31

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

"DISEÑO E IMPLEMENTACION DE UN SISTEMA CLIENTE/SERVIDOR DE RESERVACION DE HOTELES"

T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACION

P R E S E N T A N :

JUAN ALEJANDRO CASTILLO VENTURA

JOSE DAVID FERNANDEZ CASILLAS

DIRECTOR: ING. ALBERTO TEMPLOS C.

CIUDAD UNIVERSITARIA, MEXICO, D. F. 1998

TESIS CON
FALLA DE ORIGEN

Diseño e Implementación de un Sistema Cliente/Servidor de Reservación de Hoteles

INTRODUCCION.

Es necesario examinar las fuerzas que nos están afectando. Las fuerzas políticas, económicas y competitivas están forzando a las compañías y organizaciones alrededor del mundo a que reestructuren la forma en que se organizan y la manera en que pretenden llegar a sus objetivos. Va desde la forma misma en que las organizaciones están estructuradas hasta el comportamiento de los empleados en la organización modelo.

Estamos compitiendo en un mundo completamente diferente con el cual competíamos anteriormente; el campo de acción se esta reduciendo; economías locales nos afectan a todos, donde todos estamos expuestos a las consecuencias. No existe mayor evidencia que aquella que se presentó recientemente acerca de la comunidad económica europea, la cual ha afectado a la mayor parte del mundo. El mercado ha disminuido, los compradores están relacionados, no por la moneda ni por el lenguaje, sino por bits, bytes, intercambio electrónico y comunicaciones satelitales.

Las fuerzas políticas están creando nuevos bloques de tratados en Europa, Asia y hasta en los Estados Unidos con el Tratado de Libre Comercio. Los mercados financieros operan siempre junto con el reloj. Los negocios son siempre conducidos internacionalmente las 24 horas del día, 7 días a la semana. En este mundo nuevo en el que vivimos, ya no funciona la manera en que antiguamente se operaba.

Simultáneamente, todos nos enfrentamos a diferentes opciones, con una gran cantidad de críticas emergentes de tecnologías variadas -- tecnologías como micros, comunicaciones de banda ancha, miniaturización y software de altas funciones. La dificultad es, que con este amplio campo de tecnologías, no hay un plan maestro, un sólo camino al éxito, una sola arquitectura que soporte este ambiente que está emergiendo.

Hace una década, las industrias de la computación y la comunicación, eran industrias separadas. En estos días estas industrias se están uniendo con la era de la digitalización. Se están creando opciones que nos ofrecen la solución a nuestros problemas, siempre y cuando podamos encontrar el camino al éxito.

De la manera en que se persigue el éxito en este mercado cada vez más competitivo y, que a grandes pasos esta creciendo, se organizan de nuevo las ideas y/o acciones que se hayan especificado -- sistemas de manufactura, administrativos, la manera en que hacemos uso de los recursos humanos, los productos, los servicios, y aun los valores y culturas que se mantienen en alto para la compañía.

Muchas empresas están siendo desplazadas con la esperanza de que una gran fuerza de trabajo, desencadene soluciones nuevas e innovadoras a problemas y oportunidades. Estos dos términos --fuerza de trabajo y soluciones innovadoras-- es de lo que CLIENTE/SERVIDOR trata. Es por eso que Cliente/Servidor es la llave del éxito para la competencia en el siglo XXI. A través de esta década y mas allá, el éxito pertenecerá cada vez más a las compañías eficientes que capturen el mercado con los mejores productos en menos tiempo.

Las empresas hoteleras mexicanas no están exentas de todos estos problemas. Demandan soluciones que incrementen sus niveles de calidad y servicio, haciéndolas más competitivas frente a las industrias pertenecientes a otras latitudes.

Justamente la presente tesis es lo que persigue, dotar a la industria hotelera mexicana con un sistema que la haga más eficiente y competitiva acorde a sus objetivos de calidad y servicio.

CAPITULO 1. CONCEPTOS GENERALES.

1.1 Qué es Cliente/Servidor?

Aunque las palabras Cliente/Servidor son frecuentemente pronunciadas actualmente, no hay ningún acuerdo acerca del significado de su concepto. Esto nos brinda una gran oportunidad para crear nuestra propia definición. Como el nombre lo dice, Clientes y Servidores son dos entidades lógicas separadas que trabajan juntas sobre una red para realizar una tarea. Así que, ¿qué hace Cliente/Servidor diferente de otras formas de Software distribuido? Creemos que todos los sistemas Cliente/Servidor tienen las siguientes características que los distinguen:

- **Servicio**: Cliente/Servidor es, primeramente, una relación entre procesos corriendo en máquinas separadas. El proceso Servidor es quién provee los servicios. El Cliente es quien consume los servicios. En esencia, Cliente/Servidor provee una limpia separación de funciones basadas en la idea de servicio.

- **Recursos Compartidos**: Un Servidor puede dar servicio a muchos Clientes al mismo tiempo y puede regular su acceso a los recursos compartidos.

- **Protocolos asimétricos**: Hay muchas relaciones Muchos a Uno entre Clientes y Servidores. Los Clientes siempre inician el diálogo solicitando el servicio. Los Servidores esperan pasivamente peticiones de los Clientes.

- **Transparencia de locaciones**: El Servidor es un proceso que puede residir en la misma máquina que el Cliente o en diferentes máquinas a lo largo de la red. El Software Cliente/Servidor usualmente disfraza la ubicación del Servidor redireccionando las llamadas de servicio (cuando se necesita). Un programa puede ser un Cliente, un Servidor, o ambos.

- **Mezclar y juntar**: El Software Cliente/Servidor ideal es aquél que es independiente del Hardware o del Sistema Operativo en el que se esté ejecutando. Debemos de ser capaces de mezclar y juntar las plataformas en donde residan tanto el Cliente como el Servidor.

- **Mensajes basados en intercambios**: Los Clientes y los Servidores son sistemas ligeramente acoplados en donde la interacción se lleva al cabo a través de

mecanismos de intercambio de mensajes. El mensaje es el mecanismo deliberado para realizar peticiones de servicios y réplicas.

- **Encapsulación de servicios**: El Servidor es un especialista. Un mensaje le dice al Servidor que servicio es el solicitado y depende del Servidor el determinar cómo llevar al cabo el trabajo. Los Servidores pueden crecerse sin afectar a los Clientes siempre y cuando no se realicen cambios a la interface de mensajes.

- **Escalabilidad**: Los sistemas Cliente/Servidor pueden ser escalados horizontal y verticalmente. El escalamiento horizontal significa agregar o remover estaciones de trabajo Clientes con solo un pequeño impacto en el rendimiento. El escalamiento vertical significa migrar a un Servidor más robusto y rápido o a un conjunto de Servidores.

- **Integridad**: El código y los datos del Servidor son mantenidos centralmente, lo cual nos brinda como resultado un mantenimiento barato y un resguardo de la integridad de los datos compartidos. Al mismo tiempo, los Clientes permanecen únicos e independientes.

1.2 Descripción de la plataforma.

Muchos sistemas con muchas arquitecturas diferentes han sido denominados como "Cliente/Servidor". Los vendedores de sistemas seguido usan el concepto de Cliente/Servidor como si el término únicamente aplicara a sus paquetes. Por ejemplo, los vendedores de Servidores de Archivos aseguran que ellos fueron los primeros en inventar el término, y los vendedores de Servidores de Bases de Datos son conocidos en algunos círculos como los únicos vendedores de Cliente/Servidor. Pero entonces, ¿quién es el que tiene la razón? La respuesta es ambos.

La idea de contar con aplicaciones que involucran diferentes líneas de Cliente/Servidor ha sido usado a lo largo de los últimos años con la finalidad de contar con varias soluciones de Software para redes de área local. Cada una de esas soluciones, sin embargo, se distingue por la naturaleza del servicio que provee para con sus Clientes, como se muestra en los siguientes ejemplos:

- **Servidor de Archivos**: Bajo un Servidor de Archivos, el Cliente (típicamente una PC) pasa sus peticiones de registros de archivos sobre una red hacia el Servidor de Archivos. Esta es una forma primitiva de servicio de datos que hace necesario muchos intercambios de mensajes sobre la red para encontrar los datos requeridos. Los Servidores de Archivos son útiles para compartir archivos a través de una red. Son indispensables para crear un repositorio de

documentos, imágenes, diseños ingenieriles, y otros muchos tipos de objetos de datos.

- **Servidor de Base de Datos**: Con un Servidor de Base de Datos, los Clientes hacen llegar peticiones SQL (Structured Query Language) en forma de mensajes al Servidor de Base de Datos. El resultado de cada comando SQL es regresado a través de la red. El código que procesa esa petición SQL y los datos residen en la misma máquina: El Servidor usa su propio poder de procesamiento para encontrar los datos solicitados, en lugar de pasar de regreso todos los registros al Cliente y dejarlo que el encuentre la información que requiere como es el caso del Servidor de Archivos. El resultado es un uso más eficiente del poder de procesamiento distribuido. Los Servidores de Base de Datos proveen las bases para Sistemas de Toma de Decisiones los cuales requieren peticiones *ad-hoc* y reportes flexibles.

- **Servidor de Transacciones**: Con un Servidor de Transacciones, el Cliente invoca procedimientos remotos que residen en el Servidor junto con una máquina de Base de Datos SQL. Estos procedimientos remotos que están en el Servidor ejecutan una serie de sentencias SQL. El intercambio en la red consiste en un solo mensaje de petición y respuesta (lo cual difiere del Servidor de Base de Datos el cual maneja un mensaje de petición/respuesta por cada sentencia SQL dentro de una transacción). Se necesita usar un protocolo punto a punto para concretar la llamada al procedimiento remoto y así obtener resultados. Las sentencias SQL, como unidad, pueden ya sea tener éxito o fracaso. Estas sentencias SQL agrupadas son llamadas transacciones. Uno puede crear las aplicaciones Cliente/Servidor simplemente escribiendo el código para ambos componentes, para el Cliente y para el Servidor. El componente Cliente usualmente incluye interfaces gráficas de usuario (CUI-Common User Interface). El componente Servidor usualmente consiste de transacciones SQL hacia la Base de Datos. Estas aplicaciones tienen un nombre, "Procesamiento de Transacciones en Línea" o simplemente PTL. Ellas tienden a ser aplicaciones críticas para el negocio las cuales requieren de 1 a 3 segundos de tiempo de respuesta el 100% del tiempo. Las aplicaciones PTL también requieren de robustos controles de seguridad e integridad sobre la Base de Datos.

- **Servidor de Aplicaciones**: Con un Servidor de Aplicaciones, uno proporciona el código tanto para el Cliente como para el Servidor. Los Servidores de Aplicaciones no contienen necesariamente a la Base de Datos en forma centralizada.

1.3 Sistemas Cliente/Servidor actuales.

Las compañías se están enfrentando a nuevos retos que afectan en la manera en la que la gente obtiene y usa la información corporativa por lo que están implantando el modelo de cómputo Cliente/Servidor como un medio de proporcionar la información a las personas adecuadas en el momento oportuno para la toma de decisiones. Los beneficios de este modelo de cómputo son muchos y pueden variar de una compañía a otra. Las organizaciones tienen la posibilidad de obtener el control de las aplicaciones mejorando el tiempo de respuesta de las transacciones. Al descentralizar las soluciones aplicativas e implantarlas más cerca del usuario, este tiene ahora la facilidad de controlar sus propias necesidades de información. El departamento de sistemas se convierte ahora en el proveedor de los servicios de cómputo en lugar, de seguir siendo el controlador de los recursos de cómputo de la compañía.

A través del tiempo se logran mejoras en la productividad mediante el uso de interfaces gráficas de usuario (GUI, *Graphic User Interfase*) las cuales estan disponibles en las estaciones de trabajo inteligentes. Los usuarios con poco o ningún conocimiento técnico pueden solicitar un servicio o usar una aplicación sin tener que realizar ningún comando complicado. También, la estación de trabajo inteligente, en conjunto con las herramientas de desarrollo de aplicaciones prevalecientes son un excelente medio para mejorar la productividad del programador. Al integrar los sistemas de información corporativa, los departamentos de sistemas de información han tenido que fusionar los sistemas existentes con los nuevos, y conectar también aquellos que no eran compatibles. Aunque esto puede ser también costoso, las compañías han desarrollado un entendimiento real de la importancia de las comunicaciones. Muchas compañías no pueden cuantificar los ahorros inmediatos al migrar a un ambiente Cliente/Servidor, pero están identificando cómo pueden eficientar cada vez más sus negocios, por lo que están ponderando esta eficiencia en relación a los costos. El modelo Cliente/Servidor maximiza los beneficios de la inversión hecha por las compañías en la adquisición de recursos de cómputo permitiendo la explotación de las fortalezas de cada una de las plataformas involucradas en el sistema. Ya que se está implantando este modelo, las compañías podrán responder más rápidamente a las cambiantes condicionantes de negocios, incrementando o reduciendo los recursos de cómputo según se requiera.

Una investigación de Forrester Research establece que las aplicaciones Cliente/Servidor críticas en los negocios y que van marcando la pauta son: servicio a Clientes, inventarios, recursos humanos, ventas y mercadotecnia y manejo de pedidos.

1.4 El futuro de Cliente/Servidor.

Se dice con frecuencia que la década de los 90 es la "era de la información". Existen sin embargo diversos puntos de vista sobre la manera en que debería efectuarse el procesamiento de datos, aunque la mayoría de quienes opinan, coinciden en que nos encontramos en medio de un proceso de evolución que se prolongará todavía por algunos años y que cambiará la forma en que obtenemos y utilizamos la información almacenada electrónicamente.

El principal motivo de esta evolución es la necesidad que tienen las empresas de realizar sus operaciones más eficientemente, debido a la creciente presión competitiva a la que están sometidas, lo cual se traduce en la necesidad de que su personal sea más productivo, que se reduzcan los costos y gastos de operación, al mismo tiempo que se generan productos y servicios más rápidamente y con mayor calidad.

En este contexto, es necesario establecer una infraestructura de procesamiento de información que cuente con los elementos requeridos para proveer información adecuada, exacta y oportuna para la toma de decisiones y proporcionar mejor servicio a los Clientes.

El modelo Cliente/Servidor reúne las características necesarias para proveer esta infraestructura, independientemente del tamaño y la complejidad de las operaciones de las empresas y consecuentemente desempeña un papel importante en el proceso de evolución.

Aunque hoy la mayoría de los ambientes son homogéneos (la mayoría DOS o UNIX) debemos estar preparados para lidiar con nuevos ambientes durante esta década, más heterogéneos, basados en el modelo Cliente/Servidor. Según se establece en un estudio de Computer Associates denominado "Arquitectura para los 90's", esta tendencia hacia sistemas heterogéneos proviene de tres diferentes direcciones:

- En primer lugar tenemos el Downsizing desde soluciones de mainframe que está requiriendo de una plataforma robusta para enfrentar las demandas de desempeño e integridad de aplicaciones de misión crítica.

- Después tenemos la tendencia representada por esquemas de minicomputadoras utilizando sistemas tipo UNIX, donde ya es común que utilicen redes con arquitectura de aplicación Cliente/Servidor, pero que en el nuevo mundo utilizarán mucho más las PCs.

- Y en tercer lugar tenemos el proceso de Upsizing de soluciones PCs, donde soluciones exitosas que han sido construidas para individuos o pequeños grupos, crecen para atender grupos de usuarios, de datos y transacciones mucho más grandes.

En esta transición los ambientes de red heterogénea Cliente/Servidor, los tres grupos, con sus muy diferentes antecedentes, tienen las mismas preocupaciones, y muchas veces encuentran que pueden aprender de los otros. Todos enfrentan un reto sin precedentes: construir importantes aplicaciones de negocios de misión crítica con la seguridad, integridad, confiabilidad y productividad de mainframe, la facilidad de uso y conveniencia de una PC, y el poder, economía y flexibilidad de un sistema UNIX. Estas soluciones requieren lo mejor de todos los mundos.

En este estudio, "Arquitectura para los 90's", se sostiene que el ambiente computacional para el futuro es una red de área local (LAN) heterogénea y bajo el modelo Cliente/Servidor. En esta red las instalaciones de trabajo (Clientes) tendrán interfaces gráficas, habrá una mezcla de Servidores de alto poder, de Sistemas Operativos y de marcas de Hardware. En esa red también habrá conectividad a otras redes y a mainframes.

En los últimos dos años, el modelo Cliente/Servidor ha venido ganando mayor aceptación en la computación corporativa. Dos tercios de los profesionales encuestados, por Business Research Group, del mundo de la computación corporativa dijeron que ellos han implementado o piensan implementar este modelo dentro de dos años. También proyectan que el mercado de los Estados Unidos para computación Cliente/Servidor crecerá de un estimado de $33.5 billones de dólares en el 93 a más de $44 billones de dólares en 1995.

1.5 ¿Porqué las Empresas se están orientando a Cliente/Servidor?

En los últimos años, se he producido un cambio fundamental en la estructura y utilización de Sistemas de Información. Cambio que surgió en gran medida como consecuencia natural de las modificaciones en la industria, en la tecnología disponible y en la manera que las empresas realizan sus negocios.

Hoy en día la computación distribuida es ampliamente reconocida como la manera más adecuada de aprovechar los sistemas de información, en contraste con la computación centralizada, que era hasta hace un tiempo la manera común de manejar sistemas.

En los párrafos siguientes se mencionan las razones que las empresas enuncian para explicar porqué implementaron soluciones Cliente/Servidor.

1.5.1 *Centrado en el Cliente.*

Es oportuno remarcar la mención del Cliente en primer lugar ya que es el usuario final que palpa en el día a día de trabajo los beneficios de una aplicación Cliente/Servidor correctamente implementada. Y esto también es benéfico para el manejo de sistemas.
El ambiente de sistemas de las empresas se hace más atractivo, los programas tardan menos tiempo en desarrollarse y se tiene un mayor control tanto sobre los datos, como sobre las máquinas involucradas.

1.5.2 *Información necesaria acorde a las necesidades.*

Como los datos pueden ser accedidos por los usuarios finales, por supuesto cumpliendo los niveles de seguridad establecidos, esos usuarios se ven totalmente satisfechos ya que no necesitan recurrir como se hace actualmente en muchas empresas, a solicitudes especiales de listados que una vez recibidos no siempre contienen los datos que necesitan.

1.5.3 *Ordenamiento de las PCs repartidas por las empresas.*

Es por todos conocida la realidad de que algunos sectores de las compañías comenzaron a adquirir PCs, obteniendo enorme incremento en su productividad, cosa que otros departamentos imitaron formando así pequeñas islas de información, de tal modo que a lo largo de toda la empresa proliferan un número importante de máquinas imposible de administrar y controlar.

Con implementaciones de Cliente/Servidor, estas PCs se integran al conjunto de sistemas lográndose entonces organizar las tareas de administración, autorizaciones, back-up, distribución de Software, uniformidad de versiones, junto con la posibilidad de conectarse y comunicarse con el resto de la empresa.

1.5.4 *Ventajas competitivas de las compañías.*

En todo el mundo y aquí en México nuevas aplicaciones Cliente/Servidor están siendo implementadas y perfeccionadas para cubrir las necesidades de las empresas.

Bancos instalando cajeros automáticos y procesamiento de préstamos que aumentan la productividad del personal, disminuyendo los tiempos de respuesta y brindando así una mayor calidad de servicio a sus Clientes.

Diarios y revistas se diseñan y producen bajo sistemas y aplicaciones Cliente/Servidor con el objeto de reducir los tiempos de realización de ejemplares y para que la información pueda ser revisada por distintas personas preservando la integridad de los datos.

Para eliminar las tareas administrativas que implica la búsqueda de documentos, compañías de seguros están implementando el procesamiento de imágenes y siniestros vía aplicaciones Cliente/Servidor.
Empresas de manufactura están desarrollando sistemas de inventario y distribución que logran reducción de costos optimizando los procesos de stock y fabricación. La computación Cliente/Servidor ofrece un enorme campo de aplicación para todo tipo de compañías.

1.5.5 ¿Centrado en el Host o centrado en grupos de Trabajo?

Comprendiendo la filosofía de Cliente/Servidor, es evidente que las soluciones de computación en los '90 se han compuesto, de procesadores centrales (Host) conviviendo con grupos de trabajo (Workgroups) o redes locales de procesadores más pequeños. Según el tipo de trabajo a realizar tendrán mayor preponderancia unos u otros.

Los siguientes puntos dan una idea de donde está (estará) el peso en las aplicaciones específicas:

- **Acceso a los datos.** Aplicaciones que requieren el compartir datos entre una gran comunidad de usuarios son buenos candidatos para implementaciones centralizadas.

- **Seguridad.** Aplicaciones que requieren auditorías y capacidad de recuperación, así como gran cantidad de datos, usualmente requieren ambientes de mainframes debido a la gran variedad de herramientas que para estos fines se encuentran disponibles. Esta es una de las razones que justifica que estos equipos sean más complejos de manejar.

- **Administración de sistemas.** Redes de gran desarrollo son candidatos ideales para las aplicaciones de administración centralizadas propias de los mainframes. Al dejar estas tareas de rutina en manos del equipo central, se libera a los usuarios finales de la realización de respaldos, instalación de nuevas versiones de Software, manejo de datos, etc.

- **Tipos de Negocio.** Ante la necesidad de crear nuevas aplicaciones específicas cuya vida esperada no pase del corto o mediano plazo, será más adecuado

realizarlo en ambientes de grupos de trabajo. La tecnología de las computadoras personales es más volátil que la de Host, por lo que no tendría sentido invertir en plataformas de vida esperada mucho mayor que el destino del programa mismo.

- **Diseño de Aplicaciones.** Las aplicaciones típicas de mainframe han sido diseñadas básicamente acoplando el giro comercial de las empresas: facturación, cuentas por pagar, sueldos, etc. Aplicaciones propias de grupos de trabajo, en cambio, se utilizan para propósitos específicos como planillas de cálculo, gráficos, procesamiento de textos. En aplicaciones Cliente/Servidor estos programas interactúan con los programas centrales definiendo aplicaciones mucho más provechosas.

- **Interface.** Aplicaciones que requieren una agradable interface gráfica para los usuarios finales, favorecen fuertemente el ambiente de grupos de trabajo. Muchas de las mayores características de las interfaces actuales, incluyendo gráficos o íconos, no están disponibles en los entornos de mainframes.

- **Area de la Empresa que necesitan nuevas aplicaciones.** Normalmente el usuario final tiene requerimientos con urgencia que no siempre el área de sistemas puede satisfacer con la rapidez requerida. Esta situación se orienta más a los ambientes de grupos de trabajo, mientras que el área de mainframe se dedica más fuertemente, a los problemas más globales, muchas veces generado en la misma área de sistemas.

1.6 Cliente/Servidor, la ventaja de negocio.

"La computación Cliente/Servidor está sustentada en el modelo de interacción usuario-máquina", dice Abby Pinard, vicepresidente de ventas de MUST Software. "La computación en el mainframe optima los recursos de la máquina a expensas del usuario, el cual es encerrado a tareas rígidas y procedurales. Actualmente, el recurso humano es el más caro, así que estamos construyendo herramientas cuyo principal objetivo sea el hacer al usuario más productivo."

Otra forma de ver a Cliente/Servidor es dado por Michel Loria, vicepresidente de Wang. "Cliente/Servidor ha sido difícil de definir porque incluye tanto tecnología nueva como existente. Este le da al usuario una nueva forma de hacer cosas viejas, pero al final solamente se trata de PCs corriendo Windows. Cliente/Servidor le da a los usuarios un control significativamente mayor y un acceso a una poderosa tecnología."

"Irónicamente,..." Mike Meli, presidente de Design Data Systems, comenta, "actualmente te sigues encontrando con definiciones si sentido de lo que realmente significa Cliente/Servidor. Muchos dicen que Cliente/Servidor significa tener una Interface Gráfica de Usuario (GUI) conectado a una PC y así extraer información de un mainframe. Si estos expertos estuvieran diseñando aplicaciones Cliente/Servidor o entendieran Cliente/Servidor, ellos conocerían mejor. Para mi, Cliente/Servidor significa el perfecto balance de distribución de la carga de procesamiento a ser desarrollado."

La mayoría de las definiciones de Cliente/Servidor incluyen este énfasis en "el poder donde lo requieras."

Travis White, director de comunicaciones corporativas en J.D. Edwards, lo pone en este sentido: "Cliente/Servidor es un intento de balancear las capacidades de procesamiento distribuido del host con la interactividad y amabilidad de las máquinas de escritorio. Cuando esto es bien hecho, Cliente/Servidor mezcla lo ágil de un carro deportivo en el escritorio con las capacidades de carga de un camión en el Servidor."

Meli de Design Data Systems añade, "Cliente/Servidor distribuye la carga de procesamiento entre el Servidor y muchos Clientes. En un ambiente centralizado en el host, el host hace todo el trabajo incluyendo el aceptar e interpretar las peticiones que se le mandan. En una arquitectura Cliente/Servidor, desde que los datos no son almacenados localmente, aplicaciones Cliente/Servidor deben mantener un balance delicado. El Cliente debe hacer todo lo que pueda hacer por si mismo antes de requerir los recursos del Servidor mientras este manteniendo ventaja sobre memoria y poder de CPU."

En los negocios modernos, donde hay múltiples necesidades de poder de cómputo y datos en diferentes localidades geográficas, es un ambiente que empieza por una solución Cliente/Servidor. Y no importando que no se llegue a un acuerdo en la definición de Cliente/Servidor, hay muchas herramientas allá afuera que pueden manejar la mayoría de los problemas de negocio que presionan.

1.6.1 Bajando (Downsizing) desde los Mainframes y mini-computadoras.

Muchas configuraciones Cliente/Servidor resultan de la inhabilidad de host centralizados o host de tamaño medio, de liberar datos y poder de cómputo eficientemente hacia centros de trabajo dispersados geográficamente, sin importar que sea el mismo edificio o a través del mundo. La comunidad MIS tradicional siente equivocadamente que esto significa el fallecimiento del host. Mucha de esta confusión estriba en el mal entendimiento de la arquitectura Cliente/Servidor.

Cliente/Servidor tampoco excluye la posibilidad de la computación de host, ni garantiza que los recursos computacionales serán ampliamente dispersados a través de la corporación. De hecho, muchas soluciones Cliente/Servidor permiten que el host quede como está, mientras se optimizan las comunicaciones del host y se compatibiliza con las terminales y los Servidores de red local.

Las casas que venden los mainframes deben considerar cuidadosamente donde están y a dónde quieren dirigirse. Dave Charlesworth, director de Andyne Computing, ve el proceso de decisión de esta manera. "¿Servirá mi elección de solución para este proyecto, para las necesidades de los proyectos que estarán ligados? Toma en cuenta que habrá una gran inversión en educar a la gente de soporte y a los usuarios finales de este proyecto. Como resultado, el primer proyecto establece inmediatamente una inercia para el conjunto de herramientas escogidas."

Charlesworth cree que la segunda consideración es, "Mi elección me dejará libre para remplazar un componente (ejemplo, una Base de Datos) sin causar una mayor reestructuración del resto de los componentes (ejemplo, las herramientas de consulta a las Bases de Datos)? Este es un aseguramiento de componentes."

La integración del mainframe dentro del ambiente Cliente/Servidor será diferente para cada negocio y presenta algunos aspectos interesantes. "Ciertamente la integración de los mainframes dentro del modelo Cliente/Servidor es un mayor reto y un mayor objetivo para muchos de nuestros grandes Clientes," indica Peter J. Tarrant, director de ventas de IBM North America. "Ahora mismo, la preponderancia de los datos residen en los mainframes, pero necesita ser aprovechable para el ambiente Cliente/Servidor. La mayor consideración para nuestros Clientes es el acceso a la información, la integridad y la seguridad de la integración, y la maleabilidad de los sistemas."

Progress Software provee soluciones de Software multiplataforma por desarrollar en las grandes empresas e implementa aplicaciones críticas para el negocio en sus esfuerzos por bajar de ambiente (downsizing).

Robert Kilcullen, director para Pennant Systems, la compañía de sistemas de impresión de IBM, nos recuerda no perderle la vista a los principios básicos del downsizing. "Los factores críticos para un exitoso downsizing de un ambiente de mainframe incluye la integridad de los datos, el Software para la recuperación de errores, y la habilidad para distribuir funcionalidad a través de ambientes Cliente/Servidor mientras se maximizan las inversiones en el Software actual."

Buenas soluciones de impresión son esenciales en los esfuerzos del downsizing, "A través de una combinación de las funciones del Hardware, microcódigo, y Software AFP," dice Kulicullen. "Las soluciones de impresión de Pennant

proporcionan una máxima flexibilidad a los usuarios finales de acuerdo a como ellos reconfiguren sus sistemas y provean la información que el administrador requiera.

1.6.2 Integrando las Aplicaciones Heredadas

Sin hacer caso de si una empresa está haciendo downsizing o upsizing a Cliente/Servidor, existen datos y aplicaciones críticas de negocio heredadas que se deben integrar a la nueva configuración. Antes de que cualquier cambio sea hecho, se necesita planear la transición con cuidado. Este esfuerzo está enmarcado dentro de un rango que va desde la adaptación menor a las interfaces para datos y aplicaciones hasta una conversión mayor del proyecto. Así como White de J.D. Edwards nos explica, la definición de Cliente/Servidor es quizá menos importante que una clara definición de los objetivos de Tecnología de Información: "Mucha gente parece creer que la computación centralizada en el host y la computación Cliente/Servidor son mutuamente excluyentes". Así como J.D. Edwards, creemos que en el mundo real se demandarán opciones entre estos dos polos. Estamos diseñando nuestro Software para que los usuarios puedan correrlo bajo un ambiente puramente centralizado en el host, un ambiente principalmente centralizado en Cliente/Servidor, o virtualmente una combinación de ambos. Esto reduce la presión de convertir inmediatamente las aplicaciones heredadas y provee un camino de migración más natural desde un paradigma al otro.

Meli, de Design Data Systems apunta algunas de las compatibilidades que pueden hacer la transición más sencilla: "Sin hacer caso del porqué una firma está migrando hacia Cliente/Servidor, hay siempre una conversión necesaria para integrar las aplicaciones existentes dentro de un nuevo ambiente".

"El integrar los datos y sistemas heredados dentro de ambientes Cliente/Servidor es uno de los mayores retos al cual se enfrentan las organizaciones," asegura Abby Pinard de MUST Software. "Claramente, algunas aplicaciones son más fáciles de migrar que otras -soporte de decisiones, aplicaciones para grupos de trabajo y departamentales, por ejemplo. Las aplicaciones más difíciles -grandes Bases de Datos, altos volúmenes de procesamiento de transacciones en línea- permanecerán por más tiempo dentro del mainframe, hasta que las herramientas para el desarrollo de aplicaciones, monitores de transacciones, y facilidades para el manejo de sistemas sean lo suficientemente robustas para soportarlo."

Una complicación adicional en la migración hacia Cliente/Servidor son los estándares. Lo variado de los componentes de Software y de Hardware en las configuraciones de Cliente/Servidor invariablemente da surgimiento a un arreglo de estándares y estándares de facto. Esto puede ser un problema cuando se tiene

que hacer un compromiso para una plataforma que obligue a un estándar específico.

"Cliente/Servidor generará un sin número de estándares de cómputo que confundirán temporalmente al mercado," comenta White de J.D. Edwards. "Ultimamente el mercado se moverá a los estándares de facto los cuales no son caros, relativamente fáciles de usar, y están ampliamente disponibles. Micrososft tiene un buen registro ofreciendo justamente esta clase de Software, así que es justo apostar que ellos generarán los estándares que actualmente están usando."

"Existen dos clases de estándares que tienen que ver con Cliente/Servidor", apunta Senor de Information Builders. "Uno es el estándar de Llamadas a Procedimientos Remotos (RPC -Remote Procedure Call) brindado por DCE, el cual está creciendo en importancia. El otro estándar es la Llamada a Nivel de Interface (CLI -Call Level Interface) y hay, en esencia, tres estándares importantes para CLI : los API/SQL de EDA/SQL, el ODBC de Microsoft, y la Llamada a nivel Interface X/OPEN. Estos estándares que se están desarrollando minimizarán confusiones, porque solo habrá un número de interfaces aceptables que estén disponibles.

Comentando de los estándares de Oracle, Meli de design Data Systems dice, "Oracle ha tomado un gran paso hacia el desarrollo de los estándares e interfaces para Cliente/Servidor. Con la versión de Oracle 6, los procedimientos almacenados, los eventos manejados por gatillos, y el estándar usado de SQL, formará una base industrial en donde los estándares aceptados puedan ser aplicados.

1.6.3 Herramientas y Aplicaciones

Las herramientas necesarias para el desarrollo de aplicaciones Cliente/Servidor tendrán que ser tan únicas como las herramientas artesanales.

Existe una tendencia de los desarrolladores de Software de intentar modificar las herramientas actuales. Mientras esto pueda funcionar en ciertas instancias, White de J.D. Edwards, nos previene en contra de esta aproximación. "Algunos ingenieros de SW están tratando de readaptar las herramientas que están en el host centralizado para que puedan servir como herramientas de Cliente/Servidor. En J.D. Edwards creemos que el nuevo paradigma requiere enteramente nuevas herramientas que incorporen al modelo Cliente/Servidor desde abajo. Por ejemplo, muchas de las herramientas actuales generan código procedural; lo que quiere decir, que el Software siempre sigue el mismo procedimiento incluso si el modelo de negocio cambia. Creemos que el modelo Cliente/Servidor requiere de Software orientado a eventos, el cual los usuarios finales puedan fácilmente

modificar para continuar con el negocio. Para crear herramientas orientadas a eventos, tienes que romper la forma de pensar que hasta ahora has heredado".

Algunos desarrolladores piensan que están listos para hacer este rompimiento. Tarrant de IBM North America mira al desarrollo de aplicaciones de la siguiente manera. "Existen tres cambios substanciales para fabricar aplicaciones Cliente/Servidor viables: la habilidad para desarrollar rápidamente las aplicaciones, la habilidad para desplegar aplicaciones rápidamente, y la flexibilidad para modificar fácilmente conforme las necesidades del negocio cambien y sean integradas al ambiente actual. Estos cambios están siendo direccionados actualmente con un número de vendedores de aplicaciones, los cuales están proveyendo de herramientas de rápido desarrollo de aplicaciones."

El truco en el desarrollo de aplicaciones Cliente/Servidor es el construir estructuras que duren hasta el final, anticipando los cambios que inevitablemente ocurrirán. Meli, de Design Data Systems, cree que el anticiparse a la forma y dirección del cambio tecnológico es importantísimo. "Debido al rápido cambio en el desarrollo tecnológico, el mayor reto actualmente es predecir el futuro, así que el trabajo y aplicaciones que escribas hoy pueden ser usadas en las tecnologías del mañana".

Un gran número de compañías ofrecen una extensa variedad de herramientas para la computación Cliente/Servidor. Andyne Computing ofrece algunas herramientas avanzadas para acceso a Bases de Datos en plataforma cruzada que ayudarán a la empresa a prepararse para el futuro. Las comunicaciones son importantes en cualquier configuración Cliente/Servidor. Por ejemplo, FTP ofrece una gran variedad de soluciones de comunicación bajo TCP/IP. IBM también ofrece una gran selección de Hardware y Software para el AS/400, los sistemas RISC 6000, y los sistemas S/390.

CAPITULO 2. ELECCION DE LA PLATAFORMA CLIENTE/SERVIDOR.

2.1 El Modelo de Arquitectura Cliente/Servidor.

La arquitectura Cliente/Servidor elimina la necesidad de mover grandes bloques de información sobre la red para que se procese en la PC. El Servidor controla estos bloques y procesa los requerimientos; entonces transfiere solo la información pertinente a la PC Cliente, y es presentada al usuario final de una manera más fácil de entender.

2.1.1 Anatomía de un Programa Servidor.

El papel de un programa Servidor es el servir a múltiples Clientes quienes tengan interés sobre un recurso compartido perteneciente a él.

A continuación se presenta lo que típicamente realiza un programa Servidor:

- Espera que el Cliente inicie las peticiones: El programa Servidor pasa la mayor parte de su tiempo esperando pasivamente a que arriben las peticiones del Cliente, en forma de mensajes, a través de la sesión de comunicación. Algunos Servidores asignan una sesión dedicada a cada Cliente. Otros crean dinámicamente conjuntos (POOLS) de sesiones reusables. Algunos otros proveen una mezcla de los dos ambientes. Para tener éxito, el Servidor debe siempre responder a sus Clientes y estar preparado para el intenso tráfico cuando muchos Clientes solicitan servicios al mismo tiempo.

- Ejecuta muchas peticiones al mismo tiempo: El programa Servidor debe ser capaz de servir concurrentemente a múltiples Clientes, al mismo tiempo que protege la integridad de los recursos compartidos.

- Toma cuidado de las prioridades: El programa Servidor debe ser capaz de proveer diferentes niveles de prioridad a sus Clientes. Por ejemplo, un Servidor puede servir una petición para un reporte o un trabajo batch con una prioridad menor mientras mantiene la prioridad alta para los Clientes del tipo de procesamiento con transacciones en línea.

- Inicializa y corre tareas en el trasfondo (backend): El Servidor debe de ser capaz de correr tareas en el trasfondo las cuales no están relacionadas con la cometida del programa principal.

- Debe mantenerse corriendo: El programa Servidor es una aplicación típicamente sensible al negocio. Si el Servidor se cae éste impacta a todos los Clientes que dependen de sus servicios. El programa Servidor y el ambiente en el cual corre debe ser muy robusto.

- Crece grande y robusto: El programa Servidor aparenta tener un apetito insaciable por memoria y poder de procesamiento. El programa Servidor debe ser escalable y modular.

2.1.2 Análisis de los requerimientos del Servidor hacia el Sistema Operativo.

En ambientes de cómputo distribuido, las funciones del Sistema Operativo llegan a ser servicios base y extendidos. Los servicios base son parte del Sistema Operativo estándar, mientras que los servicios extendidos son agregados sobre componentes modulares de Software los cuales son colocados por encima de los servicios base. Funcionalmente es equivalente que los servicios extendidos sean provistos por más de un vendedor. No hay una regla estricta que marque cuales son las funciones que provee el Sistema Operativo base y cuales debe proveer el extendido. Actualmente las extensiones son buenos candidatos para formar parte dentro del Sistema Operativo base del futuro.

Servicios Base.

Debe ser evidente, dada la descripción del párrafo anterior, que los programas Servidor exhiben un alto nivel de concurrencia. Idealmente, una tarea separada será asignada a cada uno de los Clientes, a los cuales el Servidor esta diseñado para soportar en forma concurrente. El manejo de tareas es excelentemente hecho por un Sistema Operativo multitarea. La multitarea es una forma natural de simplificar la codificación de aplicaciones complejas que pueden ser divididas en una colección de, discretas y lógicamente distintas, tareas concurrentes. Mejora el rendimiento, modularidad, y responsabilidades de los programas Servidor. Multitarea también implica la existencia de mecanismos para la coordinación de tareas interrelacionadas e intercambios de información.

Los Servidores también requieren un alto nivel de concurrencia dentro de un solo programa. El código Servidor correrá más eficientemente si las tareas son alojadas en partes del mismo programa en vez de separarlas del programa (estas tareas son llamadas corrutinas o threads). Las tareas dentro del mismo programa son más rápidas de crear, fáciles de cambiar en contexto, y tienen un acceso más fácil a la información compartida. A continuación se describen los requerimientos del Servidor.

- **Tarea Preadquirida (Task preemtion).** Un Sistema Operativo con multitarea preadquirida (preemptive multitasking) debe asignar una hendidura fija de tiempo de ejecución para cada tarea. Sin tarea preadquirida, una tarea debe estar de acuerdo voluntariamente a ceder el procesador antes de que otra tarea pueda correr. Es mucho más seguro y fácil escribir programas Servidor en ambientes donde el Sistema Operativo maneja automáticamente todos los cambios de tareas.

- **Prioridad de tareas.** Un Sistema Operativo debe despachar a las tareas de acuerdo a su prioridad. Esta característica permite diferenciar los niveles de servicio basado en las prioridades de sus Clientes.

- **Semáforos.** Un Sistema Operativo debe proporcionar mecanismos simples de sincronización para mantener la concurrencia de las tareas y así evitar choques entre una y otra cuando accesan recursos compartidos. Estos mecanismos, conocidos como semáforos, son usados para sincronizar las acciones de tareas independientes del Servidor y alertarlas cuando algún evento significativo ocurre.

- **Comunicaciones Interprocesos (IPC).** Un Sistema Operativo debe proveer los mecanismos que permitan que procesos independientes puedan intercambiar y compartir datos.

- **Comunicaciones, Locales y Remotas, entre procesos.** Un Sistema Operativo debe permitir la redirección transparente de llamadas a procesos remotos sobre una red sin que la aplicación esté enterada de ello. La extensión de la comunicación interprocesos a través de los límites de la máquina es la llave para el desarrollo de aplicaciones donde los recursos y los procesos pueden ser fácilmente movidos a través de las máquinas.

- **Corrutinas (Threads).** Estas son unidades de concurrencia provistas desde un programa por si mismo. Las corrutinas son usadas para crear programas en el Servidor los cuales son manejados por eventos muy concurrentes. Cada evento en espera puede ser asignado a una corrutina que bloquee hasta que el evento ocurra. Mientras tanto, otras corrutinas pueden usar los ciclos de reloj del CPU aumentando la productividad y mejorando el tiempo de trabajo.

- Protección entre tareas. El Sistema Operativo debe proteger a las tareas de interferir con los recursos pertenecientes a otras tareas. Una sola tarea no debe ser capaz de llevar abajo el sistema por completo. La protección también se extiende al sistema de archivos y llamadas al Sistema Operativo.

- Sistema de archivos de alto rendimiento para varios usuarios. El sistema de archivos debe soportar a múltiples tareas y proveer los bloqueos que protejan la integridad de los datos. Los programas Servidor típicamente trabajan con muchos archivos al mismo tiempo. El sistema de archivos debe soportar un gran número de archivos abiertos sin que se deteriore en gran medida el rendimiento.

- Uso eficiente de la memoria. El Sistema Operativo debe soportar eficientemente muchos programas de gran tamaño, al igual que grandes objetos de datos. Estos programas y éstos objetos de datos deben ser fácilmente llevados de memoria a disco y viceversa en pequeños bloques granulares.

- Ligar dinámicamente extensiones en tiempo de ejecución (Dynamically Linked Runtime Extensions). Los servicios del Sistema Operativo deben ser extensibles. Un mecanismo debe ser provisto para permitirle a los servicios tener crecimientos en tiempo de ejecución sin recompilar al Sistema Operativo.

Servicios Extendidos.

Los Servicios Extendidos deben proveer el software avanzado del sistema el cual explote el potencial distribuido de las redes, brindando un acceso flexible a la información compartida y haciendo del sistema más fácil de usar y mantener. También debe hacer fácil, para los vendedores de terceros e integradores de sistemas, el crear nuevas aplicaciones servidoras.

Algunos puntos que se deben esperar formen parte de los Servicios Extendidos serían los siguientes:

- Comunicaciones ubicuas. Las extensiones del Sistema Operativo deben proveer un rico conjunto de protocolos de comunicación que permitan al Servidor comunicarse con un gran número de plataformas Clientes. En suma, el Servidor debe ser capaz de comunicarse con otras plataformas de Servidores en caso de que necesite asistencia para proveer servicios.

- Extensiones de Red del Sistema Operativo. Las extensiones del Sistema Operativo deben proveer facilidades para extender los servicios de archivos e impresión sobre la red. Idealmente, las aplicaciones deben ser capaces de

accesar dispositivos remotos (como impresoras y archivos) de una manera transparente, como si lo hicieran localmente.

- Servicios de Base de Datos y de Transacciones. Las extensiones del Sistema Operativo deben proveer un robusto sistema de manejo de Base de Datos multiusuario. Este manejador de Base de Datos debe soportar, idealmente, el lenguaje SQL para el soporte de decisiones y procedimientos almacenados en el Servidor para los servicios de transacciones. Los procedimientos almacenados en el Servidor son creados fuera del Sistema Operativo, esto a través de programadores. Funciones más avanzadas incluyen monitores de transacciones para el manejo de procedimientos almacenados (o transacciones) como si fueran unidades atómicas de trabajo las cuales son ejecutadas en uno o más Servidores.

- Sección amarilla de red. Las extensiones del Sistema Operativo deben proveer un medio para que los Clientes puedan localizar a los Servidores, así como a sus servicios, dentro de la red; esto a través de una sección amarilla o directorio de servicios. Los recursos de la red deben ser encontrados por nombre. Los Servidores deben ser capaces de registrar dinámicamente sus servicios con el proveedor del directorio.

- Autenticidad y autorización de servicios. Las extensiones del Sistema Operativo deben proveer una forma para que los Clientes prueben que ellos son quienes dicen ser. El sistema de autorización determina si el Cliente tiene los permisos para obtener un servicio remoto.

- Manejo del Sistema. Las extensiones del Sistema Operativo deben proveer una red integrada y una plataforma de manejo del sistema. El sistema debe ser manejado como un solo Servidor o múltiples Servidores asignados a dominios. Una mirada global que cubra múltiples dominios debe ser provista para Servidores que jueguen en las ligas mayores. El manejo del sistema incluye servicios de configuración del sistema, facilidades para monitorear el rendimiento de los elementos, generando alertas cuando algo esté mal, distribuyendo y manejando paquetes de software en estaciones de trabajo, checando existencias de virus e intrusos, y midiendo las aptitudes de paga-mientras-uses los recursos del Servidor.

- Tiempo de red. Las extensiones del Sistema Operativo deben proveer un mecanismo para que los Clientes y Servidores puedan sincronizar sus relojes. Este tiempo debe ser coordinado por una autoridad universal de tiempo.

- Servicios Orientados a Objetos. Esta es un área en donde los Servicios Extendidos florecerán en el futuro que se avecina. Los servicios se están convirtiendo cada vez más orientados a objetos. El Sistema Operativo requerirá proveer servicios de "agente de objetos" que permitan a cualquier objeto

interactuar con cualquier otro objeto de la red. El Sistema Operativo tendrá también que proveer los servicios de intercambio de objetos y de repositorio de objetos. Las aplicaciones Cliente/Servidor del futuro estarán comunicando objetos. Los grupos de objetos vendrán juntos en asociaciones sueltas para proveer servicios.

- Grandes Objetos en formato Binario. Imágenes, video, gráficas, documentos inteligentes, y fotos en Base de Datos probarán las capacidades de los Sistemas Operativos, Base de Datos y Redes. Estos grandes objetos en formato binario requieren extensiones del Sistema Operativo como por ejemplo manejadores de corrientes de mensajes y formatos de representación. Las redes deberán estar preparadas para transportar estos grandes objetos de formatos binarios y proveer accesos a los mismos. Los protocolos serán necesarios para su intercambio a través del sistema y para asociar estos grandes objetos a programas que sepan qué hacer cuando los vean.

2.1.3 Anatomía de un Programa Cliente.

Todas las aplicaciones Clientes tienen ésto en común: ellas requieren servicios del Servidor. Lo que las hace diferentes es qué activa las peticiones y qué interface gráfica de usuario, si hay, es la requerida. Basado en éstas diferencias, podemos clasificar a los Clientes en tres categorías: Clientes que no utilicen interfaces de usuario gráficas, Clientes que sí las utilicen, y Clientes con interfaces de usuario orientadas a objetos.

Clientes que no utilizan interfaces de usuario gráficas.

Las aplicaciones Clientes que no utilizan interfaces de usuario gráficas generan peticiones al Servidor con una cantidad *mínima* de interacción humana. Estos Clientes se pueden dividir en dos subcategorías:

1. Clientes que no necesitan multitarea: Ej. lectoras de códigos de barras, teléfonos celulares, faxes, bombas de gasolinera inteligentes, etc. Estos Clientes pueden proveer una sola interface humana en el ciclo de generación de petición.

2. Clientes que sí necesitan multitarea. Ej. Robots, programas demonios, etc. Estos programas seguido requieren servicios manejadores de eventos multitarea, de tiempo real y granulares.

Clientes que utilizan interfaces de usuario gráficas.

Clientes simples que utilizan interfaces de usuario gráficas son las aplicaciones cuyas peticiones ocasionales al Servidor resultan de la interacción de un humano con una interface de usuario gráfica. Estas interfaces son excelentes para aplicaciones de negocio con procesamiento de transacciones en línea con altos volúmenes de tareas repetitivas. Estas interfaces también son muy buenas como front-end para Clientes de Servidores de Base de Datos. Las aplicaciones simples que utilizan este tipo de interfaces son traducciones gráficas de los diálogos que se corrían sobre las terminales tontas. Las interfaces de usuario gráficas reemplazan las feas terminales verdes por cajas de diálogo gráficas, colores, barras de menús, y ventanas. Los diálogos simples usan el modelo objeto/acción en donde los usuarios pueden seleccionar objetos que representan las acciones que ellos quieren que se lleven al cabo. La mayoría de los diálogos son seriales por naturaleza. Este modelo de interacción de usuario es usado predominantemente en Windows 3.x, OS/2 1.x, y aplicaciones OSF Motif. Es también conocido como modelo gráfico CUA 89.

Clientes que utilizan interfaces de usuario orientadas a objetos.

La metáfora de la interface de usuario orientada a objetos es usada para proveer lo que el presidente general de Microsoft, Bill Gates, llama "información al alcance de la mano". Esta interface de "Cliente Universal" es altamente iconizada, interfaces orientadas a objetos que proveen el mismo acceso a la información en diferentes formatos. La interface de usuario orientada a objetos será usada por trabajadores de la información haciendo múltiples y variadas tareas cuya secuencia no puede ser predicha. Los ejemplos incluyen aplicaciones para la toma de decisiones, sistemas de entrenamiento basados en multimedia, consolas de operación, etc. Las interfaces de usuario orientadas a objetos tienen un insaciable apetito por comunicación. Los objetos en el escritorio necesitan comunicarse entre ellos y con otros Servidores. Las comunicaciones son, por necesidad, en tiempo real, interactivas, y altamente concurrentes.

2.1.4 Análisis de los requerimientos del Cliente hacia el Sistema Operativo.

Cada uno de los tres tipos de Clientes descritos anteriormente tiene un conjunto diferente de requerimientos del Sistema Operativo. En seguida se presenta una lista de éstos.

Requerimientos del Sistema Operativo	Clientes con Interfaces de Usuario...			
	No Gráficas		Gráficas	Orientadas a Objetos
	Con Multitarea	Sin Multitarea		
Mecanismos de Petición Respuesta (preferentemente con transparencia local/remota)	SI	SI	SI	SI
Mecanismos de transferencia de archivos para mover imágenes, texto, y fotos almacenadas en Base de Datos.	SI	SI	SI	SI
Multitareas preferentes	NO	SI	DESEABLE	SI
Prioridad de tareas	NO	SI	DESEABLE	SI
Comunicación entre procesos	NO	SI	DESEABLE	SI
Threads corriendo en trasfondo para comunicación con Servidores	NO	SI	SI	SI
Sistema Operativo robusto, incluyendo protección entre tareas y llamadas reentrantes al Sistema Operativo	NO	SI	DESEABLE	SI
Interfaces gráficas de usuario para Windows 3.x	NO	NO	SI	SI
Interfaces de usuario Gráficas con diálogos paralelos e interactivos, arrastrar y soltar, soporte multimedia.	NO	NO	NO	SI

Como se puede apreciar, todas las aplicaciones Cliente necesitan mecanismos para comunicar peticiones y archivos al Servidor. Las tres categorías de Clientes funcionarán mejor en ambiente robusto y multitarea. Es particularmente importante para el Cliente el ser robusto porque es imposible para los proveedores de sistemas probar el software Cliente en todas las combinaciones de hardware y software. Es importante el utilizar un Sistema Operativo que pueda proteger programas de que choquen y choquen unos con otros. Ningún programa Cliente debe causar el que se tenga que reinicializar el sistema (apagar y volver a encender o bootear).

Las interfaces de usuario gráficas y las orientadas a objetos trabajan mejor con mecanismos que manejen las peticiones de trasfondo. Si se usan corrutinas separadas para la interface de usuario y para el procesamiento de trasfondo, entonces el programa puede responder a las entradas del usuario mientras corrutinas separadas manejan la interacción con el Servidor. Así es como las interfaces de usuario gráficas evitan el icono indicador de procesamiento (reloj), un claro signo de que la máquina no continúa con el usuario. Multitarea

preadquirida también es necesaria para responder a los dispositivos multimedia y crear aplicaciones Clientes mientras muchos diálogos son desplegados en paralelo.

2.2 Descripción de las alternativas que nos ofrece el mercado.

Escoger una plataforma Cliente/Servidor no es una tarea fácil. Nuevos Sistemas Operativos para el escritorio están brotando, mientras los Sistemas Operativos viejos se están fragmentando en diversas variantes. Estas no son necesariamente malas noticias para el modelo de arquitectura Cliente/Servidor, el cual prospera gracias a la diversidad. Sin embargo, pueden ser malas noticias si se esta tratando de desarrollar Software bajo este modelo pero utilizándo la plataforma Cliente/Servidor equivocada.

Para escoger una plataforma de Sistema Operativo, primero hay que decidir en que cultura computacional se jugará. Las compañías líderes del mercado proveen canales de distribución, una infraestructura de soporte, y una gran base instalada de usuarios que son Clientes potenciales para sus productos. A continuación se presenta a los líderes contendientes:

- El mundo de la PC, el cual incluye todos los tipos de máquinas basadas en el procesador Intel 80X86 las cuales corren DOS (y sus variantes), Windows 3.X y NT, OS/2 y Netware. Las PC son obicuas en el mundo de los negocios.

- El mundo Macintosh, el cual esta fuertemente atrincherado en ciertos sectores del mundo de los negocios (entre escritores, ilustradores, diseñadores, y departamentos de ventas).

- El mundo Unix, en todas sus variantes y plataformas de hardware; es el mundo de las estaciones de trabajo y de usuarios poderosos.

- El mundo de la supermini, el cual es dominado por el hardware de DEC y soluciones basadas en VMS.

- El mundo de las mainframes, el cual es dominado por los mainframes IBM y sus estándares SAA (System Aplication Architecture). Partes del SAA estan siendo adoptados como estándares de facto por muchos fabricantes de sistemas no IBM (como las computadoras Tandem). OS/2 y OS/400 son miembros no-mainframe de la familia SAA.

2.3 *Elección de la mejor opción.*

La solución planteada para esta Tesis está basada para correr sobre el mundo PC, el cual es el más grande del mercado, con una base instalada de 60 millones de máquinas, creciendo a una tasa de 20 millones por año.

Una plataforma Cliente/Servidor corriendo en PC debe cumplir, lo mejor posible, la mayor parte de los requerimientos que se han planteado a lo largo de este capítulo. La plataforma PC debe proveer un Sistema Operativo robusto el cual soporte multitarea, una máquina de Base de Datos multiusuario, y comunicaciones en red LAN. El Servidor debe ser capaz de manejar, concurrentemente, peticiones de los Clientes con degradaciones mínimas en su rendimiento. El Cliente debe ser robusto, capaz de correr tareas concurrentes, y proveer una gran interfaz gráfica para los usuarios. Las soluciones en Cliente/Servidor para las PC deben proveer el mismo nivel de servicio como las superminis multiusuario mientras mantienen la autonomía de los Clientes PC.

El dominante ambiente MS-DOS en el escritorio es tecnológicamente empobrecido. Construir una arquitectura Cliente/Servidor en MS-DOS es como construir una casa en arena. Windows versión 3 de Microsoft no resuelve el problema porque reposa (relies) sobre el MS-DOS. Aquí es donde el OS/2 entra en la pantalla:

- Los Servidores OS/2 proveen el poder de una minicomputadora al precio de las PC. El OS/2 de 32 bits corriendo en una máquina con procesador 80486 tiene el poder de una minicomputadora, ambos en términos de velocidades brutas de procesador y sofisticaciones de software de sistema a través de características como multitarea preferente, multithreading, y construido en protección de memoria.

- OS/2 soporta un conjunto enriquecido de plataformas para Servidores de Base de Datos y Comunicaciones los cuales hacen posibles crear sistemas de "aplicaciones críticas" usando computadoras personales.
- OS/2 es una plataforma comercial "abierta", lo cual significa que se pueden escoger productos y servicios de una amplia gama de proveedores a precios competitivos. Esto se traduce en ahorros enormes en los costos de sistemas de Hardware y Software, especialmente cuando se compara con una solución, propietaria, equivalente para una supermini.

- OS/2 provee una plataforma completa y robusta. Permite que múltiples aplicaciones corran concurrentemente mientras provee la protección necesaria para que no se interfieran unas con otras.

- OS/2 provee un ambiente de integración para correr aplicaciones DOS, Windows y OS/2. El Workplace Shell hace que el escritorio se parezca al que utiliza el usuario final en sus oficinas; también soporta el "cortar y pegar" entre aplicaciones DOS, Windows y OS/2. La máquina de 32 bits del Presentations Manager provee servicios de Interface Gráficas para el Usuario (GUI - Graphical User Interface)

- Los usuarios en OS/2 pueden manipular recursos de red, como archivos e impresoras, simplemente apretando los botones del ratón sobre los objetos gráficos que representan a elementos de la red, y así llevar al cabo cualquier acción que se requiera. El ambiente local y de red tienen la misma apariencia. De una forma común, el Workplace Shell soporta tanto Netware de Novell como Lan Server de IBM; se espera que Banyan's Vines se una al club pronto.

- OS/2 elimina la necesidad para las minicomputadoras de actuar de intermediario entre los Clientes PC y los hosts.

En pocas palabras, el OS/2 esta completamente cargado con lo necesario para diseñar aplicaciones Cliente/Servidor alrededor de las computadoras personales.

2.3.1 Anatomía de un programa PM (Presentation Manager).

Los programas en PM son, en su mayoría, reactivos. Pasan la mayor parte del tiempo en un estado ocioso, en espera de eventos que pasen. Cuando un evento como el presionar de una tecla o un clic del ratón ocurre, el PM rutea la entrada al procedimiento apropiado, en forma de mensaje. El procedimiento puede entonces escoger entre reaccionar al evento o solo regresarlo al PM para que él lo maneje. Si el procedimiento necesita desplegar información en una ventana, este llama al PM para que lo haga. Después de hacer la respuesta adecuada, el procedimiento regresa al estado ocioso en espera del siguiente evento. Entonces, la primera cosa que podemos decir acerca de un programa en PM es que él hace una colección de procedimientos orientados por eventos que esperan a ser invocados para llevar al cabo el trabajo.

El corazón del PM: "Todo es una ventana"

El PM asocia procedimientos con eventos a través de la "magia" de las ventanas. Las ventanas en el PM no son más que los típicos rectángulos que se sobreenciman en la pantalla. Una ventana es un "objeto". Se convierte en "un objeto que sabe que hacer" cuando trabaja en conjunto con un procedimiento de eventos. Un programa

PM es conceptualmente un número de ventanas y algo que hace que ellas sepan que hacer.

Algunas de las ventanas aparentan la misma forma que las familiares áreas rectangulares, las cuales son típicamente llamadas "ventanas". Por otro lado, estructuras que no son ventanas - como barras de deslizamiento, botones y menús - también son considerados como ventanas dentro del PM. Una ventana pude ser visible o no en la pantalla. PM provee un tipo de ventana, llamado el objeto de ventana, el cual no tiene nada que ver con los componentes visuales. Las aplicaciones más tradicionales de PM, sin embargo, crean al menos una ventana "tradicional" llamada la "ventana principal", la cual es generalmente visible y representa a la aplicación al mundo exterior.

El cerebro de una ventana: Clases y procedimientos.

El elemento "inteligente" que define el comportamiento de una ventana es la Clase de la Ventana. Este es el elemento que "sabe que hacer". Cada ventana es una instancia de una clase de ventana, la cual determina el estilo de la ventana y especifica el procedimiento manejador que hará el manejo de los mensajes por la ventana. En la terminología de PM, estos procedimientos manejadores de eventos se conocen como procedimientos de ventana. Los procedimientos de ventana son el corazón de la programación de PM. En cada clase de ventana tiene un procedimiento de ventana el cual es responsable por todos los aspectos de procesamiento de una clase de ventana, incluyendo: cómo aparece la ventana, cómo se comporta y responde a los cambios de estado y cómo maneja la entrada de información del usuario.

Toda la interacción con la ventana es pasada como mensaje hacia su procedimiento de ventana. Cuando una aplicación crea una ventana, ésta debe especificar su clase. La "clase" le especifica al PM cual procedimiento de ventana obtiene para procesar los mensajes generados por la actividad de las ventanas. En términos de orientación a objetos, el procedimiento de ventana contiene todos los métodos que controlan a la ventana. El procedimiento de ventana decide qué método invocar basado en el mensaje que recibe.

El sistema nervioso: Mensajes del PM

Si las ventanas son el corazón del PM y los procedimientos son su cerebro, entonces los mensajes son su sistema nervioso. Típicamente, docenas de mensajes estarían circulando alrededor del sistema de PM. Un mensaje puede contener información, una petición por información, o una petición para una acción. En PM,

los mensajes son generalmente generados cuando un usuario interactúa con una ventana en la pantalla, cuando una ventana necesita comunicarse con otra ventana, o cuando el sistema detecta un evento como el mover o cambiar de tamaño a una ventana. Las aplicaciones pueden usar los mensajes del sistema del PM para comunicarse unas con otras; es una manera muy manejable de crear aplicaciones manejadas por eventos, de propósito general.

Aunque el OS/2 sea un Sistema Operativo Multitarea, las aplicaciones deben compartir el teclado de la computadora, así como el ratón y pantalla. OS/2 coloca las entradas de los eventos en una sola cola de sistema. El ruteador convierte las entradas del sistema en mensajes para las aplicaciones que controlan la ventana donde el evento fué generado.

¿Qué debe hacer un programador de PM?

Una aplicación PM hará, mínimo, la definición de una nueva ventana y creará un procedimiento para ella. Los programas PM comerciales típicamente consisten de cientos de procedimientos de ventanas desarrollados para cumplir con los requerimientos de la aplicación.

Para cada ventana, el programador tendrá que hacer:

- Definir los atributos de la ventana.
- Identificar los mensajes de entrada que el procedimiento procesará.
- Escribir el procedimiento de la ventana que maneje las respuestas a los mensajes de entrada que sean de interés.
- Documentar los mensajes de salida creados por el procedimiento de la ventana.

En pocas palabras, un programa PM consiste en un procedimiento principal que inicializa el ambiente de PM y uno o más procedimientos de ventanas que actúan en todas las cosas que les interesen, el resto es dejado para el manejo del OS/2.

Anatomía de un programa PM.

Un programa PM consiste de un procedimiento principal "main()" y un conjunto de procedimientos de ventanas. El procedimiento principal aloja los datos globales, hace la inicialización, y crea los objetos PM. Después entra en un ciclo de espera de mensajes, los cuales son depositados en su línea de espera. Cuando encuentra un mensaje, este lo manda hacia el procedimiento de ventana adecuado. Cuando un mensaje especial es recibido (WM_QUIT), el procedimiento principal limpia todo antes de terminar el programa (a sí mismo).

A continuación se presenta la estructura de un programa PM:

1. Conexión a PM
2. Carga de recursos.
3. Creación de la cola de mensajes.
4. Registro de las clases de ventanas.
5. Creación y desplegado de la(s) ventana(s) principales.
6. Creación y despliegue de la(s) ventana(s) hija(s).
7. Ciclo de despacho de mensajes.
8. Obtención de mensajes.
9. Despacho de mensajes.
10. Destrucción de ventanas.
11. Destrucción de la cola de mensajes.
12. Desconexión del PM.

CAPITULO 3. ANALISIS DEL SISTEMA.

3.1 Análisis del negocio hotelero.

3.1.1 Establecimientos de hospedaje en el mundo.

Es difícil precisar la fecha en que surgió el primer establecimiento dedicado a proporcionar el servicio de hospedaje. Sin embargo, puede decirse sin temor a equivocaciones, que su antigüedad es muy similar a la del hombre mismo. Impulsado por su instinto de conservación, el hombre se veía en la necesidad de viajar constantemente y, en ocasiones, no encontraba un lugar adecuado donde pasar la noche, ni personas conocidas a quien pedir refugio. Así nació en los pueblos primitivos la costumbre de dar albergue gratuito al viajero por una noche, continuando éste su viaje al día siguiente.

La necesidad de crear establecimientos dedicados exclusivamente a la prestación del servicio de hospedaje fue surgiendo a medida que se fueron desarrollando las relaciones sociales entre los individuos. Primero, entre tribus y clanes; después, entre pueblos, regiones y ciudades; por último, entre distintos países. En oriente por ejemplo, surgieron la *caravanera* y el *khan*, ubicados en lugares de descanso especialmente propicios para que señores y esclavos, al viajar por el desierto, pudieran detenerse en busca de reposo y meditación.

Con la aparición de la moneda se impulsó notablemente el comercio, y en consecuencia, el número y la duración de los viajes. Muchos comerciantes, al ampliar el radio de acción, ya no podían regresar a sus casas el mismo día, viéndose en la necesidad de pasar la noche en los establecimientos de hospedaje. Estos sitios empezaron a recibir cada vez un mayor número de personas con dinero que no titubeaban en gastarlo a cambio de una buena cena y de una no menos buena cama, lo que motivó también que se abrieran más y más establecimientos de este tipo.

En la antigua Roma encontramos un tipo de hospedaje denominado *posta* y destinado a albergar tanto a civiles como a militares que regresaban de sus incursiones por los dominios del Imperio. Ya en aquella época y como una muestra más de su capacidad administrativa, los romanos crearon un sistema de hospedaje dentro de una estructura turística tal, que permitía a sus ciudadanos el viajar no solo por motivos bélicos, religiosos o de comercio, sino también por

simple esparcimiento. Se construyeron caminos empedrados, alojamientos de varios tipos y centros de recreo en las playas, lográndose de esta manera hacer más cómodos los viajes con los *carruca dormitoria*, carruajes especialmente adaptados para que pudieran dormir en ellos los pasajeros.

3.1.2 *Establecimientos de hospedaje en México.*

En el México primitivo, como en otros pueblos de la antigüedad, no existían lugares públicos donde alojarse. La hospitalidad era considerada un deber sagrado, mezclado con el temor supersticioso hacia los extranjeros, quienes tal vez fueran dioses peregrinos dispuestos a repartir el bien o el mal, según la acogida que se les dispensase. Ello explica el recibimiento que tuvo Hernán Cortés a su llegada a la Gran Tenochtitlán.

Sin embargo, hacia el año 1500 d. de J. C. y como consecuencia de la gran cantidad de viajeros que se dirigían periódicamente a la ciudad con el deseo de asistir a las ceremonias religiosas o bien para realizar intercambios comerciales, surgieron los primeros establecimientos de hospedaje en México, denominados por los aztecas *Coacallis*. Se trataba de construcciones de un solo piso situadas normalmente cerca de los mercados o a la entrada de las poblaciones, con acceso directo desde la calle o bien a través de un embarcadero privado. Tenían un patio central, con un altar dedicado a *Yacatecutli*, "Señor que guía", y a su alrededor se distribuían las habitaciones para los viajeros y las crujías destinadas a guardar sus mercancías y pertenencias, así como las oficinas administrativas y las del intendente, los baños, la cocina, el comedor y un lugar para la venta del pulque. Se podían encontrar *Coacallis* que recibían al pueblo en general y a otros que estaban reservados para las clases superiores. La diferencia radicaba en el tipo de material utilizado para su construcción, en los acabados y en la decoración, así como en los baños, disponiendo las clases superiores de *Temascallis* o baños de vapor y el pueblo únicamente de baños de agua fría. Los *Coacallis* eran propiedad del Estado, quien proporcionaba el alojamiento gratuitamente, pero no así la alimentación que corría por cuenta del viajero, y su administración se encomendaba a estudiantes egresados de los *Tepochcalli* o escuelas especiales donde se enseñaba expresamente la forma de administrar dichos establecimientos.

Después de la conquista, la ruta que más se frecuentó fué la que unía a Villa Rica, esto es Veracruz, con la Ciudad de México. No es de extrañar, pues, que el primer mesón colonial se estableciera en Pinavizapa, posiblemente la actual Orizaba, el 20 de junio de 1525, siendo su propietario Francisco de Aguilar, quién pasó así a la Historia como el primer mesonero de la Nueva España. No contento con eso, meses más tarde, el 10 de octubre del mismo año, solicitó y obtuvo licencia para abrir un segundo mesón en el camino de Medellín a Veracruz, creando de ese modo la primera cadena hotelera mexicana. Pero no fué sino hasta el 1o. de diciembre de 1525 cuando se estableció el primer mesón dentro de la Ciudad de

México, obra de Pedro Hernández Paniagua, quién obtuvo licencia para instalarlo en la que se denominaría con toda propiedad Calle de Mesones. Junto con la de Hernández Paniagua, se otorgó otra licencia al emprendedor Francisco Aguilar, quién de esa forma y en la Ciudad de México montaba el tercer establecimiento de su cadena.

Pero los españoles en su afán de conquista, no se conformaron con la ruta habitual de México a Veracruz sino que buscaron por todos los medios el abrir otras nuevas. Esto explica que el 26 de julio de 1525 se autorizara a Juan de la Torre el establecimiento de una venta en despoblado en el camino de Michoacán, entre Taximaroa e Ixtalavaca. El 14 de septiembre de 1526 Juan Paredes, a nombre de Rodrigo Rengel, solicitaba licencia para abrir un mesón en el pueblo de Cholula y el 12 de octubre del mismo año Juan de la Torre obtenía nuevo premiso para un mesón en Cuernavaca, creando así su propia cadena mesonera. Poco a poco, con la fundación de nuevas poblaciones y la construcción de caminos transitables se fué cubriendo de albergues el territorio de la Nueva España. Algunos, como el de San José de Perote, que todavía existe, dieron nombre a la localidad, Perote, que era el apodo de su propietario, Pedro Ansures, hombre de estatura elevada. Dicho mesón, ubicado en el camino Real de la Villa Rica de México, contaba además con una iglesia de dos torres que utilizaban no sólo para las ceremonias religiosas sino también como refugio o fortaleza para posibles ataques. Mención especial merece la Venta de Doña María, perteneciente a la primera mesonera de la Nueva España, doña Marina Gutiérrez Flores, esposa de don Alfonso de Aldrete, hijo natural, según se decía del rey Fernando el Católico.

Dicho establecimiento, situado en el kilómetro veintiuno de la carretera México-Toluca y en el paraje conocido todavía como La Venta, tuvo gran importancia en esa época, como es natural y ante la creciente demanda de solicitudes para la apertura de albergues, las Autoridades de la Colonia no tardaron en fijar normas muy concretas al respecto. Y así, en una Acta del Cabildo de la Ciudad de México, fechada el 9 de enero de 1526 se encuentra el siguiente texto (tal y como lo cita Luis González Obregón en su obra México Viejo):

"Este dicho día los dichos señores dixeron que por cuanto los días pasados se dio licencia a Pedro Hernández Paniagua para que pudiese tener mesón para que coxiese a los forasteros o les diese de comer a ellos e a las personas que allí se llegasen a posar e no le fue dada la horden e manera que habían de tener con los dichos guéspedes, que mandavan e mandaron que el dicho Pedro Hernández o otro cualquier mesonero de esta cibdad lleve por cada tabla a cada persona que diere de comer o cenar dándole asado e cocido e pan e agua, un tomín de oro. Yten que lleve por cada persona que durmiere en su casa dándole cama de su xergón e ropa limpia de la tierra, un real. Yten que lleve por cada almud de mahiz medio real. Yten que si vendiere azeile e vinagre o quezo por menudo, que gane la tercia parte de como valiere en la cibdad al dicho tiempo por arroba. Todo lo cual mandaron que guarde e cumpla el dicho mesonero o otros cualquiera que tuvieron mesón en esta Ciubdad, so pena de que por primera vez lo paguen con el quatro tanto lo que asy llevaren en demasiado e por la segunda las setenas e por la tercer le sean dados cient azotes públicamente. E mandaron que tenga este aranzel en parte donde se puede ver e leer para que

cada uno sepa lo que ha de dar so pena de veinte pesos de oro, la mitad para las obras públicas e la otra mitad para el Juez e denunciante".

Bien podemos concluir que este valioso documento podría ser considerado como el primer Reglamento de Establecimiento de Hospedaje en México.

Las características generales de los albergues no sufrieron variaciones notables durante toda la época de la Colonia. Tanto las ventas como los mesones no se caracterizaban precisamente por dar un servicio de lujo, si bien aquéllas, al ser utilizadas necesariamente por la nobleza en sus desplazamientos, ofrecían mayores comodidades. Pero llegó el siglo XIX, trayendo consigo grandes y significativos cambios. Los vientos de la Independencia soplaban fuerte sobre México y parecía que no había lugar para pensar en otra cosa que no fuera la lucha. No obstante, una mañana de 1818 los habitantes de la Ciudad de México, al pasear por las calles del Refugio y del Espíritu Santo, hoy 16 de Septiembre e Isabel la Católica, se encuentran con una palabra totalmente desconocida para ellos: *Hotel*. Estaba sobre la fachada de un caserón ubicado precisamente en la esquina de dichas calles y dentro de un gran rótulo que decía "Hotel de la Gran Sociedad". De esta forma la Nueva España se anticipaba a los mismos Estados Unidos en poner fin a la era de los mesones con diez años de diferencia. Lo curioso es que el flamante hotel no tenía nada de flamante. Se trataba de un establecimiento que había nacido como mesón a finales del siglo XVIII y que a poco de iniciarse el XIX había adoptado el nombre de "Posada del Espíritu Santo", y sus avispados dueños, tras edificar un segundo piso con cuartos privados en su mayoría, no dudaron en rebautizar con el actualizadísimo nombre de *Hotel*, funcionando como tal hasta 1898 en que se demolió para construir el que fuera edificio de la "Casa Boker", hoy "Cia. Ferretera Mexica". Ya para este año había en la Ciudad de México veintiún hoteles funcionando y varios más en construcción, sobresaliendo el "Hotel de la Bella Unión", en la esquina que formaban las calles del Refugio y la de la Palma, por los bustos de héroes de la Independencia que adornaban los medios puntos de los balcones del piso superior, así como el "Hotel Iturbide", por ocupar el elegante palacio construido en el siglo XVIII por la marquesa de Valparaíso y que fuera efímera residencia del primer Emperador de México. Pero indiscutiblemente el título de primeros hoteleros del Continente se lo ganaron don Francisco Solares y don Francisco Coquelet por haber contruído en la antigua "Posada del Espíritu Santo" las primeras habitaciones destinadas expresamente a servir como cuartos de huéspedes, naciendo así el primer hotel en América. Sin embargo, la primera cadena de este nuevo tipo de establecimientos de hospedaje se debe a don Francisco de Iturbide, propietario de numerosas fincas urbanas en el corazón de la Ciudad de México, gracias a la desamortización de bienes eclesiásticos ordenada por el Gobierno que dedicó algunas de ellas para edificar otros hoteles, como el "Hotel Iturbide", ya mencionado, el "Hotel de San Carlos", en la actual calle de Francisco I. Madero y el "Hotel Bazar", en la que hoy es Av. Isabel la Católica.

Durante la época de Don Porfirio Díaz, la paz que se disfrutaba en México hizo posible que se alcanzasen grandes progresos materiales, que unidos al impulso que se dio al ferrocarril, motivó el auge continuo de la industria hotelera; y así, en 1910 con motivo de las fastuosas celebraciones por el centenario de la Independencia, la Ciudad de México disponía para sus visitantes de 53 hoteles y 6 casas de huéspedes. Pero en el mes de noviembre de ese año estalló la Revolución, provocando un estancamiento lógico en la industria del hospedaje, la cual duraría diez años.

En 1920 el país entró en la fase de afianzamiento de un Gobierno Institucional que garantizaba las condiciones imprescindibles para un desarrollo económico y social, contribuyendo de esta forma a crear el clima de seguridad necesario para las inversiones en los diversos sectores. Como consecuencia resurgió la Industria Hotelera que en 1921, ofrecía ya 400 cuartos de calidad turística apropiada en todo el territorio nacional. En 1922 Don Lucas de Palacio creó la "Asociación de Administradores y Propietarios de Hoteles", que más tarde se convertiría en la "Asociación Mexicana de Hoteles", iniciándose de este modo lo que podríamos llamar la Etapa Turística en la Historia de la Hotelería en México. Dicha etapa se caracterizó por la importancia que se empezó a dar al turismo procedente de otros países hasta el punto de que en 1926 el Gobierno promulgó la *Ley de Emigración*, donde por primera vez se define el concepto de *turista* como *persona extranjera que visita la república por distracción y recreo y cuya permanencia no exceda de seis meses.* En 1929 se crea la Comisión Mixta Pro Turismo, dependiente de la Secretaría de Gobernación, con el fin de fomentar y desarrollar el turismo del país, lográndose en dicho año una afluencia turística de 19164 visitantes. A partir de ese momento el Gobierno no dejó de impulsar este tipo de actividad, lo que benefició grandemente a la Industria Hotelera, llegándose en 1936 a recibir 100226 turistas.

Después de la Segunda Guerra Mundial los establecimientos hoteleros empezaron a polarizarse en dos tipos bien definidos, los destinados a huéspedes en *viaje de negocios* y aquellos propios para huéspedes en *vacaciones*. Los primeros, también conocidos como de *intereses obligados*, surgieron en aquellas localidades que, sin contar con grandes atractivos turísticos, se veían favorecidos por circunstancias de tipo económico, como las ciudades capitales de los Estados o aquellas donde se instalaba la industria petrolera. Los hoteles propios para vacaciones aparecieron fundamentalmente en las costas del Pacífico, siendo el puerto de Acapulco uno de los ejemplos más reveladores el respecto, aunque también existen grandes concentraciones hoteleras en Zihuatanejo, Manzanillo, Península de Santiago, Barra de Navidad, Melaque, Puerto Vallarta, San Blas, Mazatlán, Guaymas, San Felipe, Santa Rosalía, La Paz, Cabo San Lucas y Ensenada, sobresaliendo recientemente, Cancún y Puerto Escondido.

Para terminar este breve recorrido histórico podemos decir que los primeros *coacallis* de la hotelería azteca se han convertido en cerca de 8000 establecimientos

que actualmente ofrecen casi 250000 cuartos a los visitantes que recorren continuamente los confines de México.

3.1.3 Sistemas de Información en la Hotelería

La actividad turística en México se ha venido intensificando por el apoyo recibido por el Plan Global de Desarrollo (1980-1982) donde se señala: "El sector turismo constituye una palanca importante para el desarrollo principalmente en su aspecto regional por la ocupación que produce, las divisas que genera y las oportunidades de recreación que se inscriben en la connotación del bienestar social que se persigue. El desarrollo turístico se orientará en mayor medida hacia el fomento del turismo interno, sin descuidar los pogramas destinados a la captación de divisas."

La hotelería de gran turismo en México esta administrada en su mayor parte por representaciones extranjeras. Estas compañías hacen uso de la informática sobre todo mediante la tecnología que han traído del extranjero. También es frecuente que los despachos que asesoran a estas cadenas les proporcionen información sobre los distribuidores de software que operan en el país y cuentan con paquetes acordes a sus necesidades. Los sistemas de información en la hotelería se definen como la organización de las Bases de Datos encaminadas a satisfacer las necesidades de información que coadyuven indirectamente a que la estancia de un turista sea placentera, así como a administrar eficazmente la empresa.

Para que un sistema automatice eficazmente las operaciones de un hotel debe contener las características y estándares de la industria hotelera.

Un hotel se divide en:

- División Operativa
- División Administrativa y Financiera

3.1.4 División Operativa del Hotel.

La División Operativa se integra de todos los servicios del frente del hotel o front desk. En ella se incluyen los departamentos de reservaciones, recepción, teléfonos, ama de llaves, cajas de recepción, auditoría nocturna y servicio o asistencia a huéspedes.

3.1.5 Reservaciones.

Este departamento requiere del control de grupos, elaboración de paquetes, bloqueo de cuartos para agencias, depósitos a cuenta, pronósticos de ocupación, control de agencias, etc. Tiene como objetivo el de mantener un porcentaje alto de ocupación y realizar los pronósticos de ocupación.

3.1.6 Reservaciones Centralizadas.

Su objetivo es el de ayudar a la gente que busca hospedaje en una zona específica a encontrarlo con un mínimo de dificultades y a la vez ayudar a los hoteles a llenar sus habitaciones desocupadas.

Durante varios años, diversas empresas grandes han operado una oficina central de reservaciones permitiendo a la gente reservar alojamiento en diferentes hoteles propiedad de ese grupo particular. La selección de hospedaje y la ubicación se limita a los hoteles que forman parte de este grupo. Un Sistema Centralizado de Reservaciones independiente de cualquier hotel o compañía ofrece una gama mucho más amplia de hoteles, sobre un área más grande. El viajero puede reservar el alojamiento que requiera en las áreas cubiertas por dicho servicio y permite a los hoteles dentro del sistema ampliar su área de penetración hacia clientes con perspectiva.

Para hacer una reservación, el cliente telefonea al centro de reservaciones más cercano y solicita lo que requiere. Estos datos alimentan a una computadora que contiene toda la información relevante sobre el alojamiento de dicho tipo, disponible dentro del sistema y en cuestión de segundos se informa al cliente sobre lo que se le puede ofrecer para satisfacer su requerimiento. Si el cliente acepta, se hace la reservación con el hotel seleccionado a través del Servicio de Reservaciones.

Para que un hotel forme parte del sistema es necesario que el centro de reservaciones reciba determinada información respecto al establecimiento, incluyendo el número de diferentes tipos de habitaciones que se destinarán al sistema centralizado. Se le asigna al hotel un número adicional para evitar que de una información falsa el Servicio de Reservaciones a un forastero.

Es importante que el recepcionista del hotel mantenga al Servicio de Reservaciones informado de cualquier cambio en la disponibilidad de cuartos asignados al servicio. Si, debido a una gran demanda por alojamiento, algunos de los cuartos asignados a este servicio se requieren para cubrir las propias reservaciones del hotel, entonces estos cuartos podrán ser retirados por el recepcionista, informando

debidamente al centro de reservaciones. Así mismo, si resultan disponibles más habitaciones y se informa al servicio esto se alimantará a la computadora y por lo tanto estarán disponibles para ofrecerce a través del centro de reservaciones. A menos que el Servicio de Reservaciones no se mantenga al tanto de los cambios en la disponibilidad de habitaciones, surgirán dobles reservaciones y problemas para los empleados, las habitaciones que se encuentren libres no se programarán en la computadora y por lo tanto no se ofreceran a los clientes en perspectiva.

3.1.7 Procedimiento de reservaciones.

Cuando una persona necesita alojamiento telefonea al centro más cercano de Servicio de Reservaciones y define sus requerimientos -tipo y rango de tarifas de la habitación, fecha de llegada, duración de la estancia y lugar en que necesita el alojamiento-. La operadora del Servicio de Reservaciones alimenta esta información a la computadora que elabora, en cuestión de segundos, una selección de hoteles que puede satisfacer esta solicitud. La información aparece en una pantalla de video frente a la operadora y se pasa entonces al solicitante a través de la operadora del Servicio de Reservaciones que responde a otras preguntas, cosultando un directorio compilado de la información proporcionada por el hotel al entrar a formar parte del sistema.

Cuando la persona que llama decide en que hotel se alojará la operadora informa a la computadora que automáticamente tache esa habitación o habitaciones de la reserva de cuartos disponibles en ese hotel durante dicho periodo. Entonces la operadora notifica al hotel, ya sea por teléfono o por fax, a menos de que la reservación sea muy adelantada y se pueda informar al hotel por correo.

El recepcionista del hotel, al recibir esta información la anota en el diario bajo la fecha apropiada y la trata como una reservación ordinaria.

3.1.8 Procedimiento de cancelación.

Si el cliente en perspectiva llegara a cancelar la reservación con el mismo hotel, el recepcionista deberá informar al Servicio de Reservaciones de inmediato. Si no se hace esto, el Servicio de Reservaciones no sabra si poner la habitación de nuevo en la computadora para volver a reservarla. Así mismo, si no se informa al Servicio de Reservaciones, al hotel se le cobrará la reservación. Si el presunto huesped cancelara con el centro de reservaciones, este último deberá tomar la acción necesaria e informar al hotel.

3.1.9 *Procedimiento de no show (Incumplimiento)*

En caso de un no show (la falla de que no llegue una persona que ha reservado una habitación y no la ha canccelado) el recepcionista deberá informar al Servicio de Reservaciones dentro de un plazo de 24 hrs. para evitar que se le haga el cargo por dicha reservación al hotel.

3.2 *Implementación de Cliente Servidor (Consideraciones Técnicas)*

En el pasado, las interfaces de usuario eran construcciones simples enfocadas a terminales simples. El objetivo primario se basaba en el código de las transacciones y en la Base de Datos, dejándole al humano una simple respuesta, como una extensión de la aplicación. Pero con aplicaciones Cliente/Servidor, los papeles se han volteado. El objetivo final de las soluciones Cliente/Servidor es el de proveer aplicaciones de misión crítica que tienen facilidad de uso y sensibilidad de PCs que trabajan fuera de red (standalone).

Cliente/Servidor es primeramente una relación entre programas corriendo en máquinas separadas. Como tal, requiere una infraestructura para realizar tareas que las PCs trabajando aisladamente nunca se hubieran preocupado en hacer. Por ejemplo, comunicaciones robustas entre procesos a través de LANs deben ser incluidos en el diseño, interfaces gráficas usando Interfaces Gráficas de Usuario (GUIs) e Interfaces de Usuario Orientadas a Objetos (OOUIs) deben ser explotadas para crear aplicaciones que parezcan y se sientan más cercanas a objetos del mundo real que a procesos de programación. Las interfaces de usuario, enfocadas al trabajo humano, son ahora más complejas, sensibles y de acuerdo al medio ambiente. Los Clientes OOUI llevan al humano a un ciclo distribuido, el cual inevitablemente suma una multitud de complicaciones. Los humanos cometen grandes cantidades de errores, hacen cosas inesperadas y típicamente requieren gran cantidad de información de diversas fuentes. Los sitemas OOUIs más avanzados introducirán un nuevo concepto de *superclientes* que convertirá a la estación de trabajo del Cliente en un lugar de trabajo multimedia donde muchos diálogos paralelos serán conducidos con el Servidor.[1]

Los sistemas OOUIs también brindan al usuario mucha más libertad que los sistemas GUIs o los basados en terminal. Los usuarios son libres para organizar

[1]El término "supercliente" fue introducido por Forrester Research (Mayo, 1992) para describir estaciones de trabajo de Clientes que son conectadas a Servidores múltiples, proporcionando una mesa de trabajo altamente visual con aplicaciones entrelazadas y que requiere preferentemente multitareas. Forrester predice que los embarques para tales estaciones de trabajo crecerán de 829 mil en 1992 a 2.7 millones en 1994 (23% de estos Clientes se espera utilicen OS/2 2.0).

sus objetos visuales (y mesa de trabajo) de cualquier forma en que a ellos les plazca. Ellos no están atados a una lógica rígida de aplicaciones orientadas a tareas. Los sistemas OOUIs no tienen paneles principales y pantallas de navegación. De hecho, con el Workplace Shell, es difícil decir donde comienza una aplicación y donde termina otra (o qué es un objeto de una aplicación contra un objeto del sistema). Solo hay objetos visuales por doquier.

Lo anterior obliga a formular dos importantes preguntas: ¿Los sistemas Cliente/Servidor requieren una nueva perspectiva del desarrollo de sistemas?. En donde el diseño usado comienza predominantemente en la Base de Datos, ¿Qué surge primero: el Cliente o el Servidor?

Para contestar estas preguntas, discutiremos la anatomía de la Base de Datos de las aplicaciones Cliente/Servidor, pasando por los tres componentes de las aplicaciones Cliente/Servidor: el Cliente, el Servidor y las transacciones de la red de trabajo. Basado en esta discusión, propondremos una estrategia para el desarrollo de sistemas para aplicaciones Cliente/Servidor.

3.2.1 Comparando sistemas Cliente/Servidor del tipo de Procesamiento de Transacciones en Línea (OLTP[2]) y Sistemas para la Toma de Decisiones (DSS[3]).

La Base de Datos central de las aplicaciones Cliente/Servidor se clasifican en dos categorías: Sistemas para la Toma de Decisiones (DSS), y Sistemas de Procesamiento de transacciones en Línea (OLTP). Estas dos categorías de Cliente/Servidor proveen dramáticamente diferentes tipos de soluciones de negocio. Estas diferencias tendrán que ser entendidas antes de responder, ¿Cuál sistema se crea primero: el Cliente o el Servidor?

La anatomía de los sistemas Cliente/Servidor para soporte en decisiones.

Los sistemas de soporte en decisiones son utilizados para analizar información y generar reportes. Estos proveen la información que los profesionales necesitan para tomar mejores decisiones de negocio. Un sistema de soporte en decisiones satisfactorio debe proveer al usuario un acceso flexible a la información y las

[2] On-Line Transaction Processing
[3] Decision Support Systems

herramientas para manipular y presentar los datos en todo tipo de reportes. Los usuarios deberán ser capaces de construir consultas elaboradas, de responder a preguntas "qué si", de buscar relaciones entre los datos, de graficarlos y de moverlos a otras aplicaciones tales como hojas de cálculo y documentos en procesadores de palabras. Los sistemas de soporte en decisiones generalmente no son críticos por tiempo y pueden tolerar bajos tiempos de respuesta. Los sistemas Cliente/Servidor de soporte en decisiones típicamente no son recomendables para ambientes de producción de misión crítica de negocio. Estos sistemas tienen pobres controles de integridad y son limitados en su capacidad para accesar varias tablas. Las operaciones envueltas en la búsqueda de la información pueden manejar grandes cantidades de datos lo cual representa que el nivel de control de concurrencia no es muy granular; por ejemplo, un usuario puede querer ver y actualizar una tabla entera.

Los sistemas DSS están construidos a base de SQL dinámico sobre Base de Datos de Servidores. El lado del Cliente de la aplicación es construido usando una nueva generación de herramientas para el diseño de pantallas que permiten a no programadores construirlas tipo GUI y generar reportes con solo pintar y seleccionar.

Estas herramientas de Base de Datos permiten combinar visualmente objetos gráficos como botones de radio, cajas de chequeo y barras de opciones para crear paneles de despliegue sofisticados que interactúan directamente con la Base de Datos. La mayoría de estas herramientas GUI tienen opciones de generación automática de formas preconstruidas que pueden ser usadas para actualizar, agregar o borrar registros de la Base de Datos. Estas, proveen la habilidad de crear consultas de varias tablas y de desplegar los resultados en un formato con un registro por forma o con formas con varios registros por columna con listas de selección. Las herramientas hacen uso del diccionario de la Base de Datos para mapear columnas de tablas y así presentarlas en la pantalla. Los enunciados de SQL pueden ser construidos "en el aire" y ser asociados a un cierto botón del ratón u opción de un menú. Las herramientas proporcionan facilidades visuales para crear tablas base y para definir relaciones entre tablas.

Las herramientas más sofisticadas permiten al usuario crear ligas con otras aplicaciones en la mesa de trabajo usando el sujetapapeles o Intecambio Dinámico de Datos (Dynamic Data Exchange, o DDE). La técnica de cortar y pegar a través del sujetapapeles es usada para transferir información de otras aplicaciones bajo el control del usuario. DDE es utilizado para actualizar documentos automáticamente y hojas de cálculo con los resultados de una consulta. Esto permite automatizar textos tal fácil como " Pon la salida de esta consulta dentro de esta hoja de cálculo, luego utiliza un paquete de gráficos, imprime el reporte, y envía los resultados a una lista de distribución por medio del correo electrónico."

La anatomía de los sistemas Cliente/Servidor para OLTP.

Los sistemas Cliente/Servidor OLTP son usados para crear aplicaciones en todo tipo de negocios. Esto incluye sistemas de reservaciones, puntos de ventas, sistemas de seguimiento, control de inventario, estaciones de trabajo de corredores de bolsa y control de piso de un área de manufactura. Estas son aplicaciones típicas con misión crítica que requieren un tiempo de respuesta de 1 a 3 segundos un 100% del tiempo. El número de Clientes soportados por un sistema OLTP puede variar dramáticamente, pero el tiempo de respuesta debe mantenerse bajo los parámetros anteriores. Las aplicaciones OLTP también requieren de estrictos controles de seguridad y de integridad para la Base de Datos. La confiabilidad y la disponibilidad del sistema completo debe ser muy alta. La información debe mantenerse consistente y correcta.

En los sistemas OLTP, el Cliente interactua con un Servidor de Transacciones en lugar de interactuar con un Servidor de Base de Datos. Esta interacción es necesaria para proporcionar el alto nivel de ejecución que este tipo de aplicaciones requiere. Con un Servidor de Transacciones, el Cliente invoca *procedimientos remotos* que residen en un Servidor. Estos procedimientos remotos se ejecutan como transacciones contra la Base de Datos del Servidor. Las aplicaciones Cliente/Servidor OLTP requieren de código escrito tanto para el componente Cliente como para el Servidor de transacciones. La carga que representa la comunicación en las aplicaciones OLTP es mantenida al mínimo. La interacción del Cliente con el Servidor de transacciones es limitada típicamente a cortos intercambios estructurados. Los intercambios consisten en un simple solicito/contesto (en comparación a múltiples sentencias de SQL). Un protocolo punto a punto es necesario para realizar la llamada al procedimiento remoto y obtener los resultados.

Así, como se puede ver, desarrollando OLTP en un ambiente Cliente/Servidor aún requiere una intensiva involucración del programador. Los programadores necesitan escribir el código de las transacciones del Servidor, resolver la semántica de los intercambios en la red y desarrollar la aplicación del Cliente. Los Clientes que no han sido desarrollados con Interfaces Gráficas de Usuario (GUI) OLTP como los robots, probadores, ATMs pueden requerir menos código en el lado del Cliente. Pero, GUIs y OOUIs se están convirtiendo en la cara predominante de soluciones de negicio OLTP. En lugar de utilizar el teclado, los usuarios de Clientes GUI pueden interactuar con el Servidor OLTP a través de un ratón e iconos gráficos, menús y formas.

Se están generando herramientas para ayudar a los desarrolladores en la creación de Clientes GUI (todavía no para OOUIs) para sistemas OLTP. Las herramientas de generación GUI permiten a los desarrolladores (típicamente programadores) asociar objetos gráficos predefinidos -tales como formas, campos, menús, cajas de

listas y botones- con programas. Estas asociaciones son eventos dirigidos, así que los manejadores de eventos deben ser codificados para proporcionar la lógica de la aplicación. Esto es diferente de la forma de las herramientas de soporte en decisiones, el cual proporciona manejadores de eventos. Las herramientas GUI son categorizadas por las facilidades de programación del manejador de eventos y metáforas que éstas proporcionan.

El esfuerzo de desarrollo para OLTP y Soporte en Desiciones.

Como se muestra en la Tabla 3-1, las aplicaciones para soporte en decisiones pueden ser creadas directamente por usuarios finales. Los administradores de la red están aún necesitando ayuda para dar de alta el sistema Cliente/Servidor, y los administradores de Bases de Datos (DBA) pueden ayudar a ensamblar colecciones de tablas, vistas, y columnas que son relevantes para el usuario (el usuario debería entonces ser capaz de crear aplicaciones para soporte en decisiones sin la participación de otro DBA). Debido a que en el diseño de sistemas Cliente/Servidor para OLTP se tiene mucho más participación, en consecuencia este requiere una gran cantidad de esfuerzo en la programación de adecuación. Para desarrollar estos sistemas satisfactoriamente, necesitaremos crear nuevas propuestas para el desarrollo de sistemas. Estas nuevas propuestas enfatizarán la importancia de una etapa rápida de creación del prototipo, según lo explicaremos en las siguientes secciones.

Tabla 3-1, Comparación del Esfuerzo de Programación para Soporte en Desiciones y OLTP.

Aplicación Cliente/Servidor	Cliente	Servidor	Mensajes
Soporte para la toma de decisiones	Herramienta para soporte en desiciones "off-the-shelf" con programación para usuario final. Manejadores de eventos y comunicaciones con el servidor.	Servidor de base de datos "off-the-shelf". Tablas usualmente definidas para aplicación OLTP. El OSA crea vistas flexibles para hacer al usuario autónomo.	Mensajes SQL que estan siendo estandarizados para interoperabilidad entre multiples proveedores.
OLTP	Aplicación adecuada. La herramienta GUI define la pantalla pero los manejadores de eventos y las llamadas a procedimientos remotos requieren programación en el nivel C.	Aplicación adecuada. El código de las transacciones deberá ser programado en el nivel C. La base de datos es "off-the-shelf".	Mensajes adecuados son optimizados para desempeño y acceso seguro.

¿Cuál viene primero: Soporte en Decisiones u OLTP?

Existe otra pregunta interesante oculta en nuestro análisis del diseño de la aplicación Cliente/Servidor: Cuál viene primero: soporte en decisiones u OLTP? Necesitamos responder esta pregunta porque ambos sistemas usualmente

persiguen la misma información, y necesitamos conocer donde inicia el proceso de desarrollo del sistema. Una aplicación OLTP es un sistema de producción que representa el estado de una unidad de negocios. Las aplicaciones Cliente/Servidor OLTP usualmente reemplazan (o automatizan) un sistema manual o un sistema basado en terminales que se ejecuta en supercomputadoras o superminicomputadoras. Así, típicamente el diseño de una aplicación y su Base de Datos inicia en el nivel de OLTP. El sistema de soporte en decisiones es usado para consultar el estado del negocio como parte de la aplicación OLTP. Muy pocas Bases de Datos multiusuario son especificadas solo para soporte en decisiones. En otras palabras, utiliza un sistema OLTP para obtener la información que será analizada por el sistema de soporte en decisiones. Esto significa que la aplicación OLTP debería ser el objetivo principal del diseño del sistema Cliente/Servidor (al menos hasta que un sistema de producción este en su lugar).

CAPITULO 4. DISEÑO DEL SISTEMA DE RESERVACION DE HOTELES.

4.1 Proceso de Desarrollo en Cliente/Servidor.

El diseño de sistemas de Proceso de Transacciones en Línea tradicionales (terminales) han empezado con los datos. Las pantallas serán desarrolladas primeramente para manejar los procesos de llenado de Bases de Datos, así que serán diseñadas después de que las transacciones y las tablas sean definidas. Las aplicaciones Cliente/Servidor de Procesamiento de Transacciones en Línea (OLTP), por otro lado, requieren un acercamiento más complejo:

- La interface es más flexible que en terminales, y al usuario se le permite hacer más cosas.
- Los diseños de interfaces de usuario orientadas a objetos le dan más inteligencia a las aplicaciones por el lado del Cliente.
- Los mensajes entre los Clientes y los servidores son construidos de acuerdo al Cliente y específicos a la aplicación.
- El diseño debe ser optimizado de forma tal que tome ventaja del paralelismo inherente en la aplicación distribuida.

Lo anterior nos brinda una ventaja única para el diseño de aplicaciones Cliente/Servidor de Procesamiento de Transacciones en Línea. Se debe empezar el diseño tanto con el Cliente como con el Servidor. Así que se tienen dos puntos de arranque en una aplicación Cliente/Servidor: Las Interfaces Gráficas de Usuario / Las Interfaces Gráficas Orientadas a Objetos y el Diseño de los Datos vienen juntos en el nivel transaccional. Las transacciones mapean la pantalla para la Base de Datos y viceversa.

4.2 Prototipos Rápidos.

Una manera de evitar la situación de ¿Quién fue primero, el huevo o la gallina?, desde el desarrollo entre las aplicaciones Clientes centradas en las Interfaces de Usuario Orientadas a Objetos (OOUI) y las centradas en los servidores Manejadores de Bases de Datos (DBMS), es el uso de una metodología de prototipos rápidos. Los prototipos rápidos nos permiten desarrollar sistemas

incrementalmente. Se empieza por el Cliente y se va trabajando interactivamente hacia el Servidor.

Siempre hay que moverse en pasos pequeños, constantemente refinando el diseño. En esta forma de desarrollo de datos, el sistema es incrementalmente refinado hasta el desarrollo de un prototipo lo suficientemente maduro para ser colocado en producción. Esta aproximación podría colocar un gran peso en el programador así como en las aproximaciones tradicionales, las cuales residen en el análisis y diseño del la interface de usuario. El riesgo estriba en que, con el desarrollo a través de prototipos rápidos, se puede poner en producción un prototipo prematuro dentro de producción. En el otro extremo, se puede encontrar otro riesgo: el síndrome del prototipo perpetuo.

Esos riesgos son rápidamente sobrepasados por los beneficios que el desarrollo de prototipos en forma rápida provee en el desarrollo de sistemas Cliente/Servidor de Interfaces de Usuario Orientadas a Objetos con Procesamiento de Transacciones en Línea. En ambientes Cliente/Servidor, es difícil determinar a priori como debe, supuestamente, trabajar el sistema (ej. el diseño de las Interfaces de Usuario Orientadas a Objetos, el rendimiento de la red, la carga de múltiples usuarios). Existe una gran experiencia de que las especificaciones de los usuarios no pueden anticipar todas sus necesidades. Las especificaciones no pueden ser adecuadas convenientemente a las potencialidades que las Interfaces de Usuario Orientadas a Objetos pueden ofrecer. Si se les pregunta a los usuarios para tomarlos como guía, habrá mayores oportunidades de obtener respuestas erróneas basadas en las soluciones actuales (ej. soluciones manuales, soluciones basadas en terminales). En vez de ello, se le debe mostrar a los usuarios lo que la tecnología puede hacer por ellos y las nuevas dimensiones visuales que las Interfaces Orientadas a Objetos ofrecen, posteriormente hay que trabajar con ellos en el desarrollo de la aplicación.

Invariablemente, las Interfaces de Usuario Orientadas a Objetos pueden capturar más el proceso de negocio que las Interfaces Gráficas de Usuario o las basadas en terminales. Un prototipo rápido ayudará en el descubrimiento de los objetos de la aplicación (u objetos de negocio). Los usuarios encontrarán más fácil discutir los objetos de negocio, y el prototipo visual ayudará a desarrollar las especificaciones de diseño. Es más fácil discutir acerca de algo que se puede demostrar funcionando. Más aún, es importante que el prototipo sea desarrollado en un ambiente donde las cuestiones como el rendimiento Cliente/Servidor, la sobrecarga de la red, el manejo del sistema, y el diseño de las transacciones del Servidor puedan ser probadas en vivo (realmente). Los sistemas de Procesamiento de Transacciones en Línea requieren mucho monitoreo, afinación, ajuste, y administración.

4.3 *Desde el Prototipo hasta un Sistema Trabajando.*

Una aplicación puede verse como una colección de objetos visuales de negocio. El diseño de la aplicación empieza desde los objetos (y las vistas) que el usuario ve en la pantalla, y luego se trabaja su camino hacia el Servidor y la estructura de la Bases de Datos. Los prototipos son usados para identificar, con la participación del Cliente, el modelo de usuario: ¿Qué objetos de negocio se necesitan y qué deben hacer? El desarrollo por prototipos es una actividad iterativa con muchos comienzos falsos. A continuación se presenta cómo se desenvuelve un prototipo típico:

1. ***Entendiendo el proceso de negocios.*** Es un prerequisito el que se entienda plenamente de que se trata la aplicación al nivel de negocio. Así que hay que recoger los requerimientos y estudiar las tareas: ¿Qué es lo que el Cliente realmente quiere?

2. ***Definiendo los objetos de negocio.*** Lo que el usuario realmente ve en una aplicación de Procesamiento de Transacciones en Línea son objetos conteniendo vistas las cuales reaccionan a las entradas del usuario y acciones. Los objetos son manipulados de acuerdo a los requerimientos del proceso de negocio. Se deben crear objetos de usuario que correspondan a entidades de la vida real, por ejemplo, el asiento de un avión. ¿Cuáles son los tipos de objetos principales en la aplicación? ¿Qué otros objetos contienen? ¿Cuáles son los atributos de esos objetos? ¿Qué funciones tienen? ¿Cómo se comportan? ¿Cuáles son las relaciones entre los objetos? ¿Cuáles son las composiciones de los objetos (es parte de)? ¿Cuál es la dependencia y las relaciones de colaboración (es análogo a, es del tipo de, depende de)?

3. ***Trabajando en las vistas detalladas de los objetos.*** ¿Qué menús de acciones aplican a cada objeto? ¿Qué vistas son requeridas? ¿Son requeridas vistas de formas de negocio? ¿Pueden ser las vistas agrupadas en páginas de cuaderno? ¿Cuántos controles -campos de entrada y salida, cajas de listado, botones de acción, menús, correderas de ventanas, etc.- aparecerán en las vistas? ¿Qué nivel de ayuda es requerido? ¿Qué validación de datos a nivel de campo es requerido? ¿Cómo son invocados los procedimientos externos y los RPC (llamadas remotas de procedimientos)?

4. ***Desarrollando escenarios.*** Se pueden crear los objetos de la pantalla y animar la aplicación (herramientas de Interfaces de Usuario Orientadas a Objetos proveerán esta habilidad). Se pueden usar los escenarios para validar los modelos de objetos de la interface de usuario. Los escenarios ayudarán también a identificar las mayores interacciones manejadoras de eventos.

5. ***Caminando a través de un escenario de sistema.*** Cada escenario debe seguir una transacción desde su origen hasta su ejecución. Hay que identificar a los protocolos que ligan a los diferentes elementos. Explota este escenario en las áreas que requieran mayor detalle. Hay que ejecutar un escenario de aplicación para cada objeto de negocio. Hay que identificar los comportamientos redundantes. ¿Cuáles objetos pueden reusarse?

6. ***Identificando las fuentes de transacciones.*** Un sistema Cliente/Servidor puede ser pensado como un sistema de eventos manejado por el Cliente. El Cliente en turno es manejado por el usuario quien tiene los controles dentro de los confines del proceso de negocio. El arrastrar y soltar de objetos es, usualmente, el origen de las transacciones. La información requerida para abrir un contenedor o una vista puede ser también el origen de una transacción. Las interacciones de objetos y acciones casi siempre conducen a la generación de transacciones. También existe una alta probabilidad de que eventos, como la entrada de datos, selección de menús, presión de botones puedan generar transacciones. Esta interface visual será, eventualmente, "llevada a la vida". Los puristas Orientados a Objetos pueden pensar en las transacciones como métodos para manipular datos persistentes.[4]

7. ***Definiendo las tablas de las Bases de Datos.*** Es la primera aproximación en la definición de los objetos de la Base de Datos que corresponden a los datos visuales de los objetos de negocio que se acaban de crear. Hay que iterar en este paso hasta que se obtenga lo que se busca. La orientación de objetos y acciones pueden ayudar transformando los objetos visuales en modelos de entidades relacionales para los objetos de las Bases de Datos. Las acciones se traducen en las entidades de Bases de Datos. Hay que publicar las sentencias de CREAR TABLA, y después usarlas para documentar los tipos de datos y las relaciones entre las tablas (como la "integridad referencial").

8. ***Publicando los mensajes de Cliente/Servidor.*** Los mensajes son los métodos por medio de los cuales lo Clientes ejecutan transacciones en los servidores. Hay que definir las principales transacciones y los mensajes de solicitud y respuesta asociados a ellos, incluyendo cualquier transferencia de archivos. Los mensajes proveen el único acoplamiento entre los Clientes y los servidores. La publicación de los mensajes documenta lo que el Servidor realiza. Es un contrato entre el Servidor y los Clientes. Un mensaje típicamente contiene un campo de comandos y un campo de datos, el cual contiene los parámetros de los comandos. Nosotros, en esta tesis, dividimos a las transacciones (o comandos) en dos categorías, basadas en el tipo de respuesta que producen:

[4]En términos estrictos de Orientación a Objetos, un Servidor de Transacciones es un objeto simple que encapsula a la Base de Datos entera - como su instancia privada de datos-. Una aplicación que corre en trasfondo podría ser escrita como un conjunto de objetos que exportan métodos -transacciones-.

- Las transacciones Petición/Respuesta son aquellas que generan respuestas cortas. Estas transacciones son generalmente actualizaciones, inserciones, borrados o consultas de un solo renglón en contra de objetos de la Base de Datos.

- Las transacciones de respuesta de volumen son aquellas que generan uno o más archivos de resultado.

9. ***Desarrollando el código de un objeto a la vez.*** Hay que escribir el código SQL para cada transacción. Hay que realizar el de un objeto a la vez. Existen generalmente muchas aplicaciones por cada objeto de negocio. Hay que validar la interface de usuario y el rendimiento del sistema con el usuario.

10. ***Del prototipo al sistema trabajando.*** Un sistema trabajando es la suma de todos los objetos de negocio que contiene. Partiendo de que se esta desarrollando el sistema gradualmente, se tendrá un sistema trabajando cuando se haya codificado el último objeto de negocio.

La tecnología Cliente/Servidor ofrece a los desarrolladores el potencial para crear nuevas y revolucionarias aplicaciones visuales. La creación de este tipo de aplicaciones requiere la misma integración de la tecnología de Interfaces de Usuario Orientadas a Objetos, Sistemas Operativos, Arquitecturas de Red, y Sistemas Manejadores de Bases de Datos. Para ser exitosos, necesitamos acercarnos a sistemas de desarrollo que enfaticen el rápido desarrollo de prototipos y la involucración de los usuarios finales. Hacerlo nos permitirá explotar la sinergia entre los objetos de Bases de Datos y los objetos de interfaces Gráficas Orientadas a Objetos. Estaremos también en una mejor posición para entender, tempranamente y en proyectos, las oportunidades (por ejemplo el paralelismo) y trampas (por ejemplo, rendimiento y recuperación de errores) los cuales son introducidos al dividir una aplicación a través de la red.

El diseño de aplicaciones a través del acercamiento por prototipos elimina la necesidad de contar con grandes especificaciones. Más importante aún, ésta aproximación le da la oportunidad al usuario de participar en las especificaciones del producto y en el refinamiento que se dá paso a paso. Esta aproximación también se presta para el diseño de aplicaciones distribuidas, dada la refinación y afinación en la distribución de funciones al momento que se vaya aprendiendo más acerca del comportamiento del sistema en la vida real.

Así que, el diseño exitoso empieza en paralelo con el diseño tanto del Servidor como del Cliente. Los dos puntos de arranque son los objetos y los datos de los objetos. El pegamento que une a estos elementos son las transacciones a través de las RPCs (Llamadas Remotas a Procedimientos). Así pues, empezaremos con el Cliente y nos iremos moviendo hacia el Servidor, y viceversa. En cualquier caso, es

recomendable el moverse en pasos pequeños e ir realizando iteraciones. El prototipo visual es el que trae al diseño a la vida. Así que, ¿cuál debe venir primero, el Cliente o el Servidor? ¡Ambos vienen primero!

4.4 Diseño del Sistema de Reservación de Hoteles.

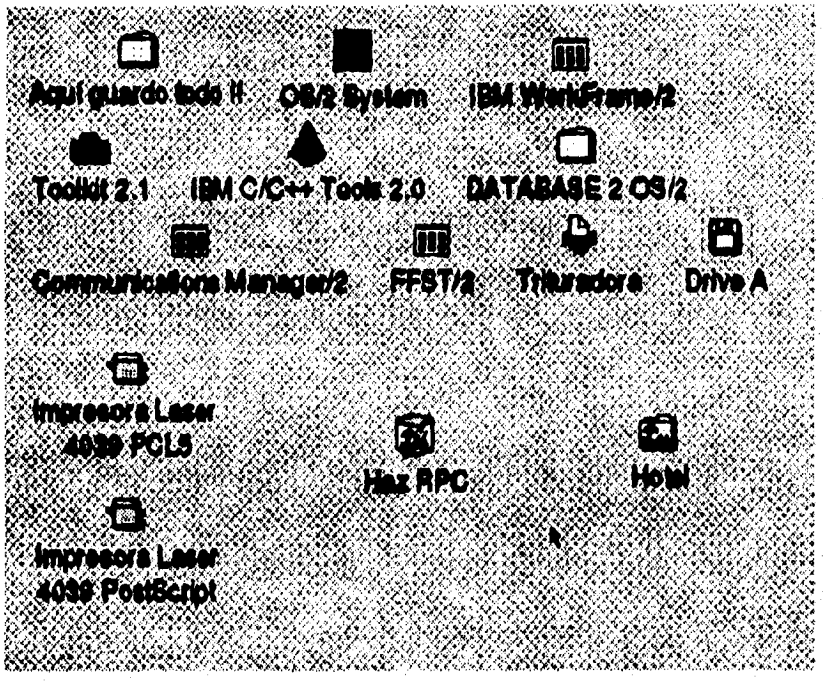
El Sistema de Reservación de Hoteles es una aplicación clásica Cliente/Servidor. El Cliente con interface gráfica orientada a objetos (OOUI) proveerá el front-end usando clases del Workplace Shell (WPS -Escritorio de trabajo-) y del System Object Model (SOM -modelo de objetos del sistema). El Servidor ejecutara las transacciones provistas por un script y una Base de Datos y estará ejecutándose en back-end. El Cliente y el Servidor se comunicaran usando Named Pipes y los servicios de archivos provistos por el producto Lan Server. Las transacciones serán trasladadas a través de Named Pipes y grandes archivos serán movidos sobre la red usando el redireccionador de Lan Server.

4.4.1 El Cliente.

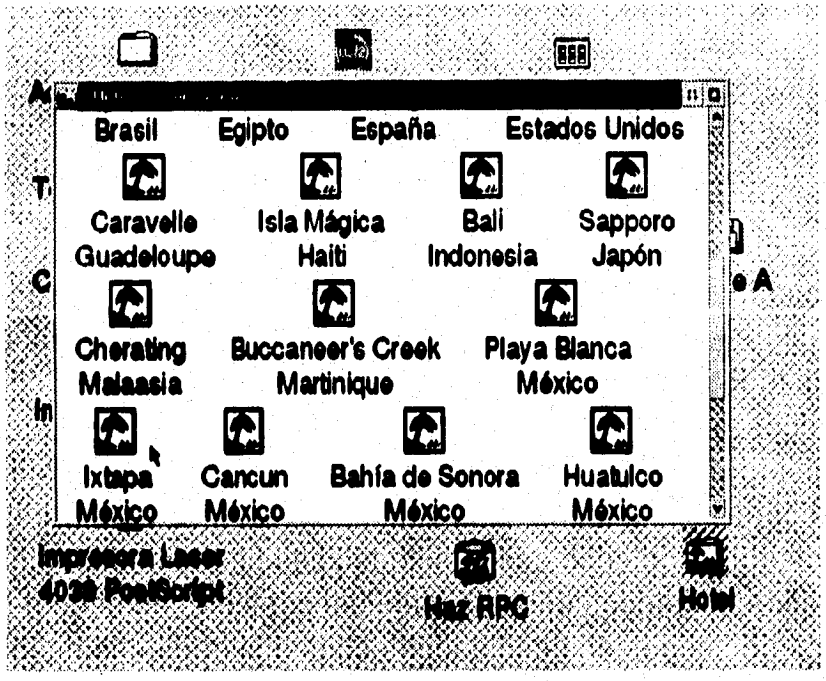
Una aplicación Cliente/Servidor es por naturaleza conducida por el Cliente. Las OOUI van un paso adelante y colocan los controles del Cliente en las manos del usuario a través de la manipulación de objetos visuales. Cualquier cosa que el usuario realice sobre los objetos visuales se llegará a convertir en la fuente de transacciones Cliente/Servidor. El Servidor, mientras tanto, espera pasivamente las peticiones de los Clientes que disparen la ejecución de sus transacciones.

A continuación se describe el proceso OOUI dentro del Sistema de Reservación de Hoteles:

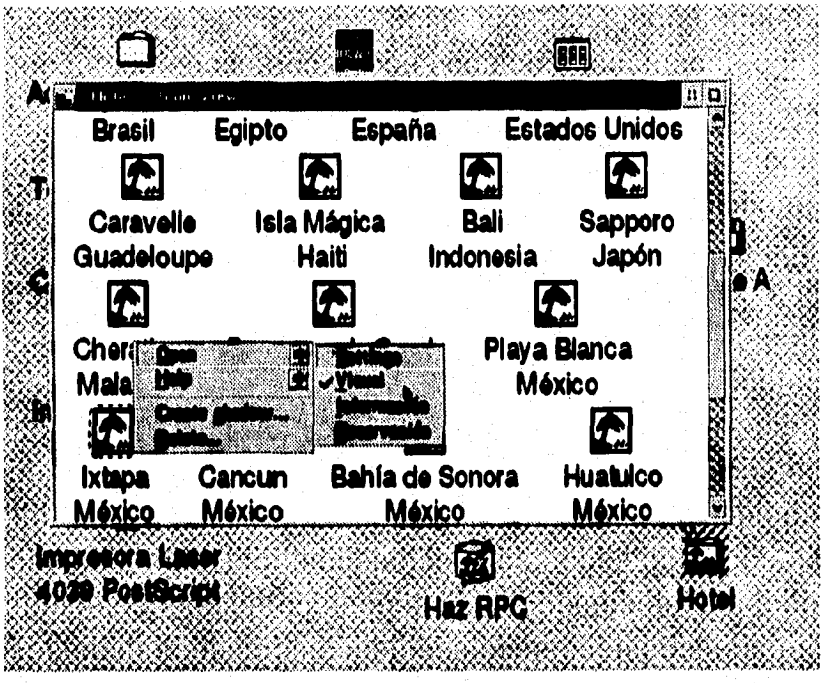
1. Dentro del área de trabajo del OS/2 (WPS) existirá un objeto contenedor de todos los hoteles pertenecientes a la cadena. Existirá además otro objeto llamado ***HazRPC*** que tendrá como propósito el someter transacciones al Servidor. El propósito de ponerlo en el área de trabajo es el de permitirle a los administradores eliminar el objeto y sus múltiples threads.



2. Al momento que se abre el contenedor de hoteles (que se puede hacer de múltiples maneras, entre ellas dando doble clic al mouse sobre el objeto) producirá una serie de acciones. La estación de trabajo Cliente obtendrá una lista de los hoteles que hay en el Servidor. El resultado es devuelto en forma de archivo, el cual será usado por el objeto contenedor de hoteles para poblarse así mismo por medio de la creación de instancias de objetos que representen a los diferentes hoteles. La transacción que regrese la lista de hoteles se llamará ***obten_nombre_hotel***. Todos los manejadores de transacciones del lado del Cliente son manipulados como threads por el objeto ***HazRPC***.

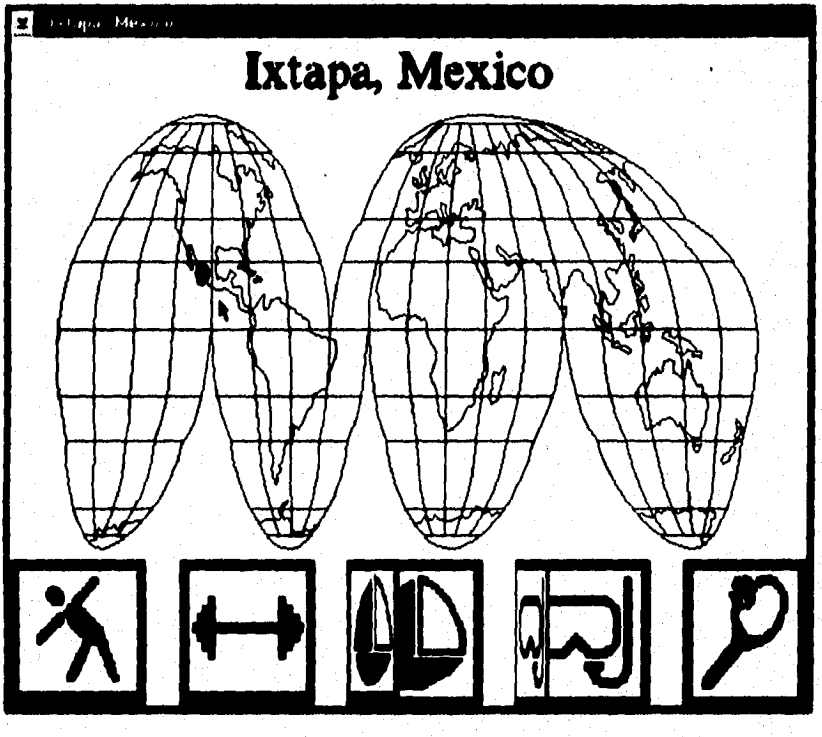


3. Cada objeto representativo de cualquier hotel contendrá un menú pop-up. Dicho menú muestra cuatro diferentes actividades a realizar: Precios, Visuales, Descripción y Reservaciones. Al seleccionar el menú Visual le permitirá al usuario averiguar todo acerca del hotel seleccionado (donde esta, cuales son las facilidades que ofrece, etc.).



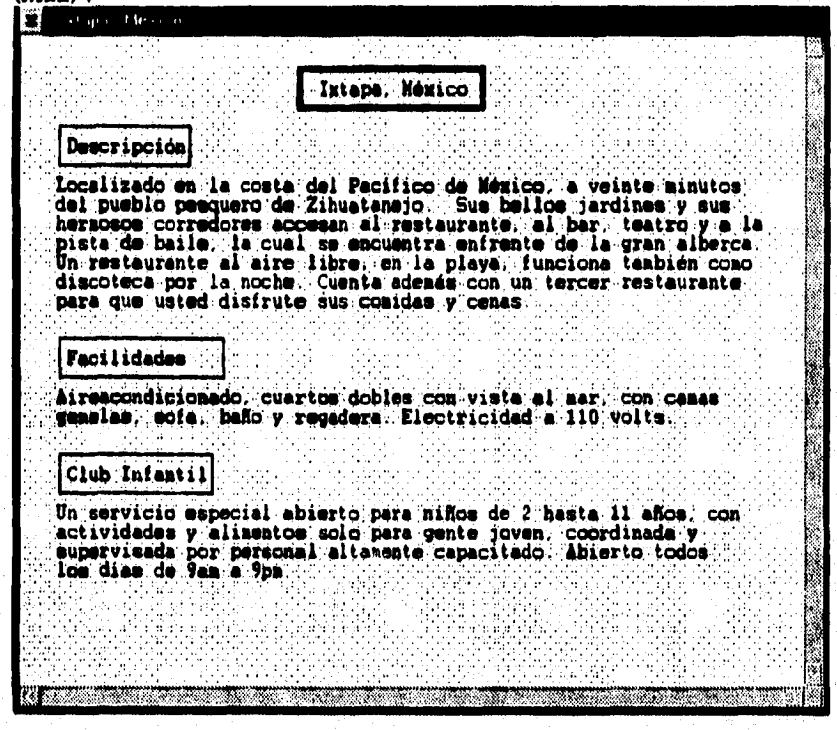
4. Al seleccionar el objeto representativo de algún hotel se disparará una transacción que obtenga toda la información asociada a él, esta transacción la llamaremos ***obten_info_hotel***. La transacción regresará un archivo de tipo "metafile", el cual contendrá información visual y escrita. También regresará varios archivos en formato "bitmap", los cuales al ser desplegados en la pantalla nos brindarán la información respecto a los diferentes deportes que se pueden

practicar en el hotel así como las diferentes facilidades que se ofrecen. Los gráficos que se verán son el trabajo del objeto ***Vee_Hotel***. Este programa es invocado por el objeto representativo del hotel como un proceso separado cuando la vista del hotel es seleccionada. El proceso de Vee_Hotel despliega cada icono de deporte y ejecuta el mapa en formato "metafile".



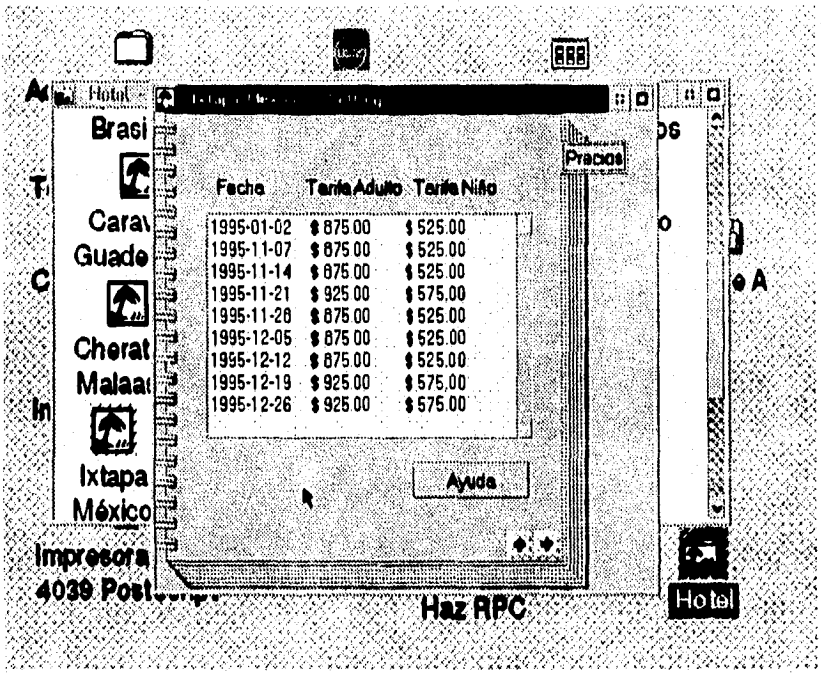
5. La descripción del Hotel, la cual forma parte de las vistas del objeto que representa a un hotel, despliega un archivo de texto con información acerca del hotel. Para ello, cada vez que se seleccione esta vista lanzará una transacción al Servidor llamada ***obten_descripcion_hotel***. El Servidor regresa un archivo de texto. Este texto será entonces desplegado utilizando una rutina del Sistema

Operativo OS/2 de entrada multilínea, llamada en ingles "Multiline Entry (MLE)".



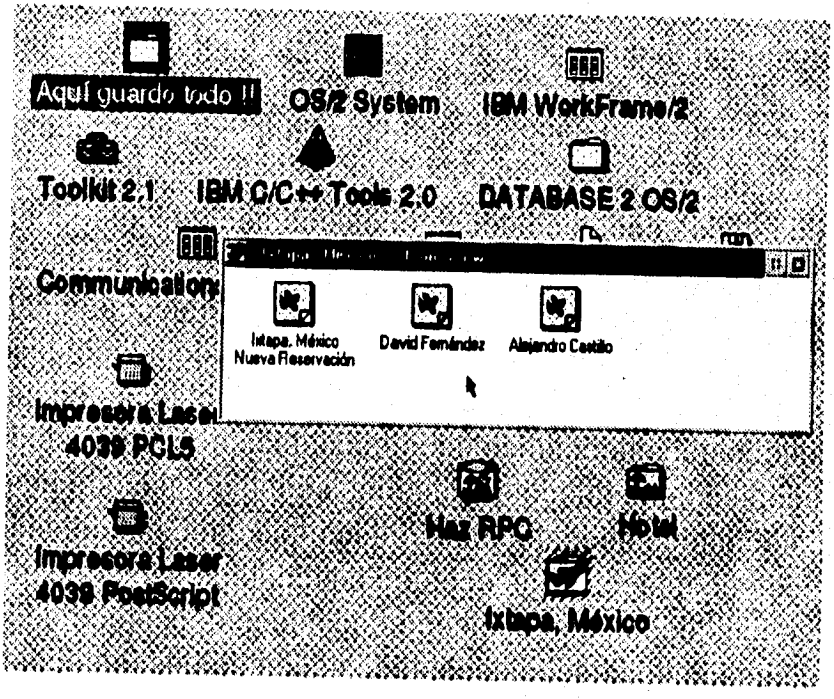
6. Obteniendo los precios del hotel es otra vista que nos muestra una carpeta en la cual, en una de sus hojas, contiene información acerca de los precios del hotel. Los precios al ser variables por temporada son desplegados en ventanas de listas ("scrollable listbox"). Lo que realmente sucede al seleccionar esta vista es el lanzamiento de una transacción hacia el Servidor que llamaremos

obten_precios_hotel. La transacción da como resultado un archivo que contiene una lista de precios para el hotel en particular. Esta vista mostrará como se emplea una técnica en la cual una carpeta que conforma a la definición CUA'91 puede ser usada como un contenedor de ventanas de dialogo. Esta técnica será heredada por el objeto de reservaciones para hoteles.



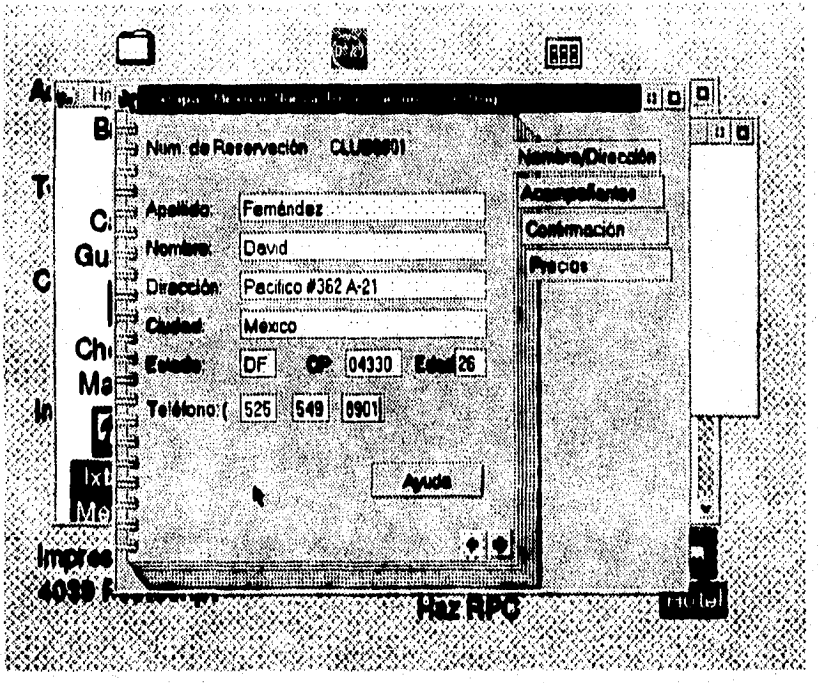
7. La vista de reservaciones de hotel abrirá otro folder el cual contendrá los objetos para la reservación de hoteles. Los objetos, como vemos, contendrán a otros objetos, y así sucesivamente. Esta actividad nos conduce a crear un objeto folder para reservaciones (el cual será otra clase más). Al abrir este objeto folder

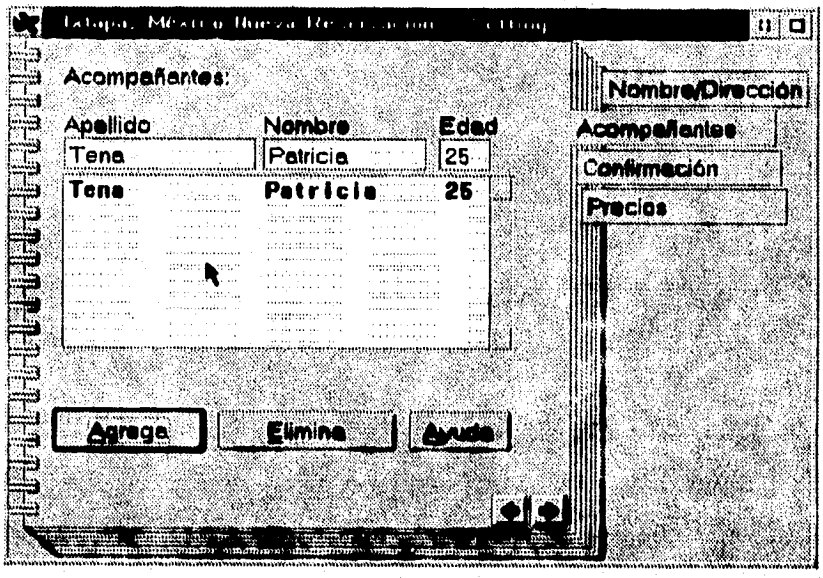
lanzará una transacción, que llamaremos ***encuentra_todas_reserv***, la cual regresará una lista de objetos de reservación. Este objeto folder se poblará así mismo creando nuevos objetos de reservación para cada registro obtenido del Servidor. Así que se tendrán objetos para cada huésped.



8. Al abrir un folder de algún huésped se desplegará una carpeta con muchos diálogos y organizada en páginas. Se podrá visualizar la carpeta yendo hacia atrás y hacia adelante a través de las páginas. Nuestro Sistema de Reservación

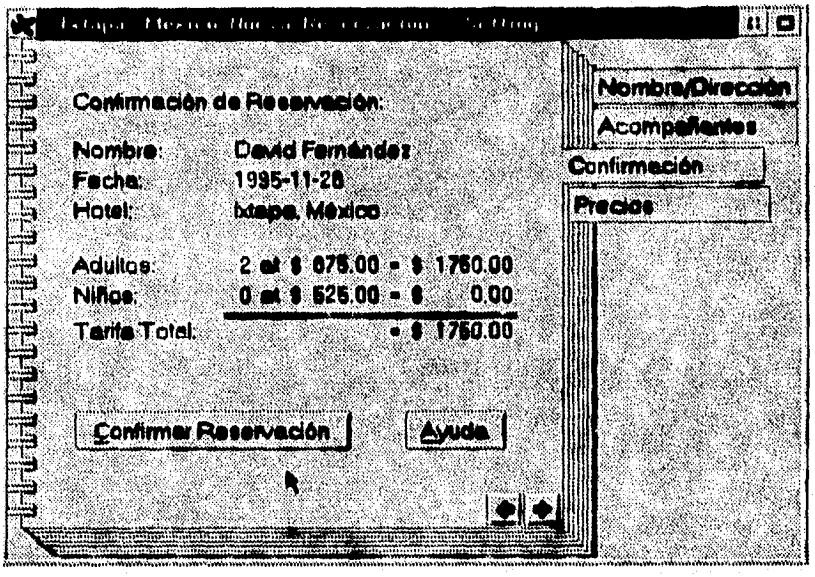
de Hoteles consiste en cuatro páginas de dialogo: reservación, miembros del grupo, confirmación y precios.





9. La página de precios luce similar a la vista en el objeto de hoteles. Cuando alguien selecciona la semana en la que desea vacacionar genera que la caja de diálogo de la clase para reservaciones sea heredada de la clase padre. Con esta herencia no cuesta nada repetir la caja de diálogo. Tanto el código como la transacción son idénticas a la de los precios de hotel.

10. Al presionar el botón de "Confirmar Reservación" se someterá una transacción al Servidor que llamaremos ***agrega_reservacion***.



11. Si se desea cancelar una reservación simplemente se arrastrará el objeto hacia el triturador. Esto producirá que el triturador informe sus objetos cuando ellos estén a punto de ser destruidos. Este informe lanza una transacción que llamaremos ***cancela_reservacion***. El objeto será también destruido.

12. La ayuda estará disponible en todo lugar. Al presionar F1 se invocará una ayuda para contexto; o se podrá solicitar ayuda a través de menús o botones. Al invocarse la ayuda se invocarán páneles de ayuda local IPF. El modelo CUA'91 es usado.

De acuerdo a lo dicho en los párrafos anteriores, el Sistema de Reservación de Hoteles cociste de ocho transacciones. Estas transacciones establecen la liga entre el Servidor y sus Clientes. Ellas representan el conjunto de llamadas a procedimientos remotos exportadas por el Servidor. La interface del mensaje de transacción es la siguiente pieza de información crucial que se necesitará entender para la aplicación Cliente/Servidor de Reservación de Hoteles.

4.4.2 Transacciones.

Las transacciones para el Sistema Cliente/Servidor de Reservación de Hoteles son:

Transacción 1. OBTEN_NOMBRE_HOTEL
Descripción: Esta transacción obtiene una lista con los nombres de los hoteles.
Mensaje de Entrada: "ID_TRANSACCION=1"
Proceso:

SELECT from HOTELES
generación de archivo respuesta
COMMIT

Mensaje de Respuesta:

"TRANSACCION_FIN= EXITO ARCHIVO_RESP= nombre_arch"

Transacción 2. OBTEN_INFO_HOTEL
Descripción: Esta transacción obtiene el nombre de un archivo en formato "metafile" y el nombre de todos los bitmaps de deportes y actividades asociadas al hotel. Esta información es desplegada en la pantalla que contiene el mapa de localización y los gráficos de deportes y actividades.
Mensaje de Entrada: "ID_TRANSACCION=2 HOTEL= nombre"
Proceso:

SELECT from HOTELES
SELECT from ACTIVIDADES
generación de archivo respuesta
COMMIT

Mensaje de Respuesta:

"TRANSACCION_FIN= EXITO ARCHIVO_RESP= nombre_arch"

Transacción 3. OBTEN_DESCRIPCION_HOTEL
Descripción: Esta transacción obtiene el nombre de un archivo que contiene una descripción textual sobre el hotel específico. Este es el archivo de información el cual es desplegado usando la técnica MLE.
Mensaje de Entrada: "ID_TRANSACCION= 3 HOTEL=nombre"
Proceso:

SELECT from HOTELES
generación de archivo respuesta
COMMIT

Mensaje de Respuesta:

"TRANSACCION_FIN= EXITO ARCHIVO_RESP= nombre_arch"

Transacción 4. OBTEN_PRECIOS_HOTEL
Descripción: Esta transacción obtiene una lista de precios para los hoteles. La lista contiene el número de la semana, el precio para adultos, y el precio para niños.
Mensaje de Entrada: "ID_TRANSACCION= 4 HOTEL=nombre"
Proceso:

SELECT from HOTELES
generación de archivo respuesta
COMMIT

Mensaje de Respuesta:

"TRANSACCION_FIN= EXITO ARCHIVO_RESP= nombre_arch"

Transacción 5. ENCUENTRA_TODAS_RESERV
Descripción: Esta transacción obtiene una lista con los nombres de la gente quien ha confirmado su reservación de hotel.
Mensaje de Entrada: "ID_TRANSACCION= 5 HOTEL=nombre"
Proceso:

SELECT from RESERVACIONES
generación de archivo respuesta
COMMIT

Mensaje de Respuesta:

"TRANSACCION_FIN= EXITO ARCHIVO_RESP= nombre_arch"

Transacción 6. ENCUENTRA_RESERVACION
Descripción: Esta transacción obtiene toda la información asociada con una reservación en particular.
Mensaje de Entrada: "ID_TRANSACCION= 6 RESERV_NO= número"
Proceso:

SELECT from RESERVACIONES
generación de archivo respuesta
COMMIT

Mensaje de Respuesta:

"TRANSACCION_FIN= EXITO ARCHIVO_RESP= nombre_arch"

Transacción 7. AGREGA_RESERVACION
Descripción: Esta transacción crea una transacción. El hotel y la semana a viajar son identificados en el mensaje. El mensaje contiene toda la información de los miembros del grupo, números de teléfono, dirección, forma de pago, etc.
Mensaje de Entrada:
"ID_TRANSACCION= 7 HOTEL= nombre SEMANA= fecha NOMARCH= nombre"
Proceso:

INSERT into RESERVACIONES
COMMIT

Mensaje de Respuesta:
"TRANSACCION_FIN= EXITO ARCHIVO_RESP= NINGUNO RESERV_NO= número"

Transacción 8. CANCELA_RESERVACION
Descripción: Esta transacción cancela una reservación identificada por su número de reservación.
Mensaje de Entrada: "ID_TRANSACCION= 8 HOTEL= hotel NUM_RESERV= número"
Proceso:

```
DELETE from RESERVACIONES
COMMIT
```

Mensaje de Respuesta:
"TRANSACCION_FIN= EXITO ARCHIVO_RESP= NINGUNO"

4.4.3 Creación de la Base de Datos

La Base de Datos de los Hoteles se conforma de cuatro tablas, cuyos enunciados de creación se presentan a continuación:

```
-- CREACION DE LA BASE DE DATOS DEL SISTEMA DE RESERVACION DE HOTELES
--
================================================================================
=

EXEC DBA: CONTINUE_ON_ERROR;

-- dropear cualquier Base de Datos llamada HOTEL
EXEC DBA: DROP_DB HOTEL;

-- crear una Base de Datos llamada HOTEL
EXEC DBA: CREATE_DB HOTEL F TEST_DATABASE;

-- bindear el programa rsql
EXEC DBA: BIND HOTEL RSQL.BND ISO REPEATABLE_READ BLOCK_ALL PUBLIC;

-- conectarse a la Base de Datos
EXEC DBA: STARTUSE HOTEL S;

-- crear la tabla de Hoteles
EXEC SQL: CREATE TABLE HOTELES(HOTEL       CHAR(32)    NOT NULL,
                               PAIS        CHAR(32)    NOT NULL,
                               FOTO        CHAR(20)    NOT NULL,
                               DESCRIPCION CHAR(20)    NOT NULL);

-- crear índice para la Base de Datos de Hoteles
EXEC SQL: CREATE UNIQUE INDEX HOTELESX1 ON HOTELES(HOTEL);
```

```
-- crear la tabla de Actividades
EXEC SQL: CREATE TABLE ACTIVIDADES(  HOTEL       CHAR(32)    NOT NULL,
                                    ACTIVIDAD   CHAR(20)    NOT NULL);


-- crear la tabla de Precios
EXEC SQL: CREATE TABLE PRECIOS( HOTEL            CHAR(32)    NOT NULL,
                                FECHA_SEMANA     DATE        NOT NULL,
                                PRECIO_ADULTO    CHAR(8)     NOT NULL,
                                PRECIO_NIÑO      CHAR(8)     NOT NULL);

-- crear indice para la tabla de Precios
EXEC SQL: CREATE UNIQUE INDEX PRECIOSX1 ON PRECIOS(HOTEL, FECHA_SEMANA);

-- crar la tabla de Reservaciones
EXEC SQL:
CREATE TABLE RESERVACIONES(  HOTEL           CHAR(32)    NOT NULL,
                             FECHA_SEMANA    DATE        NOT NULL,
                             APELLIDO        CHAR(32)    NOT NULL,
                             NOMBRE          CHAR(32)    NOT NULL,
                             EDAD            CHAR(3)     NOT NULL,
                             MIEMBROS        CHAR(1)     NOT NULL,
                             RESERV_NO       CHAR(9)     NOT NULL,
                             FORMA_PAGO      CHAR(9),
                             TELEFONO        CHAR(20),
                             DIRECCION       CHAR(32),
                             CIUDAD          CHAR(32),
                             ESTADO          CHAR(2),
                             CP              CHAR(10));

-- crear indice para la tabla de Reservaciones
EXEC SQL: CREATE INDEX RESERVX1 ON RESERVACIONES(HOTEL, FECHA_SEMANA);

-- realizar un commit.
EXEC SQL: COMMIT;

-- desconectarse de la Base de Datos para exportar
  EXEC DBA: STOPUSE HOTEL;

-- importar los datos de la tabla
EXEC DBA: IMPORT HOTEL HOTELES.DEL REPLACE DEL HOTELES;
EXEC DBA: IMPORT HOTEL ACTIVIDA.DEL REPLACE DEL ACTIVIDADES;
EXEC DBA: IMPORT HOTEL PRECIOS.DEL REPLACE DEL PRECIOS;

-- realizar un bind al programa Hotel del Servior
  EXEC DBA: BIND HOTEL HOTELSVR.BND ISO REPEATABLE_READ BLOCK_ALL PUBLIC;
```

4.4.4 Las Nuevas Clases SOM/WPS

Los Objetos del Cliente

El Cliente del Sistema de Reservación de Hoteles pude entenderse muy fácilmente a nivel conceptual. Consiste de cinco clases derivadas de la biblioteca de clases SOM/WPS. ¿Por qué escogimos estas cinco clases? ¿Cómo las seleccionamos? ¿De donde se derivan? Todas estas son preguntas arquitecturales válidas y nos encargaremos de responderlas en este capítulo.

¿Cómo seleccionamos nuestras clases?

No existen reglas rígidas sobre cómo escoger clases. Solo se debe tomar en cuenta la lógica de los objetos, el nivel de aclopamiento requerido, las funciones que puedan fácilmente ser reusadas a través de subclases, protección intensa, desempeño, requerimientos de memoria, etc.

Las clases de objetos básicas

Debemos estar enterados que en el escenario de la parte anterior a este inciso en el que el Sistema de Reservación de Hoteles tiene dos objetos de negocios, estos realmente sobresalen: hoteles y reservaciones de clientes. Así que naturalmente, crearemos las dos clases: **Hotel** y **Resv**. Se puede observar también, desde este escenario, que estos dos objetos son muy similares. Por ejemplo, los dos tienen vistas de cuaderno con una página de precios de Hoteles. "Similar" en POO (Programación Orientada a Objetos) se traduce en subclase de e inherente. Así que definiremos Resv como clase hija de Hotel.

Por lo que, ¿Quién es el padre de **Hotel**? Escogimos la clase **WPTransient** debido a que funciona como buen padre para aquellas clases que no requieren la persistencia de objeto proporcionada por el WPS (nosotros estamos obteniendo nuestra persistencia del Servidor). También introduce menor carga de trabajo que otras clases del WPS.

Hay que recordar también que existe un miembro de soporte en nuestra lista de objetos: el programa **VeHotel**. Este programa presenta las vistas de Descripción del Hotel y Visual del Hotel cuando son seleccionadas del menu pop-up del objeto Hotel. Decidimos implementar estas vistas como programa (el cual se ejecuta como un proceso independiente) para ilustrar un punto y una técnica. El Workplace Shell esta implementado como un proceso simple y nuestras clases WPS se ejecutan dentro del proceso del Workplace Shell. Consecuentemente, las limitaciones de los recursos de nuestras clases, tanto para el manejo de ventanas y dispositivos de conteto estan basadas en las limitaciones del proceso del Workplace Shell. La técnica para evadir este problema es la de implementar una función en procesos separados donde sea posible. Pero existe un problema con esta técnica: *los metodos WPS solo pueden ser llamados desde dentro del proceso del Workplace Shell.*

Las clases del folder "Especial"

Los escenarios también muestran que cada uno de los hoteles y reservaciones tienen su propio folder contenedor. Sí, los folders son objetos. Así que, introduciremos dos nuevos objetos: **Hotelfldr y Resvfldr.** ¿Por qué no solo usamos la clase estandar WPFolder? O, mejor aún, ¿Por qué no solo dejamos que los usuarios del Sistema de Reservación de Hoteles creen sus propios folders al arrancarlos de una plantilla estandar?

La respuesta es que queremos un "folder especial". Tendrá un parecido a los folders ordinarios del Workplace Shell, pero este podrá realizar algunas extrañas cosas para nosotros. Por ejemplo, queremos memoria persistente en el Servidor de Transacciones del Sistema de Reservación de Hoteles -no el sistema de archivos locales. Este folder especial deberá propagarse por sí solo automáticamente con objetos (o registros que representen objetos) basado en el Servidor de Transacciones. Este es el porque creamos nuestros dos folders "especiales" como clases. Como se podrá intuir, la clase padre de estos folders no es otra que la clase **WPFolder.**

La clase monitor de tansacciones del Cliente: HazRPC

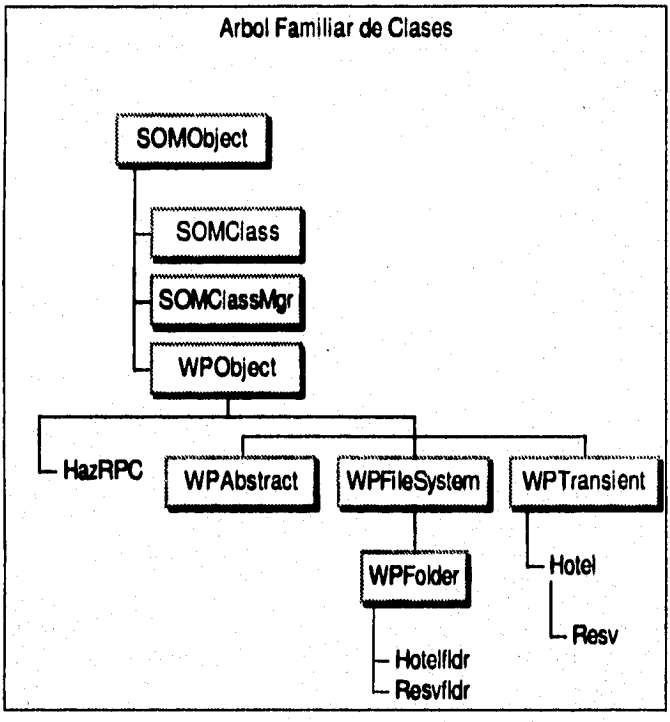
¿Qué hay acerca de **HazRPC**? Esta es la extraña clase presentada en el escritorio del Sistema de Reservación de Hoteles; es aquella sobre la que hablamos anteriormente y que usaremos para proyectos "especiales". Realmente, **HazRPC** es la clase encargada de interactuar con el Servidor. Puede pensarse en esta como un

"Monitor de Transacciones" en la estación de trabajo del Cliente. ¿Por qué necesitamos una clase como esta? ¿Por qué no mejor cada clase realiza directamente su RPC?

La respuesta esta basada en que el sistema no deberá bloquearse mientras espera por una respuesta del Servidor. Esto provocaría que la OOUI no fuera sensible a los usuarios. Hay que recordar que las OOUIs estan dirigidas a realizar muchas tareas visuales al mismo tiempo. Así que, proponemos crear una nueva corrutina cada vez que es emitida una RPC. Estas corrutinas manejarán las interacciones con el Servidor mientras que las otras clases se mentienen interactuando con el usuario. Así todos estarán felices. Por lo anterior, **HazRPC** es la clase de objeto "bien conocido" que acepta solicitudes de RPC y emite corrutinas. ¿Qué hacen las corrutinas con la respuesta del Servidor? Definiremos un protocolo que le informe a la corrutina a cual método u objeto llamar cuando se esta devolviendo la respuesta. La corrutina llama entonces al método y después muere.

La gerarquía de clases del Sistema de Reservación de Hoteles

La siguiente figura muestra todas las clases del Sistema de Reservación de Hoteles y su relación con las clases SOM/WPS.



4.4.4 Analizando las Clases

En esta parte analizaremos las cinco clases SOM/WPS del Sistema de Reservacion de Hoteles: **Hotel, Hotelfldr, HazRPC, Resv** y **Resvfldr** así como el programa **VeHotel**. Explicaremos el funcionamiento de las clases a través de sus OIDLs. El OIDL es el contrato entre la clase proveedora y el resto del mundo.

La Clase Hotelfldr

El OIDL para la clase **Hotelfldr** muestra las modificaciones a tres de los métodos de la clase **WPFolder** del SOM/WPS y la introducción de un nuevo método.

En seguida se presenta una breve descripción de lo que hacen los cuatro métodos:

- ***wpclsQueryStyle*** regresa un estilo de default para la clase **Hotelfldr.** En nuestro caso, tenemos deshabilitada la creación de plantillas.

- ***wpclsQueryTitle*** regresa el título de default para la clase **Hotelfldr.**

- ***poblaFolder*** y ***wpPopulate*** trabajan como un par para implementar la población del folder **Hotelfldr** con objetos **Hotel**. He aqui una secuencia de eventos:
  1. **wpPopulate,** como una modificación a la clase **WPFolder,** es invocada cuando el folder es abierto. La población del folder puede ser un proceso de varios pasos; probamos las banderas del folder para poblarlo solo en su primer apertura.

  2. Encontramos entonces la instancia **HazRPC**, se prepara la transacción OBTEN_NOMBRE_HOTEL y comienza la transacción al invocar el método **HazRPC**, **Ejecuta_Transaccion**. Los parámetros de **Ejecuta_Transaccion** indican que **poblaFolder** para la instancia **Hotelfldr** debería ser invocada después de que la transacción se ha completado.

  3. Cuando se ejecuta **poblaFolder,** este encuentra el identificador del folder para la instancia **Hotelfldr,** encuentra la clase **Hotel** y entonces pobla el folder con objetos **Hotel** usando el método SOM/WPS **wpclsNew.** Hay que notar que hemos pasado una cadena de actualización a cada instancia de la clase **Hotel** para adaptarle su posición del folder y pasarle el nombre del hotel y el país a la instancia. Cada objeto Hotel utiliza la cadena de actualización para inicializar sus datos.

La clase Hotel

En el OIDL de la clase **Hotel** se modificaron diez métodos de **WPTransient** SOM/WPS y se introdujeron cinco nuevos métodos.

En seguida se presenta una descripción de lo que los métodos realizan:

- *wpclsQueryModuleHandle* regresa el manejo del módulo para una clase específica. El manejo del módulo se necesita cuando accesamos los recursos atados a nuestro DLL del objeto. Los recursos de iconos y páginas de diálogo estan atadas al DLL del objeto **Hotel**. Se podrá observar como usamos este método al especificar el icono de clase y las páginas agregadas a las definiciones del cuaderno. La clase **Resv** también necesita la función que este método provee, pero no necesita implemetar el método porque **Resv** hereda el método de la clase **Hotel**.

- *wpclsQueryStyle* regresa el estilo de default para la clase **Hotelfldr**. En nuestro caso, tenemos deshabilitada la función de creación de plantillas y deshabilitado también el copiar, mover, renombrar y borrar objetos **Hotel**. Hemos hecho esto, para que los objetos **Hotel** siempre representen a aquellos objetos almacenados en nuestra **Base** de Datos.

- *wpclsQueryTitle* regresa el título de default de la clase **Hotel**.

- *wpclsQueryIconData* regresa un icono de default para la clase **Hotel**. En este caso, el icono esta atado al DLL del objeto, y usamos **wpclsQueryModuleHandle** para encontrar el manejador del módulo del DLL. **Resv** también hace uso de este método por la herencia.

- *wpclsQueryDefaultView* especifica la vista que es invocada cuando el usuario hace un doble click en nuestro icono de Hotel. Nosotros especificamos la vista Visual del Hotel como la vista de default.

- *wpAddSettingsPages* es modificada para agregar nuestra página de Precios del Hotel al cuaderno de actualizaciones del objeto. Nuestro método de ejemplo, *AgregaHotelPreciosPagina*, hace todo el trabajo. Sin embargo, en este caso, es una simple tarea, requiere solamente la inicialización de una estructura INFOPAGINA. Nótese que la estructura INFOPAGINA es inicializada con infoemación de estilo, el procedimiento de diálogo para esta página, un recurso manejador de modulo y su ID, un título y un nombre de archivo de ayuda con su ID de recurso. También hay que notar que el apuntador de ejemplo **somSelf**, es pasado como el valor de pCreateParms. Este apuntador proporciona una liga entre el aislamiento relativo de un procedimiento de diálogo del PM con su objeto asociado. Esta liga es usada para accesar la información de ejemplo del objeto.

- *wpModifyPopupMenu*, *wpMenuItemSelected* y *wpOpen* son modificados para agregarles las vistas Visual de Hotel, Descripción de Hotel y Reservación de Hotel y su función asociada al menu popup del objeto **Hotel**. Hay que notar que **wpModifyPopupMenu** llama a **wpInsertMenuItems** para agregar las opciones

del menu a la selección de apertura en el menu popup. La selcción de una opción del menu por un usuario provoca que sea invocado el método **wpMenuItemSelected**. Si el método es uno de nuestras nuevas vistas, el método **wpOpen** es llamado. Nuestra modificación a este método inicia con las transacciones para las vistas Visual de Hotel y Descripción de Hotel. Si la vista de Reservación de Hotel es selecionada, un folder **Resvfldr** es creado. En caso de que se quiera saber sobre la Vista de Definiciones, la respuesta es que no se requiere intervención alguna para esta vista. Es desplegada a través de nuestra clase padre **WPTransient**, incluyendo nuestra página de cuaderno de definiciones de Precios de Hotel. Como veremos posteriormente, tenemos un procedimiento de diálogo del PM que implementa la función de nuestra página de cuaderno de definiciones de Precios de Hotel. En este procedimiento de diálogo es también donde se inicia la transacción para obtener la información de precios de Hotel.

- Los métodos ***muestraVisualHotel***, ***muestraDescripcionHotel*** y ***muestraPreciosHotel*** son invocados al completarse las transacciones OBTEN_INFO_HOTEL, OBTEN_DESCRIPCION_HOTEL y OBTEN_PRECIOS_HOTEL, respectivamente. Los métodos **muestraVisualHotel** y **muestraDescripcionHotel** inician el programa VeHotel y le pasan los resultados de la transacción Los resultados de la transacción para **muestraPreciosHotel** deben ser finalmente desplegados por la página de cuaderno de definiciones de Precios de Hotel. Esto se hace al pegarle un mensaje al procedimiento de diálogo para la vista Precios de Hotel notificandole sobre la terminación de la transacción.

- ***wpSetup*** es el último de nuestra lista de métodos implementados en nuestra clase **Hotel**. Este método actualiza las variables de ejemplo *nombrehotel* y *nombreciudad* de los valores de la cadena de definición pasada al momento de que el ejemplo del objeto es creado. **wpScanSetupString** es usado para obtener los valores de las palabras clave.

El procedimiento **HotelPreciosProcDlg** implementa la función de la página de diálogo de Precios de Hotel. La página de cuaderno de Precios de Hotel implementa su función al interceptar tres mensajes del PM. Estos mensajes son descritos a continuación:

- **WM_INITDLG** es recibido al momento en que la página de cuaderno es inicializada. Nosotros inicializamos la transacción OBTEN_PRECIOS_HOTEL desde este punto. Nótese que el apuntador de ejemplo *somSelf* es obtenido del parametro mensaje mp2 del PM. Aquí es donde el parámetro creado, iniclalizado en la estructura PAGINAINFO de **AgregaHotelPreciosPagina**, termina en este extraño mundo del PM. Con el parámetro *somSelf*, podemos accesar la información del ejemplo del objeto por la inicialización de la variable

local *somThis* y después por el acceso de la variable ejemplo por su nombre precedida de un caracter de subrayado (_). Como un ejemplo, inicializamos la variable ejemplo *preciosDlgVentana* con el siguiente enunciado:

```
_preciosDlgVentana = hwndDlg ;
```

Esta variable de ejemplo es usada para notificar al procedimiento de diálogo de la terminación de la transacción OBTEN_PRECIOS_HOTEL, nuestro siguiente punto de discución.

- **WM_LEE_HOTEL_PRECIOS** es un mensaje del PM que hemos definido para la notificación de la terminación de la transacción OBTEN_PRECIOS_HOTEL. Cuando este mensaje es recibido, la caja de la lista de Precios de Hotel es actualizada con la información de la transacción.

- El **WM_CONTROL** es interceptado para implementar las interacciones del usuario con la caja de lista que contiene los precios del hotel. Cuando un usuario selecciona una opción en la caja de lista, la cual representa una fecha de reservación y el precio para aquella fecha, la información es almacenada en una variable de ejemplo. Esta información de precios es usada para terminar la reservación.

La Clase HazRPC

En el caso de la clase **HazRPC**, se modificaron tres métodos **WPAbstract** del SOM/WPS y se introdujo uno nuevo.

En seguida se presenta una breve descripción sobre lo que hacen los métodos:

- ***Ejecuta_Transaccion*** es un método que, de acuerdo con la función **send_txn_thresd**, ejecuta la finalización en el Cliente de una transacción. Entre estas dos funciones, se ejecutan los siguientes pasos:

  1. Una estructura de parámetro de transacción es inicializada para pasar una corrutina de transacción. Esto incluye una cadena ASCIIZ para que se convierta la dirección del método. La dirección del método es la dirección del método a ser invocado cuando la transacción termina.
  2. Una corrutina de transacción es iniciada usando la rutina en C **_beginthread**.
  3. La corrutina entonces utiliza **DosCallNPipe** para pasar la transacción al Servidor y obtener los resultados de la transacción.

4. La terminación satisfactoria de la transacción es verificada. Si la transacción se termina satisfactoriamente, el método, cuya dirección determinamos en el paso 1, es invocado; en caso contrario, un mensaje de error es desplegado. Notese que una cola de mensajes fue establecida para esta corrutina después de la ejecución de **DosCallNPIpe**. Sin esta cola de mensajes, nuestro mensaje de error y cualquier mensaje desplegado en los métodos de clase invocados desde esta corrutina no podrían ser vistos.

- Los métodos **wpSetup**, **wpSaveState** y **wpRestoreState** son métodos **WPAbstract** que son modificados para implementar la información de configuración persistente para la clase **HazRPC**. Esta información incluye el nombre de la Server Pipe de Hotel, el nombre de la computadora donde se localiza el Servidor y la trayectoria que se puede seguir para encontrar los archivos de respuesta de la transacción. Los valores para esta información de configuración son pasados inicialmente en una cadena de definiciones cuando el objeto **HazRPC** es creado. El método **wpScanSetupString** es usado en **wpSetup** para buscar la cadena de definiciones e inicializar las variables ejemplo. Los valores son retenidos en las variables ejemplo tanto tiempo como la computadora este activa, pero después de la siguiente reinicialización del sistema, los valores se pierden.

La información de ejemplo es dejada permanentemente disponible al ser almacenada en memoria persistente, usando el método **wpSaveString**. Se puede llamar a este método cuando en la modificación de **wpSaveState** es llamado por el Workplace Shell como el objeto que esta inactivo o cuando el sistema se esta dando de baja. Durante la ejecución del método **wpRestoreState**, el cual es llamado cuando el objeto es activado nuevamente, se puede llamar al método **wpRestoreString** para restablecer los valores de la información de ejemplo de los valores almacenados antes en memoria persistente.

El programa VeHotel

El programa VeHotel esta constituido de tres partes, las cuales se describen brevemente a continuación:

- Un procedimiento *main* interpreta los parámetros y despliega vistas basado en los parámetros proporcionados. El nombre de un archivo de respuesta de transacción le es proporcionado como un parámetro y los nombres de los archivos bitmap, metafile y de texto son obtenidos de este archivo. El procedimiento también establece las bases del PM (como por ejemplo, una cola de mensajes) para este proceso.

- Los procedimientos ***DespliegaVistaVentana*** y ***VistaVentanaProc*** crean un marco de ventana. El area del Cliente de esta ventana es el "lienzo" donde los bitmaps, metafiles y archivos de texto seran "pintados".
- Los procedimientos ***DespliegaMetafile***, ***MetafileProc***, ***DespliegaBitmap***, ***BitmapProc*** y ***DespliegaTexto*** implementan la función del PM de pintado de metafiles, bitmaps y archivos de texto.

4.4.5 El Servidor

El Servidor de Transacciones es simplemente una computadora dedicada en la red; este Servidor provee cierto código el cual ejecuta las transacciones sobre la Base de Datos. El código de las transacciones y la Base de Datos residen en la misma máquina. La capacidad multiusuario es proporcionada al crear casos múltiples del proceso Servidor de Transacciones. El Cliente invoca la ejecución de las transacciones en el Servidor a través de los mensajes de requerimientos enviados sobre la red usando Named Pipes. Para cada requerimiento, un mensaje de respuesta es enviado de regreso utilizando el mismo caso de Named Pipes. Los archivos grandes de resultados son enviados de regreso también a través de LAN Server.

Los elementos del servidor.

El propósito de esta parte es el de comentar acerca de los diferentes componentes del Servidor de Transacciones del Sistema de Reservación de Hoteles, y mostrar así cómo interactúan juntos.

¿Por qué es necesario un Servidor Multiusuario?

Evidentemente, no deseamos que nuestros Clientes dependan de un solo proceso Servidor. Si hacemos esto, corremos el riesgo de tener un Cliente acaparando todos los recursos del sistema y privando de estos a los demás Clientes. Un Servidor debe ser designado para proveer el mejor desempeño global para todos sus Clientes. Consecuentemente, los procesos del Servidor deberán proporcionar un cierto nivel de multitarea y soportar más de un Cliente al mismo tiempo.

¿Qué necesita un Servidor Multiusuario?

Ahora que ya hemos introducido la noción de un Servidor multitarea en lugar de un Servidor reusable serialmente, abriremos la llave de nuevos requerimientos. Todos estos requerimientos son para compartir los recursos del Servidor. En seguida se muestra la lista de lo que se necesita:

1. **Comunicaciones de Multisesión Punto a Punto:** Obviamente, no podemos tener una simple sesión de comunicación, porque se convertirá rápidamente en un cuello de botella. Así que, necesitamos un protocolo que soporte sesiones múltiples concurrentes. ¿Cuantas? Esto depende de si las sesiones son persistentes o no persistentes. Si son persistentes, se necesitará una sesión por Cliente. Si no son persistentes, se podrá crear una pila de sesiones reusables. Tomaremos el diseño de tipo no persistente, el cual nos permite proveer un servicio sobre demanda y utilizar las reglas de los promedios estadísticos. Esto significa que proporcionaremos menos sesiones que el número de Clientes. Así pués, un Cliente podrá ocasionalmente esperar a que se libere una sesión reusable. Desde luego, no definimos un límite superior en el número de Clientes que nuestra pila del Servidor pueda soportar. El concepto de alojamiento dinámico de conexiones, cuando es necesario, hace a la red más fácil de manejar que con alojamientos estáticos de sesiones. El protocolo Named Pipes nos permite crear un solo Named Pipe con múltiples casos. Los servicios de transacciones Named Pipe hacen fácil el crear llamadas de servicio no persistentes.

2. **Procesos del Servidor de Transacciones Multitarea:** El proceso del Servidor de Transacciones se relaciona con el Cliente y ejecuta transacciones de SQL. ¿ Por qué es un proceso y no un corrutina? Porque el Administrador de Base de Datos de OS/2 permite solamente una conexión a la Base de Datos por proceso. ¿Cuántos procesos son necesarios? Tantos como el número de sesiones concurrentes que se haya elegido para dar soporte. En nuestro diseño, cada proceso del Servidor de Transacciones poseerá un caso de Named Pipe y una conexión al Administrador de Bases de Datos.

3. **Acceso Compartido a los Datos:** Nuestro Servidor necesita proteger la integridad de los datos compartidos mientras permite acceso concurrente a la información. Nuestro Servidor de Transacciones esta construido por encima del Administrador de Bases de Datos de OS/2, el cual es una máquina Servidor de SQL multiusuario.

4. **Acceso a Archivos Compartidos:** Nuestro Servidor necesita soportar el regreso de grandes archivos de resultados a sus Clientes. Estos archivos de resultados

pueden contener múltiples renglones de tablas u objetos largos, tales como bitmaps, metafiles, archivos de texto o marcos de imágenes. ¿Cómo son pasados estos objetos al Cliente? Los servidores producen los objetos de archivos largos y regresan su nombre y ubicación a los Clientes como parte de los mensajes de respuesta. Los Clientes pueden entonces manipular directamente los archivos o copiarlos. Un Servidor de archivos, como el programa LAN Server, es necesario para proporcionar a múltiples Clientes y a los procesos del Servidor en la red un acceso a los archivos compartidos del sistema.

¿ Cómo inicia el proceso de transacciones y como muere ?

Existen muchos esquemas para iniciar el proceso dinámicamente y así satisfacer los requerimientos realizados por el Cliente. Cada esquema introduce su propio conjunto heurístico interesante. Sin embargo, hay que recordar que toma tiempo el crear un proceso. ¿ Se quiere que esto pase cuando un requerimiento llega ? Sería más adecuado tener un número estático de procesos. Si los procesos utilizan mucho espacio en memoria, tendrán que compartir algunas de las funciones utilizando DLLs. Ocasionalmente, se deberá afinar el Servidor agregándole o retirándole algunos procesos de transacciones.

Nuestro objetivo en esta parte es el de establecer las funciones básicas con un simple programa lanzador de procesos. Se puede tomar este código y modificarlo de acuerdo a los requerimientos. El programa lanzador utiliza, como parámetros de entrada, los nombres de los procesos a ser lanzados y el número de veces de repetirlos. Por ejemplo, para iniciar tres procesos concurrentemente del Servidor de Transacciones se tendría:
LANZADOR *HOTELSVR* 3

donde ***HOTELSVR*** es el nombre del programa del Servidor de Transacciones del Sistema de Reservación de Hoteles. Después de que son creados los tres procesos, el lanzador se bloquea así mismo y espera por cualquier tecla a ser oprimida en la máquina del Servidor. Una tecla oprimida despierta al lanzador y provoca que elimine todos los procesos que creó produciendo así la terminación de la aplicación del Servidor.

El Servidor de Transacciones y el Servidor de Bases de Datos.

La máquina del Servidor contiene tanto los procesos del Servidor de Transacciones como al Administrador de Bases de Datos del OS/2. Los procesos del Servidor de Transacciones son casos del programa ***HOTELSVR***, sobre cuyo código iremos

caminando poco a poco. Cada proceso ***HOTELSVR*** crea un caso de un Named Pipe y se conecta a la Base de Datos ***HOTEL***. Los Clientes emiten sus transacciones al hacer llamadas ***DosCallNPipe***. Esta llamada puede ser manejada automáticamente por el primer proceso Servidor disponible. Si no hay proceso Servidor disponible, los Clientes encolarán sus requerimientos hasta que uno se libere. Todo este encolamiento es proporcionado automáticamente por Named Pipes. Hay que recordar que todos los Clientes usan la misma ocurrencia del Named Pipe.

El Servidor de Transacciones y el Servidor de Archivos.

El Servidor de archivos es el programa LAN Server, el cual también es instalado en la máquina del Servidor. Los procesos del Servidor crean grandes archivos de respuestas sobre el Servidor de archivos; los Clientes pueden entonces accesar directamente estos archivos a través de la red utilizando el programa LAN Requester que forma parte del programa LAN Server.

HOTELSVR.SQC: El Servidor de Transacciones.

Ahora que ya sabemos como juegan las piezas, estamos listos para presentar el elemento que adecua al Servidor de Transacciones: el programa ***HOTELSVR.SQC***. Este es un programa diseñado para ejecutar transacciones específicas del Sistema de Reservación de Hoteles. Las transacciones operan sobre la información almacenada en la Base de Datos ***HOTEL***. Las transacciones son disparadas por mensajes enviados por Clientes remotos utilizando Named Pipes.

En seguida se presenta una lista de las características principales del programa ***HOTELSVR.SQC***:

- **El despachador de transacciones**: Esta parte del código proporcionada por la función ***atiende_transacción()*** es un switch que invoca al procedimiento apropiado para el manejo de la transacción. El switch despacha un procedimiento de manejo basado en el valor del campo "identificador de la transacción" que es leído del mensaje de requerimientos entrante.

- **Transacciones específicas**: El Servidor ejecuta las ocho transacciones descritas anteriormente. Estas transacciones son, desde luego, todas nuevas y específicas para la aplicación del sistema.

- **El mecanismo de grandes archivos de respuesta:** Esta es una parte donde el Servidor provee funciones de valor agregado.

  El mecanismo de grandes archivos de respuesta se muestra en la transacción ***obtén_nombre_hotel***. Es muy simple:

  1. Las transacciones que generan respuestas multirenglón crean un archivo de respuesta específico para el Cliente. El Servidor es responsable de nombrar el archivo y asignarles nombres únicos.
  2. El manejador de transacciones escribe los resultados de una consulta, renglón por renglón, en un archivo de respuesta. Esto se realiza como parte del ciclo del cursor SQL.
  3. El formateo de los renglones en el archivo de respuesta se base en el formato comúnmente conocido tanto para el Cliente como para el Servidor.
  4. El manejador de transacciones cierra el archivo de respuesta cuando el último renglón es seleccionado.
  5. El manejador de transacciones regresa el nombre del archivo de respuesta al Cliente como parte del mensaje de respuesta de la transacción.
  6. El Cliente puede accesar el archivo de respuesta utilizando el LAN Requester. El archivo puede ser directamente manipulado o copiado a través de la red.
  7. El Cliente es responsable de borrar el archivo de respuesta del subdirectorio compartido.

4.5 Mantenimiento.

El mantenimiento de Software es, por supuesto, mucho más que la "corrección de errores". Podemos describir el mantenimiento describiendo cuatro actividades que se llevan a cabo tras lanzar un programa.

La primera actividad de mantenimiento se da debido a que no es razonable asumir que la prueba del software haya descubierto todos los errores latentes de un gran sistema de software. Durante el uso de cualquier gran programa, se encontrarán errores, siendo informado el equipo de desarrollo. El proceso que incluye el diagnóstico y la corrección de uno o más errores se denomina *mantenimiento correctivo.*

La segunda actividad que contribuye a la definición de mantenimiento se da debido al rápido cambio inherente a todo aspecto de la informática. Se anuncian nuevas generaciones de hardware en un ciclo de 36 meses, regularmente aparecen nuevos sistemas operativos o nuevas versiones de los antiguos; y frecuentemente se mejoran o se modifican los equipos periféricos y otros elementos de sistemas. Por otro lado, la vida útil del software de aplicación puede facilmente sobrepasar los diez años, haciendose obsoleto para el entorno del sistema para el que fue orginalmente desarrollado. Por tanto el *mantenimiento adaptativo* -una actividad que modifica el software para que interaccione adecuadamente con su entorno cambiante- es tanto necesaria como corriente.

La tercera actividad que puede aplicar a la definición de mantenimiento se da cuando un paquete de software tiene éxito. A medida que se usa el software, se reciben de los usuarios recomendaciones sobre las nuevas posibilidades, sobre modificaciones de funciones ya existentes y sobre mejoras en general. Para satisfacer estas peticiones, se lleva a cabo el *mantenimiento perfectivo*. Esta actividad contabiliza la mayor cantidad de esfuerzo gastado en el mantenimiento del software.

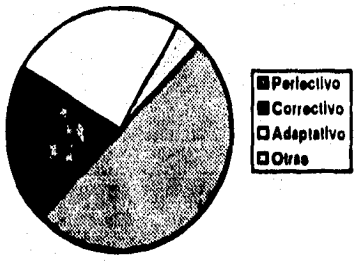
La cuarta actividad de mantenimiento se da cuando cambia el software para mejorar una futura facilidad de mantenimiento o fiabilidad o para proprocioar una base mejor para futuras mejoras. A menudo denominado *mantenimiento perfectivo*, esta actividad es todavía relativamente rara en el mundo del software.

El mantenimiento del software existente puede llevarse hasta el 60 por ciento de todo el esfuerzo gastado por una organización de desarrollo.

El software, diferencia del hardware, no se desgasta, y por lo tanto la actividad principal asociada al mantenimiento del hardware -remplazamiento de piezas estropiadas o rotas- simplemente no aplica.

Para adoptar o perfeccionar debemos determinar nuevos requerimientos, rediseñar, generar código y probar el software existente. Tradicionalmente tales tareas han sido denominadas en conjunto como mantenimiento.

De hecho, aproximiadamente la mitad de todo el mantenimiento de software es perfectivo, tal y como se ilustra en la siguiente figura.



Distribución de las actividades de mantenimiento.

El sistema de reservación de hoteles, desarrollado en este trabajo, es de fácil mantenimiento ya que, bajo un esquema cualitativo, tiene una estructura de sistema comprensible, es de fácil uso y manejo, además de que se utilizó al lenguaje de programación C que es un estándar. De igual forma, el sistema operativo empleado, asi como las técnicas y procedimientos de bases de datos y flujo de datos son un estándar en la industria de cómputo.

La técnica de desarrollo por prototipos establece al mantenimiento como un proceso dentro del ciclo de desarrollo.

Spiegel describe el proceso de prototipo:

> "La creación de prototipos de software es un proceso (el hecho, el estudio o la habilidad) de modelar los requerimientos del usuario en uno o mas niveles de detalle, incluyendo modelos operativos. Se disponen los recursos del proyecto para producir versiones que aumenten el detalle del software descrito por los requerimientos. Cada versión del prototipo hace que el software pueda ser revisado por los usuarios, los diseñadores y los gestores... Este proceso continuo, hasta que se desee, consiguiendo versiones ejecutables listas para ser lanzadas tras varias iteraciones".

Si se desarrollan varios niveles de prototipos, es posible tener un conjunto de repuestos de software que se pueden usar cuando se reciben peticiones de mantenimiento correctivo.

CONCLUSIONES.

Gracias al trabajo realizado hemos dotado a la industria hotelera mexicana con la herramienta necesaria para satisfacer los requerimientos de conectividad, productividad y rendimiento, haciéndola más eficiente y competitiva, a través de un sistema prototipo, basado en el modelo computacional Cliente/Servidor, que remplaza sistemas manuales o sistemas basados en terminales que corren en Sistemas Mayores (mainframes) o Sistemas Intermedios (superminis) por un sistema con interfaces gráficas amigables y sencillas de utilizar capaz además de realizar procesamiento intenso.

En el desarrollo de este trabajo pudimos percibir que muchos factores tecnológicos y de negocio están llegando a considerarse importantes por las empresas del ramo hotelero en la forma en que ellas escogen la solución a sus problemas de negocio. Cada factor trae consigo un requerimiento único que debe ser direccionado. En la mayoría de los negocios hoteleros, hoy en día, estamos observando la necesidad de responder rápidamente a los cambios de tal forma que se continúe siendo competitivo. Los hoteles que adopten el cambio más rápido que sus competidores llegarán a convertirse en los líderes en el mercado. Otro factor que las compañías hoteleras deben considerar para llegar a ser exitosas es el uso de la información y de la tecnología de la información que las lleve a tener ventajas competitivas. Hoy dia, la información es vista como una ventaja estratégica para la industria. En el ambiente de los negocios actuales hay una tendencia hacia las estructuras organizacionales aerodinámicas caracterizada por reducidos niveles gerenciales. Para llegar a convertir esta estrategia en exitosa, hay una necesidad creciente para dotar a todos los empleados de iniciativa que los haga analizar la información y hacerlos tomadores de desiciones.

Con este trabajo, llevamos a cabo un rápido cambio en la forma en que la industria hotelera percibe a la tecnología de la información como recurso para eficientar su negocio, al crear oportunidades para las compañías hoteleras con nuevas estrategias que respondan a las tendencias que están tomando lugar en el ambiente de negocio. El poder y la capacidad de las estaciones de trabajo se ha incrementado dramáticamente, permitiendo a una variedad mas amplia de soluciones de negocio hotelero, que se encuentran disponibles en una plataforma, a que sean fácilmente implementadas y a un bajo costo.

Dándole el uso a la información como una herramienta competitiva y dotando a los empleados con la libertad de tomar decisiones, grupos de empleados se han formado dentro de grupos de trabajo de forma que efectivamente compartan experiencia y recursos. Los grupos de trabajo están utilizando tecnologías de información para facilitar esta comunicación y este compartir recursos. Tan pronto

como los grupos de trabajo encuentren la necesidad de compartir recursos, las estaciones de trabajo inteligentes que se encuentran aisladas estan siendo combinadas dentro de sistemas de computo del grupo de trabajo para que se direccionen hacia las necesidades del grupo de trabajo entero. Muchos grupos de trabajo han seleccionado estos sistemas de cómputo para correr las aplicaciones críticas a las operaciones del negocio. Pensamos que el sistema de reservaciones es un proceso crítico dentro de la operación del negocio hotelero.

Tan pronto como el uso de las estaciones de trabajo inteligentes se llegue a difundir más y más, las empresas estarán reconociendo la necesidad de integrar estas estaciones de trabajo con todos los otros recursos disponibles en toda la compañía o departamento. Esta integración requiere de dos capacidades: una red extensa que alcance a todos los recursos en la empresa, y el soporte de un ambiente multi-provedor.

A los grupos de trabajo se les ha confiado la decisión de escoger los sistemas que más se acerquen a satisfacer sus requerimientos, y un amplio ambiente heterogeneo ha surgido dentro de la empresa. Por ello las compañías buscan por soluciones con soporte a sistemas abiertos que les permita responder mas eficientemente y puedan tomar ventaja del ambiente de negocios actual.

Este tesis forma parte de las soluciones anteriormente descritas, y demuestra la funcionalidad del modelo computacional cliente/servidor, ya que brinda al negocio hotelero de un sistema abierto que puede trabajar en un ambiente multi-vendedor (multiplataforma), el cual optima los recursos de la empresa al reducir los costos inerentes al proceso de reservaciones, proceso medular del negocio hotelero.

Este trabajo no tuvo como finalidad el brindar un producto terminado y comercializable, sino un prototipo a través del cual se pudo implementar la arquitectura cliente servidor, además de utilizar ramas de la computacion tan variadas y bastas como son los sistemas operativos, la ingeniería de programación, estructuras de datos, bases de datos, redes, etc. que permitieron crear una sinergia para lograr una solución a un problema real como lo es el eficientar el sistema de reservaciones, el cual es ampliamente utilizado por empresas que se dedican a la industria hotelera, de aeronáutica, restaurantera, de eventos, etc.

APENDICE A

Programa fuente HOTEL.C

```
/*****************************************************************************/
/* NOMBRE DEL OBJETO : HOTEL -- Una Clase para Hotel                         */
/*                                                                           */
/* DESCRIPCION       : Se trata del Objeto para el Hotel                     */
/*                                                                           */
/*****************************************************************************/
#define Hotel_Class_Source
#define M_Hotel_Class_Source
#include <string.h>
#include <stdio.h>
#include <memory.h>
#include <stdlib.h>
#include "Hotel.ih"              /* Encabezado de implementacion  */
#include "Hotel.ph"              /* Encabezado privado            */
#include "hazRPC.h"              /* Encabezado de Clase HazRPC    */
#include "Hotetxns.h"            /* Definicion de Transaccion     */
#include "Hoteldlg.h"            /* Encabezado para Dia logos     */

/*---------------------------------------------------------------------------*/
/* Variables Globales y Declaracion de Funciones para este modulo */
/*---------------------------------------------------------------------------*/
HMODULE   hmod = NULLHANDLE;
CHAR      ClassTitle[] = "Hotel";          /* Titulo de la Clase   */
CHAR      szHelpLibrary[] = "Hotel.hlp";
CHAR      pszMsg[120];

MRESULT EXPENTRY HotelPricesDlgProc(HWND hwndDlg, ULONG msg,
                    MPARAM mp1, MPARAM mp2);

/*****************************************************************************/
/* Metodo de Sobreposicion de Clase                                          */
/*****************************************************************************/
#undef SOM_CurrentClass
#define SOM_CurrentClass SOMMeta

/*---------------------------------------------------------------------------*/
/* Nuevo Metodo de Clase: clsQueryModuleHandle     */
/*---------------------------------------------------------------------------*/
SOM_Scope HMODULE  SOMLINK HotelM_clsQueryModuleHandle(M_Hotel *somSelf,
        PSZ ClassName,
        integer4 ClassMajorVersion,
        integer4 ClassMinorVersion)
{
  HMODULE   hmod = NULLHANDLE;        /* Apuntador al Modulo DLL   */
  zString   ModulePathName;           /* Nombre y Ruta del Modulo DLL*/
  APIRET    rc;
```

```
  /* M_HotelData *somThis = M_HotelGetData(somSelf); */
  M_HotelMethodDebug("M_Hotel","HotelM_clsQueryModuleHandle");

  /* ShowMsg("Hotel.c","clsQueryModuleHandle"); */

  /*-------------------------------------------------*/
  /* Obtencion del Nombre del Modulo */
  /*-------------------------------------------------*/
  ModulePathName =
  _somLocateClassFile(SOMClassMgrObject,
              SOM_IdFromString(ClassName),
              ClassMajorVersion,
              ClassMinorVersion);

  rc = DosQueryModuleHandle( ModulePathName, &hmod);
  if (rc)
  { ShowMsg(" Hotel ", "Error, Modulo no encontrado");
    return;
  }

  return (hmod);
}

/*-------------------------------------------------------------------------*/
/* OVERRIDE: _wpclsQueryStyle                                              */
/*-------------------------------------------------------------------------*/
SOM_Scope ULONG  SOMLINK HotelM_wpclsQueryStyle(M_Hotel *somSelf)
{

  /* M_HotelData *somThis = M_HotelGetData(somSelf); */
  M_HotelMethodDebug("M_Hotel","HotelM_wpclsQueryStyle");

  return (parent_wpclsQueryStyle(somSelf) | CLSSTYLE_NEVERTEMPLATE |
      CLSSTYLE_NEVERMOVE | CLSSTYLE_NEVERCOPY | CLSSTYLE_NEVERRENAME);
}

/*-------------------------------------------------------------------------*/
/* OVERRIDE: _wpclsQueryTitle                                              */
/*-------------------------------------------------------------------------*/
SOM_Scope PSZ  SOMLINK HotelM_wpclsQueryTitle(M_Hotel *somSelf)
{
  /* M_HotelData *somThis = M_HotelGetData(somSelf); */
  M_HotelMethodDebug("M_Hotel","HotelM_wpclsQueryTitle");

  return (ClassTitle);
}
/*-------------------------------------------------------------------------*/
/* OVERRIDE: _wpclsQueryIconData                                           */
/*-------------------------------------------------------------------------*/
SOM_Scope ULONG  SOMLINK HotelM_wpclsQueryIconData(M_Hotel *somSelf,
        PICONINFO pIconInfo)
{
  /* M_HotelData *somThis = M_HotelGetData(somSelf); */
  M_HotelMethodDebug("M_Hotel","HotelM_wpclsQueryIconData");
```

```
/*  ShowMsg("Hotel.c","wpclsQueryIconData");  */

  /*-------------------------------------------------*/
  /* Obtencion del Apuntador del Modulo */
  /*-------------------------------------------------*/
  hmod = _clsQueryModuleHandle(somSelf, ClassTitle, 1, 2);

  /*-----------------------------------------------------------------*/
  /* Llenado de la Estructura de Datos ICONINFO  */
  /*-----------------------------------------------------------------*/
  if (pIconInfo)
  {
    pIconInfo->fFormat = ICON_RESOURCE;
    pIconInfo->hmod   = hmod;
    pIconInfo->resid  = ID_ICON;
  }
  return (sizeof(ICONINFO));
}

/*-----------------------------------------------------------------------*/
/* OVERRIDE: _wpclsQueryDefaultView                                      */
/*-----------------------------------------------------------------------*/
SOM_Scope ULONG  SOMLINK HotelM_wpclsQueryDefaultView(M_Hotel *somSelf)
{
/*  ShowMsg("Hotel.c","wpclsQueryDefaultView");  */
  /* M_HotelData *somThis = M_HotelGetData(somSelf); */
  M_HotelMethodDebug("M_Hotel","HotelM_wpclsQueryDefaultView");

  return (OPEN_HOTELVISUAL);
}

/*************************************************************************/
/* Metodos para Nuevas Instancias                                        */
/*************************************************************************/
#undef SOM_CurrentClass
#define SOM_CurrentClass SOMInstance

/*-----------------------------------------------------------------------*/
/* Nuevo Metodo: _AddHotelPricesPage                                     */
/*-----------------------------------------------------------------------*/
SOM_Scope ULONG SOMLINK Hotel_AddHotelPricesPage(Hotel *somSelf,
        HWND hwndNotebook)
{
  PAGEINFO pageInfo;

  /* HotelData *somThis = HotelGetData(somSelf); */
  HotelMethodDebug("Hotel","Hotel_AddHotelPricesPage");

  memset((PCH)&pageInfo,0,sizeof(PAGEINFO));
  pageInfo.cb            = sizeof(PAGEINFO);
  pageInfo.hwndPage      = NULLHANDLE;
  pageInfo.usPageStyleFlags  = BKA_MAJOR;
  pageInfo.usPageInsertFlags = BKA_FIRST;
  pageInfo.pfnwp         = HotelPricesDlgProc;
  pageInfo.resid         = hmod;
```

```
  pageinfo.dlgid          = IDD_HOTELDLG;
  pageinfo.pszName        = "Hotel Precios";
  pageinfo.pCreateParams  = somSelf;
  pageinfo.idDefaultHelpPanel = 2000;
  pageinfo.pszHelpLibraryName = szHelpLibrary;

  return _wpInsertSettingsPage( somSelf, hwndNotebook, &pageinfo );
}

/*--------------------------------------------------------------------*/
/* Nuevo Metodo: showHotelPrices                                      */
/*--------------------------------------------------------------------*/
SOM_Scope BOOL  SOMLINK Hotel_showHotelPrices(Hotel *somSelf,
                        PSZ  txn_rsp_file,
                        PSZ  datafile_drive_path,
                        PSZ  txn_response)
{
  HotelData *somThis = HotelGetData(somSelf);
  HotelMethodDebug("Hotel","Hotel_showHotelPrices");

  /*-------------------------------------------------*/
  /* Interpreta la respuesta TXN, actualiza  */
  /* las variables de instancia, manda un    */
  /* mensaje al dia logo                     */
  /*-------------------------------------------------*/
  strcpy(_rsp_filename, txn_rsp_file);
  strcpy(_datafile_drive_path, datafile_drive_path);
  WinPostMsg(_pricesDlgWindow, WM_READ_HOTEL_PRICES,
        (MPARAM)0, (MPARAM)0);

  return(TRUE);
}

/*--------------------------------------------------------------------*/
/* Metodo Nuevo: showHotelVisual                                      */
/*--------------------------------------------------------------------*/
SOM_Scope BOOL  SOMLINK Hotel_showHotelVisual(Hotel *somSelf,
                        PSZ  txn_rsp_file,
                        PSZ  datafile_drive_path,
                        PSZ  txn_response)
{
  CHAR      msg[120];          /* buffer para el mensaje de error  */
  CHAR      object_name[13];   /* buffer para el objeto con error  */
  RESULTCODES ExecResult;      /* resultado del proceso hijo       */
  CHAR      program[160];      /* programa/parametros de DosExecPgm */
  ULONG     rc;                /* Codigo de Retorno de DosExecPgm  */

  HotelData *somThis = HotelGetData(somSelf);
  HotelMethodDebug("Hotel","Hotel_showHotelVisual");

  sprintf(program,"%s%c %d %s %s %s %s%c",
       "Hoteview.exe" , 0, HOTEL_GRAPHICS_VIEW,
       _Hotelname, _Hotelcountry, datafile_drive_path, txn_rsp_file, 0);
/*
  ShowMsg("Estoy en Hotel","Antes de ejecutar HOTEVIEW.EXE");
```

```
  ShowMsg("Hotelname", _Hotelname);
  ShowMsg("Hotelcountry", _Hotelcountry);
  ShowMsg("txn_response", txn_response);
  ShowMsg("txn_rsp_file", txn_rsp_file);
  ShowMsg("datafile_drive_path", datafile_drive_path ); */

/*  sprintf(program,"%s%c %d %s %s %s %s%c",
      "Hoteview.exe", 0, HOTEL_GRAPHICS_VIEW,
      "Ixtapa", "Mexico", "C:\\HOTELSVR", "R0001621.TMP", 0);*/

  rc = DosExecPgm(object_name,      /* objeto fallido si el programa falla */
        sizeof(object_name),  /* tamanio del objeto          */
        EXEC_ASYNC,           /* sincronizacion de la ejecucion del programa    */
        program,          /* nombre del programa y parametros   */
        NULL,             /* medioambiente inherente          */
        &ExecResult,        /* codigo de terminacion       */
        program);         /* nombre del programa          */

  if (rc)
  { sprintf(msg, "DosExecPgm RC=%d\n<%s>", rc, program);
    ShowMsg(" Hotel ", msg);
  } /* endif */

  return(TRUE);
}

/*-------------------------------------------------------------------*/
/* Metodo Nuevo: showHotelDescription                                */
/*-------------------------------------------------------------------*/
SOM_Scope BOOL  SOMLINK Hotel_showHotelDescription(Hotel *somSelf,
                        PSZ  txn_rsp_file,
                        PSZ  datafile_drive_path,
                        PSZ  txn_response)
{
  CHAR      msg[120];         /* buffer para el mensaje de error  */
  CHAR      object_name[13];      /* buffer para el objeto con error    */
  RESULTCODES ExecResult;         /* resultado del proceso hijo     */
  CHAR      program[160];       /* programa/parametros de DosExecPgm */
  ULONG     rc;             /* Codigo de Retorno de DosExecPgm  */

  HotelData *somThis = HotelGetData(somSelf);
  HotelMethodDebug("Hotel","Hotel_showHotelDescription");

  sprintf(program,"%s%c %d %s %s %s %s%c",
      "Hoteview.exe", 0, HOTEL_TEXT_VIEW,
      _Hotelname, _Hotelcountry, datafile_drive_path, txn_rsp_file, 0);

/*
  ShowMsg("Estoy en Hotel","Antes de ejecutar HOTEVIEW.EXE");
  ShowMsg("Hotelname", _Hotelname);
  ShowMsg("Hotelcountry", _Hotelcountry);
  ShowMsg("txn_response", txn_response);
  ShowMsg("txn_rsp_file", txn_rsp_file);
  ShowMsg("datafile_drive_path", datafile_drive_path );
*/
```

```
/*  sprintf(program,"%s%c %d %s %s %s %s%c",
        "Hoteview.exe", 0, HOTEL_GRAPHICS_VIEW,
        "Ixtapa", "Mexico", "C:\\HOTELSVR", "R0001621.TMP", 0); */

   rc = DosExecPgm(object_name,       /* objeto fallido si el programa falla */
           sizeof(object_name),   /* el tamanio del objeto          */
           EXEC_ASYNC,          /* ejecucion del programa de sincronizacion    */
           program,           /* nombre y parametros del programa   */
           NULL,              /* medio ambiente inherente       */
           &ExecResult,         /*codigo de terminacion      */
           program);          /* nombre de programa             */

   if (rc)
   { sprintf(msg, "DosExecPgm RC=%d\n<%s>", rc, program);
     ShowMsg(" Hotel ", msg);
   } /* endif */

   return(TRUE);
}

/*-------------------------------------------------------------------------*/
/* Nuevo Metodo: getSelectedPriceData                                      */
/*-------------------------------------------------------------------------*/
SOM_Scope PSZ   SOMLINK Hotel_getSelectedPriceData(Hotel *somSelf)
{
   HotelData *somThis = HotelGetData(somSelf);
   HotelMethodDebug("Hotel","Hotel_getSelectedPriceData");

   return(_selected_price_data);
}
/*************************************************************************/
/* Overrides a metodos ejemplo                                           */
/*************************************************************************/
/*-------------------------------------------------------------------------*/
/* Nuevo Metodo: _AddResvConfirmPage                                       */
/*-------------------------------------------------------------------------*/
SOM_Scope BOOL  SOMLINK Hotel_wpAddSettingsPages(Hotel *somSelf,
          HWND hwndNotebook)
{
   /* HotelData *somThis = HotelGetData(somSelf); */
   HotelMethodDebug("Hotel","Hotel_wpAddSettingsPages");

   if (_AddHotelPricesPage(somSelf, hwndNotebook))
     return(TRUE);
   else
   {
     ShowMsg("Hotel_wpAddSettingsPages", " Falla al agregar una pagina de definiciones.");
     return( FALSE );
   }
}

/*-------------------------------------------------------------------------*/
/* OVERRIDE: _wpModifyPopupMenu                                            */
```

```
/*-----------------------------------------------------------------*/
SOM_Scope BOOL  SOMLINK Hotel_wpModifyPopupMenu(Hotel *somSelf,
        HWND hwndMenu,
        HWND hwndCnr,
        ULONG iPosition)
{
  HotelData *somThis = HotelGetData(somSelf);
  HotelMethodDebug("Hotel","Hotel_wpModifyPopupMenu");


  ShowMsg("Estoy en Hotel","Estoy en el procedimiento _wpModifyPopupMenu");
   if (!_wpInsertPopupMenuItems(somSelf, hwndMenu, 1,
                  hmod, ID_OPENMENU, WPMENUID_OPEN))
    { ShowMsg(" HotelWPS ", "ERROR: _wpInsertPopupMenuItems");
     return(FALSE);
    }
  return (parent_wpModifyPopupMenu(somSelf,hwndMenu,hwndCnr,iPosition));
}


/*-----------------------------------------------------------------*/
/* OVERRIDE: _wpMenuItemSelected                                   */
/*-----------------------------------------------------------------*/
SOM_Scope BOOL  SOMLINK Hotel_wpMenuItemSelected(Hotel *somSelf,
        HWND hwndFrame,
        ULONG ulMenuId)
{
  /* HotelData *somThis = HotelGetData(somSelf); */
  HotelMethodDebug("Hotel","Hotel_wpMenuItemSelected");


  ShowMsg("Estoy en Hotel","Estoy en el procedimiento _wpMenuItemSelected");

  switch( ulMenuId )
  {

    case OPEN_HOTELVISUAL:
    case OPEN_HOTELDESCRIPTION:
    case OPEN_HOTELRESERVATION:
      _wpOpen(somSelf, NULLHANDLE, ulMenuId, 0);
      break;

    default:
      return parent_wpMenuItemSelected(somSelf, hwndFrame, ulMenuId);
      break;
  }
}

/*-----------------------------------------------------------------*/
/* OVERRIDE: _wpopen                                               */
/*-----------------------------------------------------------------*/
SOM_Scope HWND  SOMLINK Hotel_wpOpen(Hotel *somSelf,
        HWND hwndCnr,
        ULONG ulView,
        ULONG param)
```

```
{
  WPObject *Folder;
  SOMANY  *HazRPCClass;
  SOMANY  *HazRPCObject;
  SOMAny  *ResvfldrClass;
  SOMAny  *ResvfldrObject;
  CHAR    objectid[64];
  SOMANY  *NewObj;
  CHAR    ClientMethod[32];
  CHAR    txn_string[80];
  BOOL    rc;
  CHAR    SetupString[256];

  HotelData *somThis = HotelGetData(somSelf);
  HotelMethodDebug("Hotel","Hotel_wpOpen");


 ShowMsg("Estoy en Hotel","Estoy en  _wpOpen");

  /*-------------------------------------------------*/
  /* Preparacion para la llamada RPC */
  /*-------------------------------------------------*/
  HazRPCClass = _somFindClass(SOMClassMgrObject,
                SOM_IdFromString("HazRPC"), 1,2);
  if (HazRPCClass == NULL)
    { ShowMsg(" Hotelfldr ", "Error: _somFindClass HazRPC");
      return(NULLHANDLE);
    }
  HazRPCObject = _wpclsQueryObject(HazRPCClass,
                    WinQueryObject("<HazRPC>"));
  if (HazRPCObject == NULL)
    { ShowMsg(" Hotelfldr ", "Error: _wpclsQueryObject HazRPC");
      return(NULLHANDLE);
    }

  /*-------------------------------------------------*/
  /* Preparacion de la transaccion basado en la */
  /* vista elegida                                   */
  /*-------------------------------------------------*/
  switch (ulView)
  {
    case OPEN_HOTELVISUAL:
/*        strcpy(_Hotelname, "Ixtapa"); */
      sprintf(txn_string, "TRANSACTION_ID=%d HOTEL=%s",
                  GET_HOTEL_GRAPHICS, _Hotelname);
      strcpy(ClientMethod, "ShowHotelVisual");

/*
      ShowMsg("Estoy en Hotel_wpOpen","Open_HOTELVISUAL");
      ShowMsg("Hotelname", _Hotelname);
      ShowMsg("Hotelcountry", _Hotelcountry);
      ShowMsg("txn_string", txn_string);
*/
```

```
        rc = _Execute_Transaction(HazRPCObject,
                        txn_string,
                        somSelf,
                        "ShowHotelVisual");
/*
        ShowMsg("Estoy en Hotel_wpOpen","Open_HOTELVISUAL");
        ShowMsg("Hotelname", _Hotelname);
        ShowMsg("Hotelcountry", _Hotelcountry);
        ShowMsg("txn_string", txn_string);
*/

        return(NULLHANDLE);
        break;

    case OPEN_HOTELDESCRIPTION:
        sprintf(txn_string, "TRANSACTION_ID=%d HOTEL=%s",
                    GET_HOTEL_DESCRIPTION, _Hotelname);
        ShowMsg("Estoy en Hotel_wpOpen","Open_HOTELDESCRIPTION");
        ShowMsg("Hotelname", _Hotelname);
        ShowMsg("Hotelcountry", _Hotelcountry);
        ShowMsg("txn_string", txn_string);

        rc = _Execute_Transaction(HazRPCObject,
                        txn_string,
                        somSelf,
                        "ShowHotelDescription");
        return(NULLHANDLE);
        break;

    case OPEN_HOTELRESERVATION:
        ShowMsg("Estoy en Hotel","Estoy en case OPEN_HOTELRESERVATION");
        Folder = _wpclsQueryFolder(_somGetClass(somSelf),
                        "<WP_DESKTOP>",
                        TRUE);
        if (Folder == NULL)
          { ShowMsg(" Hotel ", "Error al tratar de encontrar el ID del folder del desktop");
            return(FALSE);
          }

        ResvfldrClass = _somFindClass(SOMClassMgrObject,
                        SOM_IdFromString("Resvfldr"), 1,2);
        if (ResvfldrClass == NULL)
          { ShowMsg(" Hotelfldr ", "Error: _somFindClass: Resvfldr");
            return(FALSE);
          }
        sprintf(objectid,"<%s, %s>", _Hotelname, _Hotelcountry);
        ResvfldrObject = _wpclsQueryObject(ResvfldrClass,
                            WinQueryObject(objectid));
        if (ResvfldrObject == NULL)
          /*-------------------------------------------------------*/
          /* Crear Nuevo folder de Reservaciones */
          /*-------------------------------------------------------*/
          {
            ShowMsg("Estoy en Hotel","Agregar nuevo folder reservacion y objeto");
            sprintf(SetupString,
```

```
                "TITLE=%s, %s;OPEN=DEFAULT;OBJECTID=<%s, %s>;"
                "HELPLIBRARY=RESV.HLP;HELPPANEL=3000;"
                "ICONRESOURCE=100,RESVFLDR;HOTEL=%s;COUNTRY=%s;",
                _Hotelname, _Hotelcountry, _Hotelname, _Hotelcountry,
                _Hotelname, _Hotelcountry);

          NewObj = _wpclsNew(ResvfldrClass, "Resvfldr",
                SetupString, Folder, FALSE);
          if (NewObj == NULL)
            { ShowMsg(" Hotel ", "Error Creando el Folder Reservacion");
              return(FALSE);
            }
        }
       else
         {

           /*-------------------------------------------------*/
           /* Reimprime el folder existente                   */
           /*-------------------------------------------------*/
           ShowMsg("Estoy en Hotel","El folder ya existe reservacion");
           _wpOpen(ResvfldrObject, NULLHANDLE, OPEN_DEFAULT, 0);
         }
       break;

    default:
       return(parent_wpOpen(somSelf,hwndCnr,ulView,param));
   }
}

/*-------------------------------------------------------------------*/
/* OVERRIDE: _wpSetup                                                */
/*-------------------------------------------------------------------*/
SOM_Scope BOOL  SOMLINK Hotel_wpSetup(Hotel *somSelf,
        PSZ pszSetupString)
{
  ULONG stringlen=32;
  BOOL freturn;

  HotelData *somThis = HotelGetData(somSelf);
  HotelMethodDebug("Hotel","Hotel_wpSetup");

/*
  ShowMsg("Estoy en Hotel.c", "_wpSetup");
  ShowMsg("pszSetupString",pszSetupString);
*/


 /* stringlen=20; */
  freturn= _wpScanSetupString(somSelf, pszSetupString, "COUNTRY",
             _Hotelcountry, &stringlen);

  stringlen=32;
  freturn = _wpScanSetupString(somSelf, pszSetupString, "HOTELS",
             _Hotelname, &stringlen);
```

```
/*
  ShowMsg("Antes de salir de _wpSetup", "Anallcemos...");
  ShowMsg("_Hotelname",_Hotelname);
  ShowMsg("_Hotelcountry",_Hotelcountry);
  ShowMsg("pszSetupString",pszSetupString);

*/

  return (parent_wpSetup(somSelf,pszSetupString));
}


/*****************************************************************************/
/* Procedimientos de Dialogo                                                 */
/*****************************************************************************/
MRESULT EXPENTRY HotelPricesDlgProc(HWND hwndDlg, ULONG msg,
                    MPARAM mp1, MPARAM mp2)
{
Hotel   *somSelf;              /* Apuntador de ejmplo              */
HotelData *somThis;            /* Apuntador de datos de ejemplo    */

SOMAny  *HazRPCClass;          /* Apuntador de la clase HazRPC     */
SOMANY  *HazRPCObject;
CHAR    txn_string[80];        /* usado para construir el mensaje de Txn */
BOOL    rc;                    /* codigo de retorno                */
FILE    *stream;               /* manejador del archivo de respuestas de txn */
ULONG   i;                     /* variable de indice               */
CHAR    Hotelprice[PRICEDATALEN];  /* usada para leer datos de archivo */
SHORT   LboxSelection;         /* indice de seleccion              */
CHAR    filename[CCHMAXPATH];

switch (msg)
{
  case WM_INITDLG:
    /*-------------------------------------------------------*/
    /* Uso de instrucciones de Windows para */
    /* salvar el apuntador de ejemplo       */
    /*-------------------------------------------------------*/
    WinSetWindowPtr(hwndDlg, QWL_USER, (PVOID)mp2);
    somSelf = (Hotel *)mp2;
    somThis = HotelGetData(somSelf);

    HazRPCClass = _somFindClass(SOMClassMgrObject,
                  SOM_IdFromString("HazRPC"), 1,2);
    if (HazRPCClass == NULL)
      { ShowMsg(" Hotelfldr ", "Error: _somFindClass HazRPC");
        return(FALSE);
      }
    HazRPCObject = _wpclsQueryObject(HazRPCClass,
                    WinQueryObject("<HazRPC>"));
    If (HazRPCObject == NULL)
      { ShowMsg(" Hotelfldr ", "Error: _wpclsQueryObject HazRPC");
        return(FALSE);
      }
```

```
_pricesDlgWindow = hwndDlg;
sprintf(txn_string, "TRANSACTION_ID=%d HOTEL=%s",
            GET_HOTEL_PRICES, _Hotelname);
rc = _Execute_Transaction(HazRPCObject,
                txn_string,
                somSelf,
                "ShowHotelPrices");
if (!rc)
  { ShowMsg("Hotel","Error: Ejecutando la transaccion GET_HOTEL_PRICES ");
    return(FALSE);
  }
return (MRESULT) TRUE;
break;

case WM_READ_HOTEL_PRICES:
 /*-------------------------------------------------------------*/
 /* Obtener acceso a los datos de ejemplo        */
 /*-------------------------------------------------------------*/
 somSelf = WinQueryWindowPtr(hwndDlg, QWL_USER);
 if (somSelf == NULL)
 { ShowMsg("HotelPricesDlgProc", "No pudo ejecutar instrucciones de window");
   break;
 }
 somThis = HotelGetData(somSelf);

 /*-------------------------------------------------------------*/
 /* Llena la caja de lista con datos de Txn    */
 /*-------------------------------------------------------------*/
 sprintf(filename,"%s\\%s",_datafile_drive_path,_rsp_filename);
 stream = fopen(filename,"r");
 if (stream == NULL)
   { ShowMsg(" Hotelfldr Error: Abriendo el Archivo de Precios del Hotel", filename);
     return(FALSE);
   }

 do
 { if (fgets(Hotelprice,PRICEDATALEN,stream) == 0) break;
    Hotelprice[strlen(Hotelprice)-1] = 0;
    WinInsertLboxItem(WinWindowFromID(hwndDlg,ID_PRICELIST),
              LIT_END,
              Hotelprice);
 } while (TRUE); /* enddo */
 fclose(stream);
 remove(filename);
 break;

case WM_CONTROL:
 /*-------------------------------------------------------------*/
 /* Obtener acceso a los datos de ejemplo        */
 /*-------------------------------------------------------------*/
 somSelf = WinQueryWindowPtr(hwndDlg, QWL_USER);
 if (somSelf == NULL)
 { ShowMsg("HotelPricesDlgProc", "No pudo ejecutar las instrucciones de window");
   break;
 }
```

```
    somThis = HotelGetData(somSelf);


    /*-------------------------------------------------------------------*/
    /* Mensajes de seleccion del manejador de la caja de lista   */
    /*-------------------------------------------------------------------*/
    switch (SHORT2FROMMP(mp1))
    {
     /*-----------------------------------------------------*/
     /* Actualiza datos de precios cuando una   */
     /* opcion de la caja de lista es seleccionada */
     /*-----------------------------------------------------*/
     case LN_SELECT:
       LboxSelection =
         (USHORT)WinSendDlgItemMsg( hwndDlg,
              ID_PRICELIST,
              LM_QUERYSELECTION,
              (MPARAM)0,
              (MPARAM)0);
       WinSendDlgItemMsg( hwndDlg,
              ID_PRICELIST,
              LM_QUERYITEMTEXT,
              MPFROM2SHORT(LboxSelection, PRICEDATALEN),
              (MPARAM)_selected_price_data);
       break;
    }  /* end switch (SHORT2FROMMP(mp1)) */

   return (MRESULT) TRUE;
   break;

}  /* end switch(msg) */

return (WinDefDlgProc(hwndDlg, msg, mp1, mp2) );

}  /* end HotelDlgProc() */
```

Bibliografia

Stegenga, Jerry; Jollans, Adam
"OS/2 2.11 Power Techniques"
Editorial: QUE
USA, 1994

Young, Michel
"Programmer's Guide to OS/2"
Editorial: SYBEX
USA, 1988

Orfalli, Robert; Harkey, Dan
"Client Server Programing with OS/2 2.1"
Editorial: VNR
USA, 1993

Pressman, Roger
"Ingeniería del Software. Un Enfoque Práctico"
Editorial: McGraw Hill
México, 1990

Korth, Henry; Silberschatz, Abraham
"Fundamentos de Bases de Datos"
Editorial: McGraw Hill
México, 1990

Weizman, Dov
"Client/Server Foundations. The Changing Times"
Editorial: Argo Technologies, Inc.
USA, 1994

Weizman, Dov
"Client/Server Foundations. Competition"
Editorial: Argo Technologies, Inc.
USA, 1994

Supplement to Datamation
"Client/Server: Costs Vs. Benefits"
Junio 15, 1995
USA

Supplement to Datamation
"Client/Server: The Bussines Advantage"
Junio 15, 1994
USA

Supplement to Datamation
"Answering Corporate Needs With Client/Server Cumputing"
Junio 15, 1993
USA

Advertising Section to Datamation
"How to revitalize host systems for client/server computing"
Junio 1, 1995
USA

Dr. Dobb's Especial Report
The interoperable objects revolution"
Marzo 1, 1995
USA

Software Magazine
"Client/Server computing"
Julio,1993
USA

PC Memo VAR/TIPS
"Arquitectura para los 90s, Cliente/Servidor"
Octubre, 1993
Mexico

Supplement to Information Week
"Client/Server'94"
Octubre, 1994
USA

Corporate Computing
"Client/Server Survival Guide"
Junio, 1993
USA