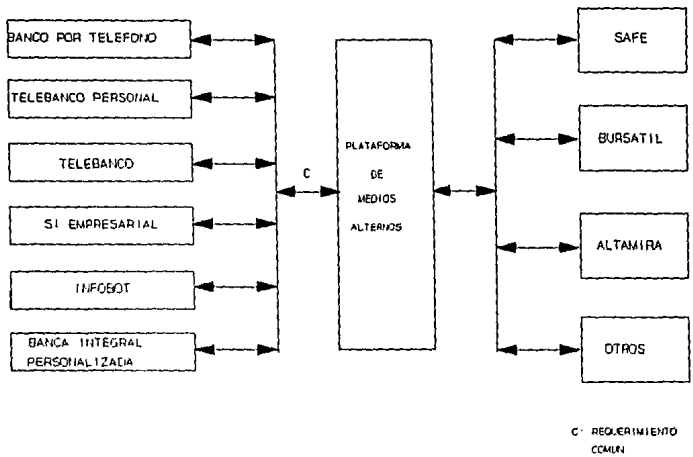


Fig. 3.7 MODELO DE LA PLATAFORMA TRX. DE MEDIOS ALTERNOS

principales funciones de los medios alternos, y a su vez esta plataforma accesa las diferentes arquitecturas aplicativas.

3.4.3 Diagrama de estructura

Una etapa importante en el proceso de diseño es la transformación de un diagrama de flujo de los datos en un diagrama de estructura. Esta etapa convierte las transformaciones abstractas en una jerarquía de unidades de programa, lo que representa un paso importante en la transición de una solución abstracta de un problema en una realización concreta de esa solución.

Los diagramas de estructura describen el sistema de programación como una jerarquía de partes y lo muestran gráficamente como un árbol.

En la figura 3.8 presentamos el diagrama de estructura de la plataforma. Este diagrama documenta cómo aplicar los elementos del diagrama de flujo de datos de la plataforma que obtuvimos en la figura 3.5 como una jerarquía de unidades de programa. Los rectángulos son las unidades de programa, y las flechas punteadas son los datos que se pasan entre los elementos del diagrama de estructura.

La unidad "PLATAFORMA" es la que controla las otras unidades.

La rama de la izquierda del diagrama representa la entrada de los datos, ya que el flujo de los

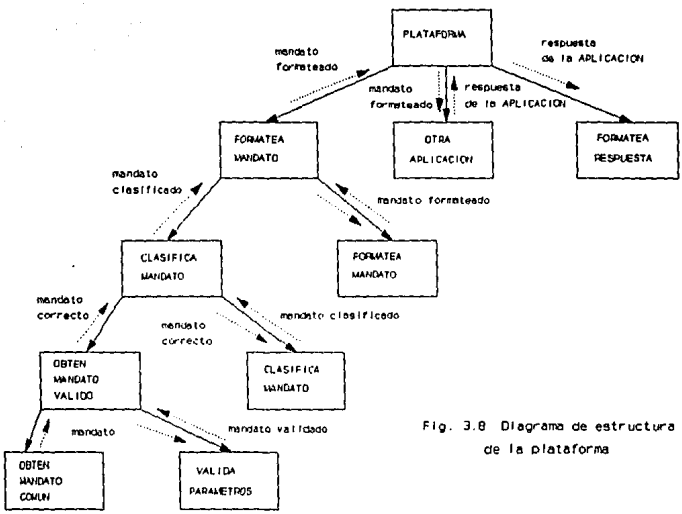


Fig. 3.8 Diagrama de estructura de la plataforma

datos va hacia arriba hacia la unidad raíz. Los datos se originan en la unidad "OBTEN MANDATO COMUN", y sufren varias transformaciones, como validar los datos, reformatearlos, pero se trata de la misma entrada.

La rama de enmedio representa la transformación central de los datos, ya que el flujo sale de la unidad raíz y regresa a ella. En esta unidad es donde se va a transformar la entrada para dar lugar a la salida. Como en esta unidad es donde se ejecuta el mandato, regresa una respuesta de éxito o fracaso a la unidad raíz.

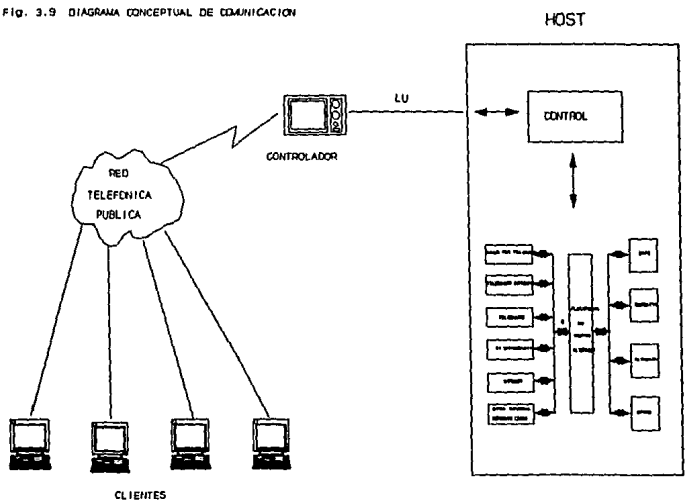
La rama de la derecha da salida a los datos, ya que el flujo sale de la unidad raíz. Aquí regresamos el resultado de la ejecución del mandato.

El siguiente paso después de haber construido el diagrama de estructura, sería describir cada unidad de programa en un lenguaje o pseudocódigo, para posteriormente pasarlo a un lenguaje de programación como Cobol.

En un paso posterior del diseño se decidirá si cada unidad de programa se aplica como un programa independiente, o podría ser que todas las unidades se apliquen como un solo programa.

3.4.4 Diagrama de comunicación

Fig. 3.9 DIAGRAMA CONCEPTUAL DE COMUNICACION



En el diagrama de la figura 3.9 tenemos el ambiente de comunicaciones de los sistemas de medios alternos, o sea cómo viaja una transacción desde el momento en que un usuario la manda desde su terminal.

La terminal se conecta con la red telefónica pública, la cual se conecta con un controlador de terminales, el cual le asigna a esta terminal una UL (unidad lógica). Estas UL se pueden considerar como una dirección de una terminal, que crea una sesión de comunicación con el equipo central (HOST). Por lo tanto la transacción llega al equipo central donde se procesa, y se regresa una respuesta al usuario o cliente siguiendo el camino inverso.

Parte IV COMENTARIO FINAL

Aunque de forma somera, creemos haber mostrado como el conocimiento y la aplicación de conceptos y metodologías provenientes de disciplinas como el Análisis de Sistemas, la Ingeniería de Software y la Calidad Total pueden resultar un apoyo valioso para quienes, en una organización, tienen la responsabilidad de tomar decisiones que permitan a ésta cumplir cabalmente su misión y con ello, sobrevivir en un ambiente de competencia cada vez más complejo y dinámico.

Apéndice CONCEPTOS BASICOS DEL CICS/VS

Todos los sistemas de información en línea de la institución funcionan en un ambiente CICS/VS, y los sistemas de Medios Alternos que son los que estudiamos aquí son sistemas en línea. Por lo tanto si queremos entender mejor el funcionamiento de estos sistemas, debemos tener algunos conocimientos básicos de lo que es el CICS/VS. Esa es la intención de incluir este tema aquí, pero como no es parte principal del tema de este trabajo lo incluimos como un apéndice.

El Customer Information Control System/Virtual Storage (CICS/VS) es un producto IBM que atiende aplicaciones en línea. Actúa como una interfase entre el sistema operativo (DOS/VS) y sus programas de aplicación (por ejemplo, los programas de Banco por Teléfono). En un entorno de procesamiento en línea, se necesitan muchas funciones de control. El CICS/VS cumple estas funciones, simplificando así el trabajo del programador de aplicación. Para el sistema operativo, el CICS/VS es una única tarea que opera en una partición/región.

Procesamiento en línea

El CICS/VS amplía en gran medida la programación de las aplicaciones en línea.

En una aplicación en línea el operador de una terminal ingresa una transacción (ejemplo: una Consulta de Saldo de Cheques), eso provoca que se ejecute el programa que atiende esa transacción; este programa lee el registro adecuado del archivo de cheques, y regresa la respuesta a la terminal del operador. La respuesta es inmediata (véase Fig. A.1).

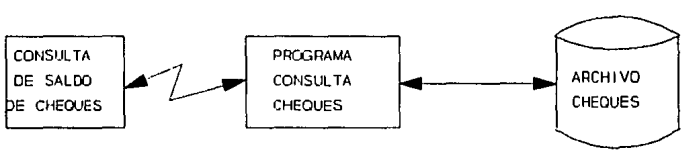


Fig. A.1 Ejemplo de entorno en línea

Una transacción ingresada en una terminal consta de dos partes: un código de transacción, también llamado 'ID de transacción' (pre-definida por el programador de sistemas) seguida de los 'datos' que han de ser procesados.

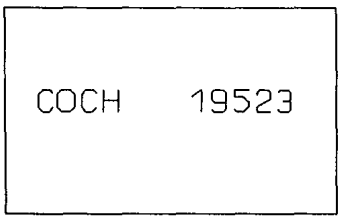
Flujo de las transacciones CICS/VS

El CICS/VS supervisa toda la actividad de comunicación de datos, lo que significa que el CICS/VS:

. administra las terminales
. administra los datos, y

. administra los programas de aplicación

Veamos como se ingresa una transacción CICS/VS en una terminal, ejemplo la transacción de consulta de saldo de cheques, y supongamos que la ID de la transacción es 'COCH' y la cuenta de cheques es '19523'.



Cuando se ingresa esta información el CICS/VS la lee en la memoria del equipo central, luego el CICS valida la ID de la transacción. Existe una tabla en donde están las transacciones válidas que puede manejar el CICS, y en ella cada transacción está asociada con un programa de aplicación.

Ejemplo:

COCH	COCHPGM

Para una ID de transacción válida, el CICS crea una "tarea" para procesar esa transacción. Esta tarea es en realidad un *bloque de control* que contiene información que el programa de aplicación y el CICS necesitan para llevar a cabo la unidad de trabajo a completar.

A continuación el CICS carga el programa de aplicación que ha sido codificado para procesar la transacción COCH.

El CICS le cede el control al programa de aplicación y comienza éste a ejecutarse. Como ésta es una transacción de consulta, el programa lee el registro del archivo maestro que tenga el número 19523.

Una vez leído el registro, el programa de aplicación "formatea" una respuesta a la consulta original, *devuelve el control al CICS/VS*, y termina la tarea.

Después el CICS transmite este mensaje a la terminal y la transacción termina.

La terminal queda disponible para el ingreso de otra transacción (véase Fig. A.2).

El CICS/VS es el programa principal de un entorno en línea. Los programas de aplicación operan bajo el CICS/VS.

En el entorno CICS/VS deben observarse dos reglas de programación:

. Todo programa de aplicación en línea debe devolver el control al CICS/VS.

. Las instrucciones no se modificarán dinámicamente.

Componentes del CICS/VS

El Sistema CICS/VS consta de:

. Módulos de Administración

. Tablas

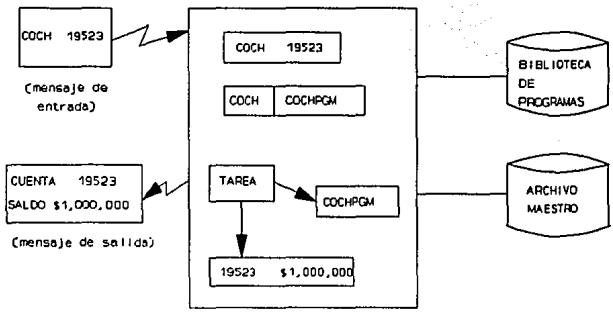


Fig A 2 Flujo de las transacciones CICS/VS

. Bloques de Control

LOS MODULOS DE ADMINISTRACION son los programas CICS/VS que mantienen una interfase entre el sistema operativo y los programas de aplicación. Todo módulo de administración cumple una función particular. Por ejemplo, cuando un programa de aplicación emite un pedido de lectura de un registro, el módulo de administración "control de archivo" (File Control Program) procesa el pedido. Cuando un programa de aplicación emite un pedido de transmisión de un mensaje a una terminal, el módulo de administración "control de terminales" (Terminal Control Program) actúa como la interfase. Los pedidos de entrada/salida se hacen al CICS/VS, en lugar de hacerlo al sistema operativo como en el caso de procesamiento en lotes.

A nivel programa de aplicación, la administración de las comunicaciones por el CICS/VS es tan completa que la comunicación con las terminales puede ser tan simple como leer un registro de entrada o escribir una línea de salida.

El CICS/VS toma a su cargo la parte difícil de las operaciones de entrada/salida, dejando al programa de aplicación sólo la lógica de la ejecución(véase Fig. A.3).

LAS TABLAS definen el entorno del sistema CICS/VS. Las tablas, generadas por el programador de sistemas, están funcionalmente asociadas con los módulos de administración. Por ejemplo, todas las definiciones de archivos están en la "tabla de control de archivos" (File Control Table), de modo que todos los programas de aplicación y tareas pueden compartirlas. Por este motivo, los archivos no se definen en el programa de aplicación, a diferencia de los

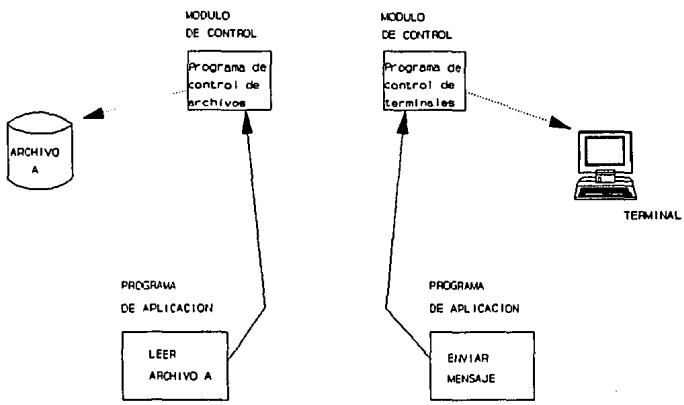


Fig A.3 LOS MODULOS DE ADMINISTRACION

programas batch. La "tabla de control de terminales" (Terminal Control Table) define cada terminal de la red; el programa de aplicación no necesita así ocuparse de los atributos físicos de las varias terminales del sistema(véase Fig. A.4).

LOS BLOQUES DE CONTROL contienen información sobre el tipo de sistema. Al iniciarse una transacción se crea un bloque de control llamado Area de Control de Tarea (TCA). La TCA es el bloque de control que el CICS utiliza para controlar el procesamiento de una transacción. Provee acceso a todas las áreas de memoria adquiridas en conexión con la tarea. La TCA sirve asimismo como vehículo de comunicación entre el programa de aplicación y los módulos de administración del CICS/VS.

Al inicializarse el CICS/VS se cargan en el almacenamiento todos los principales módulos de administración, tablas y el Area Común del Sistema (CSA). La CSA es el principal bloque de control del CICS/VS.

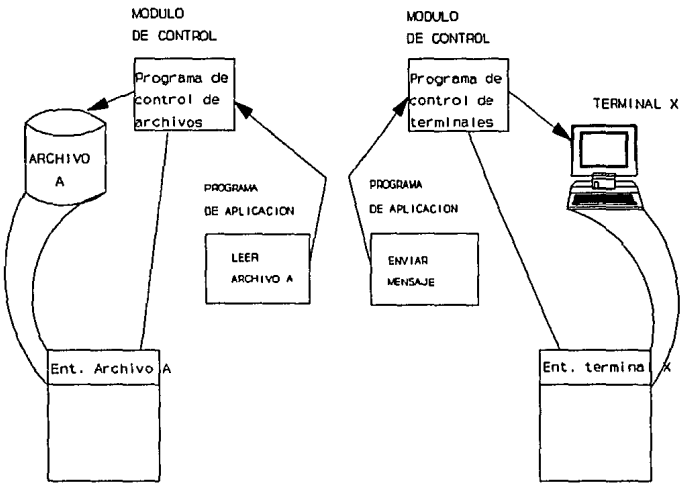


Fig. A.4 Las tablas del CICS/VS

BIBLIOGRAFIA

Ghezzi, C.; Jazayeri, M.; Mandrioli, D., *Fundamentals of software Engineering*, Prentice-Hall Inc., 1991.

Juran, J. M., Editor-in-Chief, *Juran's Quality Control Handbook*, McGraw-Hill Inc., 1988.

Optner, Stanford L., *Análisis de sistemas*, Fondo de Cultura Económica, 1978.

Rosander, A. C., *La Búsqueda de la Calidad en los servicios*, Ediciones Díaz de Santos, S.A., 1992.

Sommerville, Ian, *Ingeniería de Software*, Addison-Wesley Iberoamericana, 1988.